Ulkoinen muisti ja I/O:n toteutus

Muistihierarkia
Virtuaalimuisti

Kiintolevyt ja muut pyörivät levyt

I/O:n toteutus ja I/O:n tyypit
Laiteajuri ja laiteohjain

Tiedostojärjestelmä

Erilaiset levymuistit

Tällä luennolla käsitellään tietokonejärjestelmän ulkoisen muistin toteutus. Esittelemme ensin järjestelmän muistihierarkian ja virtuaalimuistijärjestelmän idean. Käymme sen jälkeen läpi kiintolevyjen peruspiirteet - niiden fyysisen rakenteen ja kuinka tieto talletetaan niihin. Esittelemme sitten, kuinka levyt ja kaikki muutkin I/O-laitteet liitetään järjestelmään ja kuinka käyttöjärjestelmä toteuttaa niiden käytön laiteajurien ja laiteohjaimien avulla. Lopuksi käymme läpi tiedostojärjestelmän ja tiedostopalvelimien pääpiirteet. Vilkaisemme myös erilaisten magneettisten ja optisten levyjen ominaisuuksia.

Muistihierarkia

Aika/tila -ongelma

- kaikki tieto ei mahdu keskusmuistiin
  - kaiken suoritusaikana tarvittavan tiedon pitää olla keskusmuistissa
- ulkoinen muisti on halvempaa (per tavu) kuin keskusmuisti
- ulkoinen muisti on hyvin paljon hitaampaa kuin keskusmuisti
  - liian hidasta suorittimelle

Aika/tila -optimointi

- suuret tietomäärät eivät mahdu keskusmuistiin, vaan ne pitää tallettaa ulkoiseen muistiin
- suoritusaikana tarvittavat pienet tietomäärät voidaan pitää keskusmuistissa
- kaiken suoritusaikana tarvittavan tiedon pitää olla keskusmuistissa
- tietoa tarvitsee kopioda keskusmuistin ja ulkoisen muistin välillä kesken suoritusta

Minkä tahansa ohjelman suoritusaikana tarvitaan huomattava määrä tietoa. Sekä itse ohjelmakoodin että kaiken suoritusaikana viitattavan datan pitää olla jossakin. Suoritusaikana viitattavan datan pitää olla hyvin nopeasti saatavilla, joten se ei voi olla missään hitaammalla medialla kuin keskusmuistissa. Perusongelmana on, että usein ohjelman tarvitsema muistitila on suurempi kuin mitä keskusmuistissa on tilaa saatavilla, joten kaikki suoritusaikana tarvittava tieto ei mahdu keskusmuistiin. Ehkä hyvinkin suuri osa ohjelman tarvitsemasta datasta täytyy siis tallettaa ulkoiseen muistiin.

Muistihierarkia

Aika/tila -ongelma

- kaikki tieto ei mahdu keskusmuistiin
  - kaiken suoritusaikana tarvittavan tiedon pitää olla keskusmu
- ulkoinen muisti on halvempaa (per tavu) kuin keskusmuisti
- ulkoinen muisti on hyvin paljon hitaampaa kuin keskusmuisti
  - liian hidasta suorittimelle

Aika/tila -optimointi

- suuret tietomäärät eivät mahdu keskusmuistiin, vaan ne pitää tallettaa ulkoiseen muistiin
- suoritusaikana tarvittavat pienet tietomäärät voidaan pitää keskusmuistissa
- kaiken suoritusaikana tarvittavan tiedon pitää olla keskusmuistissa
- tietoa tarvitsee kopioda keskusmuistin ja ulkoisen muistin välillä kesken suoritusta

No miksi emme vain rakentaisi tarpeeksi suurta keskusmuistia? Yleisesti ottaen tämä ei ole mahdollista. On helppo kuvitella yksinkertainenkin, pieni ohjelma, jonka suoritusaikana tarvitsema datan määrä voi kuitenkin olla valtaisa. Keskusmuistin koko on suhteellisen pieni, koska keskusmuistiteknologia on huomattavasti kalliimpaa kuin nykyisin käytössä olevat ulkoisen muistin teknologiat, esimerkiksi kovalevyt. Ulkoisen muistin etuna on halvempi hinta, mutta sen merkittävänä haittapuolena on hitaus. Tiedon haku kovalevyltä voi kestää miljoona kertaa niin kauan kuin keskusmuistista. Muistelkaapa vaikka 1. luennon juustokakkuesimerkkiä!

Muistihierarkia

Aika/tila -ongelma

- kaikki tieto ei mahdu keskusmuistiin
  - kaiken suoritusaikana tarvittavan tiedon pitää olla keskusmuistissa
- ulkoinen muisti on halvempaa (per tavu) kuin keskusmuisti
- ulkoinen muisti on hyvin paljon hitaampaa kuin keskusmuisti
  - liian hidasta suorittimelle

Aika/tila -optimointi

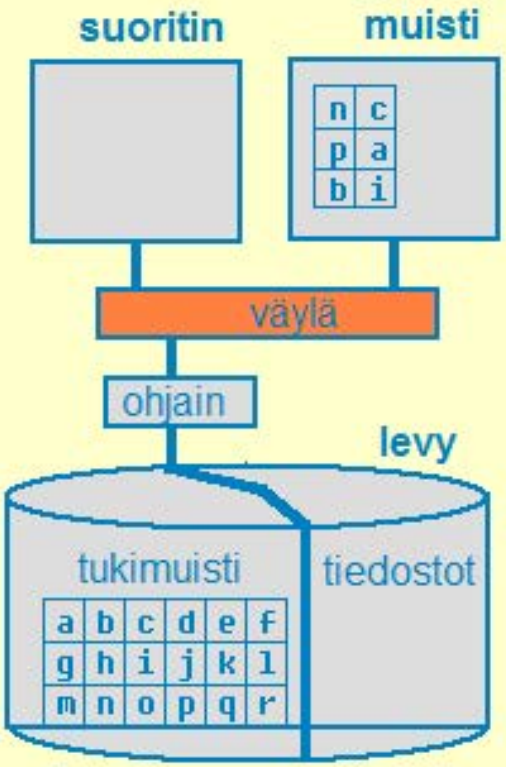
- suuret tietomäärät eivät mahdu keskusmuistiin, vaan ne pitää tallettaa ulkoiseen muistiin
- suoritusaikana tarvittavat pienet tietomäärät voidaan pitää keskusmuistissa
- kaiken suoritusaikana tarvittavan tiedon pitää olla keskusmuistissa
- tietoa tarvitsee kopioda keskusmuistin ja ulkoisen muistin välillä kesken suoritusta

Ratkaisuna aika/tila -ongelmaan on tietenkin optimointi. Kun ohjelman tarvitsema tieto ei kokonaisuudessaan mahdu keskusmuistiin, meidän tulee paloitella se sopivan kokoisiin osiin ja huolehtia siitä, että ohjelman ollessa suoritusvuorossa kaikki sillä hetkellä tarvittava tieto on keskusmuistissa. Tämä on tietenkin paljon helpompi sanoa kuin toteuttaa. Se myös tarkoittaa sitä, että aika ajoin ohjelman suoritus on keskeytettävä siksi aikaa, kun seuraavan suoritusjakson aikana tarvittavaa tietoa kopioidaan ulkoisesta muistista keskusmuistiin ja ehkä viime suoritusjakson aikana muutettua tietoa kopioidaan keskusmuistista ulkoiseen muistiin. Tähän voi kulua paljonkin aikaa.

Virtuaalimuisti

Automaattinen ratkaisu keskusmuistin ja ulkoisen muistin väliseen aika/tila -optimointiin

Ongelma: miten tehdä käytössä olevasta suoritusaikaisesta muistista yhtä nopea kuin keskusmuisti ja yhtä suuri kuin levymuisti?



Kaksitasoinen toteutus

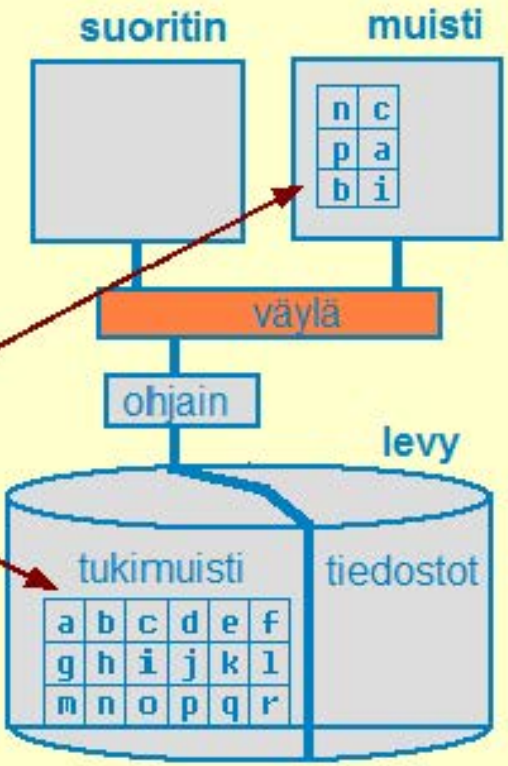
- kulloinkin käytössä olevat alueet ovat keskusmuistissa
- kaikki muistialueet ovat levymuistissa
- iso osa levymuistia on varattu tätä tarkoitusta varten
  - virtuaalimuistin tukimuistin levytila on pois tiedostoilta
- kopiointi levymuistin ja keskusmuistin välillä tarvittaessa
  - automaattista, käyttöjärjestelmä hoitaa
  - ohjelman suoritus on keskeytyksissä tällä aikaa

Virtuaalimuisti on automaattinen ratkaisu edellämainittuun ongelmaan, jossa jollain tavalla pitää huolehtia siitä, että keskusmuistissa on aina kulloinkin tarvittavat tiedot, vaikka suuri osa ohjelman tiedoista sijaitseekin levyllä. Virtuaalimuisti antaa mielikuvan järjestelmästä, jossa ohjelman käytettävissä oleva muistin määrä on kiintolevyn luokkaa ja jossa muistin nopeus on kuitenkin keskusmuistin luokkaa. Tämä on tietenkin vain käyttöjärjestelmän luomaa harhaa, jota käyttäjä eli ihminen ei oman hitautensa vuoksi useinkaan havaitse.

Virtuaalimuisti

Automaattinen ratkaisu keskusmuistin ja ulkoisen muistin väliseen aika/tila -optimointiin

Ongelma: miten tehdä käytössä olevasta suoritusaikaisesta muistista yhtä nopea kuin keskusmuisti ja yhtä suuri kuin levymuisti?



Kaksitasoinen toteutus

- kulloinkin käytössä olevat alueet ovat keskusmuistissa
- kaikki muistialueet ovat levymuistissa
- iso osa levymuistia on varattu tätä tarkoitusta varten
  - virtuaalimuistin tukimuistin levytila on pois tiedostoilta
- kopiointi levymuistin ja keskusmuistin välillä tarvittaessa
  - automaattista, käyttöjärjestelmä hoitaa
  - ohjelman suoritus on keskeytyksissä tällä aikaa

Virtuaalimuistin toteutus on kaksi-tasoinen. Kaikki ohjelman käyttämät tiedot eli ohjelman koko muistiavaruus on talletettuna kovalevylle omaan virtuaalimuistin partitioon eli tukimuistiin ja ainoastaan kulloinkin käytössä olevat muistialueet ovat kopioituna keskusmuistiin. Käyttöjärjestelmän virtuaalimuistijärjestelmä huolehtii tietojen kopionnista automaattisesti ja pyrkii ennustamaan, mitkä ohjelman tiedoista pitää kullakin hetkellä pitää keskusmuistissa. Tietojen kopioinnin aikana kyseistä ohjelma suorittava prosessi on odotustilassa ja suoritusvuoro on jollakin toisella sovellusprosessilla tai käyttöjärjestelmällä.

Virtuaalimuistin toteutus

Erilaisia tapoja osittaa muistiavaruus (viitattavien osoitteiden joukko) virtuaalimuistin toteuttamiseksi

- kanta- ja rajarekisterit, sivuttava virtuaalimuisti
- segmentointi, sivuttava segmentointi, muut yhdistelmät

Ohjelmassa olevat osoitteet eivät ole suoraan keskusmuistiosoitteita

- jokainen käytetty osoite täytyy muuttaa suoritusaikana keskusmuistiosoitteeksi

Pääosa toteutuksesta ohjelmistotasolla

- kunkin prosessin muistialueiden kirjanpito, levyllä olevan tukimuistin kirjanpito
- tietojen kopiointi keskusmuistin ja tukimuistin välillä

Tärkeä osa toteutuksesta laitteistotasolla

- laitteistotuki MMU:ssa (muistinhallintayksikössä)
- TLB tekee osoitteenmuunnoksen keskusmuistisoitteeksi useimmiten laitteistolla
- MMU tukee valittua virtuaalimuistin toteutustapaa

Virtuaalimuistin toteutustapoja on useita erilaisia ja käymme tässä nyt läpi vain muutaman perustapauksen. Yksi tapa toteuttaa virtuaalimuisti on käyttää ttk-91 -koneen tapaan kanta- ja rajarekistereitä, usein kyllä useampia kuin yhtä paria. Usein kanta- ja rajarekisteripareja on useita, joista jokainen speksaa jonkin sortin muistiavaruuden osan. Huonona puolena tässä on, että yhden rekisteriparin määrittelemän muistialueen tulee olla keskusmuistissa yhtenäisenä kokonaisuutena ja että tällaiset kokonaisuudet ovat suhteellisen suuria ja eri kokoisia.

Virtuaalimuistin toteutus

Erilaisia tapoja osittaa muistiavaruus (viitattavien osoitteiden joukko) virtuaalimuistin toteuttamiseksi

- kanta- ja rajarekisterit, sivuttava virtuaalimuisti
- segmentointi, sivuttava segmentointi, muut yhdistelmät

Ohjelmassa olevat osoitteet eivät ole suoraan keskusmuistiosoitteita

- jokainen käytetty osoite täytyy muuttaa suoritusaikana keskusmuistiosoitteeksi

Pääosa toteutuksesta ohjelmistotasolla

- kunkin prosessin muistialueiden kirjanpito, levyllä olevan tukimuistin kirjanpito
- tietojen kopiointi keskusmuistin ja tukimuistin välillä

Tärkeä osa toteutuksesta laitteistotasolla

- laitteistotuki MMU:ssa (muistinhallintayksikössä)
- TLB tekee osoitteenmuunnoksen keskusmuistiosoitteeksi useimmilla laitteistolla
- MMU tukee valittua virtuaalimuistin toteutustapaa

Yleisin virtuaalimuistin toteutustapa on sivutus. Siinä keskusmuisti jaetaan saman kokoisiin pienehköihin sivukehyksiin, joihin kuhunkin voidaan sitten sijoittaa minkä tahansa prosessin mikä tahansa tämän sivukehyksen kokoinen muistialue eli sivu. Prosessin muistiavaruus on siis jaettu samankokoisiin sivuihin, ja osa niistä on sijoitettuna keskusmuistiin vapaana olleisiin sivukehyksiin. Virtuaalimuistin sivutaulut pitävät kirjaa, missä päin keskusmuistia kukin muistiavaruuden sivu kulloinkin sijaitsee vai onko kyseinen muistiavaruuden sivu talletettuna ainoastaan levylle virtuaalimuistin tukimuistiin.

Virtuaalimuistin toteutus

Erilaisia tapoja osittaa muistiavaruus (viitattavien osoitteiden joukko) virtuaalimuistin toteuttamiseksi

- kanta- ja rajarekisterit, sivuttava virtuaalimuisti
- segmentointi, sivuttava segmentointi, muut yhdistelmät

Ohjelmassa olevat osoitteet eivät ole suoraan keskusmuistiosoitteita

- jokainen käytetty osoite täytyy muuttaa suoritusaikana keskusmuistiosoitteeksi

Pääosa toteutuksesta ohjelmistotasolla

- kunkin prosessin muistialueiden kirjanpito, levyllä olevan tukimuistin kirjanpito
- tietojen kopiointi keskusmuistin ja tukimuistin välillä

Tärkeä osa toteutuksesta laitteistotasolla

- laitteistotuki MMU:ssa (muistinhallintayksikössä)
- TLB tekee osoitteenmuunnoksen keskusmuistisoitteeksi useimmiten laitteistolla
- MMU tukee valittua virtuaalimuistin toteutustapaa

Toinen vaihtoehto on osittaa muistiavaruus loogisiksi kokonaisuuksiksi, isohkoiksi segmenteiksi, joista kukin sitten talletetaan yhtenäiselle muistialueelle keskusmuistiin tarvittaessa. Menetelmän huonona puolena keskusmuistin hallinta eri kokoisten muistialueiden vuoksi, mutta hyvänä puolena loogisten kokonaisuuksien pitäminen yhdessä. Näiden lisäksi käytössä on vielä näiden eri menetelmien erilaisia useamman tasoisia yhdistelmiä, joiden avulla virtuaalimuistiratkaisuista tulee tehokkaampia mutta myös monimutkaisempia.

Virtuaalimuistin toteutus

Erilaisia tapoja osittaa muistiavaruus (viitattavien osoitteiden joukko) virtuaalimuistin toteuttamiseksi

- kanta- ja rajarekisterit, sivuttava virtuaalimuisti
- segmentointi, sivuttava segmentointi, muut yhdistelmät

Ohjelmassa olevat osoitteet eivät ole suoraan keskusmuistiosoitteita

- jokainen käytetty osoite täytyy muuttaa suoritusaikana keskusmuistiosoitteeksi

Pääosa toteutuksesta ohjelmistotasolla

- kunkin prosessin muistialueiden kirjanpito, levyllä olevan tukimuistin kirjanpito
- tietojen kopiointi keskusmuistin ja tukimuistin välillä

Tärkeä osa toteutuksesta laitteistotasolla

- laitteistotuki MMU:ssa (muistinhallintayksikössä)
- TLB tekee osoitteenmuunnoksen keskusmuistisoitteeksi useimmiten laitteistolla
- MMU tukee valittua virtuaalimuistin toteutustapaa

Yhteistä kaikille virtuaalimuistin toteutustavoille on, että ohjelman käyttämät tiedon osoitteet ovat vain virtuaalisia, eikä niistä suoraan näe, missä päin keskusmuistia kyseinen tieto on talletettuna, jos missään. Aina tietoon viitattaessa järjestelmän pitää ensin selvittää, (a) onko kyseinen tieto keskusmuistissa vai ei ja (b) missä päin keskusmuistia tieto sijaitsee. Jos tieto ei sijaitse keskusmuistissa, ohjelman suoritus pitää keskeyttää, kunnes tieto on haettu tukimuistista keskusmuistiin. Tottakai virtuaalimuistijärjestelmän taulukoista löytyy sitten myös tieto siitä, missä päin tukimuistia haluttu tieto tällöin sijaitsee.

Virtuaalimuistin toteutus

Erilaisia tapoja osittaa muistiavaruus (viitattavien osoitteiden joukko) virtuaalimuistin toteuttamiseksi

- kanta- ja rajarekisterit, sivuttava virtuaalimuisti
- segmentointi, sivuttava segmentointi, muut yhdistelmät

Ohjelmassa olevat osoitteet eivät ole suoraan keskusmuistiosoitteita

- jokainen käytetty osoite täytyy muuttaa suoritusaikana keskusmuistiosoitteeksi

Pääosa toteutuksesta ohjelmistotasolla

- kunkin prosessin muistialueiden kirjanpito, levyllä olevan tukimuistin kirjanpito
- tietojen kopiointi keskusmuistin ja tukimuistin välillä

Tärkeä osa toteutuksesta laitteistotasolla

- laitteistotuki MMU:ssa (muistinhallintayksikössä)
- TLB tekee osoitteenmuunnoksen keskusmuistisoitteeksi useimmiten laitteistolla
- MMU tukee valittua virtuaalimuistin toteutustapaa

Pääosa virtuaalimuistin toteutuksesta on ohjelmistotasolla ja virtuaalimuistijärjestelmä muodostaakin merkittävän osan jokaista nykyistä käyttöjärjestelmää. Aina kun virtuaalimuistijärjestelmää todella tarvitaan, niin tietoa täytyy kopioida levymuistin ja keskusmuistin välillä. Tähän kuluu niin paljon aikaa, että tuhansienkin konekäskyjen suoritus virtuaalimuistin toteutuksessa ei tunnu vielä juuri missään.

Virtuaalimuistin toteutus

Erilaisia tapoja osittaa muistiavaruus (viitattavien osoitteiden joukko) virtuaalimuistin toteuttamiseksi

- kanta- ja rajarekisterit, sivuttava virtuaalimuisti
- segmentointi, sivuttava segmentointi, muut yhdistelmät

Ohjelmassa olevat osoitteet eivät ole suoraan keskusmuistiosoitteita

- jokainen käytetty osoite täytyy muuttaa suoritusaikana keskusmuistiosoitteeksi

Pääosa toteutuksesta ohjelmistotasolla

- kunkin prosessin muistialueiden kirjanpito, levyllä olevan tukimuistin kirjanpito
- tietojen kopiointi keskusmuistin ja tukimuistin välillä

Tärkeä osa toteutuksesta laitteistotasolla

TLB = Translation Lookaside Buffer

- laitteistotuki MMU:ssa (muistinhallintayksikössä)
- TLB tekee osoitteenmuunnoksen keskusmuistisoitteeksi useimmiten laitteistolla
- MMU tukee valittua virtuaalimuistin toteutustapaa

Virtuaalimuistijärjestelmä ei kuitenkaan voi toimia ilman MMU:ssa olevaa laitteistotukea. Ajatelkaapa vain kuinka paljon ttk-91 -koneen suoritus hidastuisi, jos jokaisen konekäskyn jokaisen muistiviitteen kohdalla tapahtuva rajatarkistus ja kantarekisterin arvon lisäys muistisoitteen saamiseksi tehtäisiin tavallisilla konekäskyillä sen sijaan, että ne tapahtuvat laitteistotasolla MMU:ssa. Vastaavalla tavalla MMU:ssa on TLB (Translation Lookaside Buffer) esimerkiksi sivuttavan virtuaalimuistin osoitteenmuunnosten tekemiseen laitteiston avustuksella. TLB toimii vähän välimuistin tapaan siten, että sieltä löytyvät nopeasti kaikki viime aikoina tehdyt osoitteenmuutokset. Muut sitten kestävät kauemmin.

Virtuaalimuistin toteutus

Erilaisia tapoja osittaa muistiavaruus (viitattavien osoitteiden joukko) virtuaalimuistin toteuttamiseksi

- kanta- ja rajarekisterit, sivuttava virtuaalimuisti
- segmentointi, sivuttava segmentointi, muut yhdistelmät

Ohjelmassa olevat osoitteet eivät ole suoraan keskusmuistiosoitteita

- jokainen käytetty osoite täytyy muuttaa suoritusaikana keskusmuistiosoitteeksi

Pääosa toteutuksesta ohjelmistotasolla

- kunkin prosessin muistialueiden kirjanpito, levyllä olevan tukimuistin kirjanpito
- tietojen kopiointi keskusmuistin ja tukimuistin välillä

Tärkeä osa toteutuksesta laitteistotasolla

- laitteistotuki MMU:ssa (muistinhallintayksikössä)
- TLB tekee osoitteenmuunnoksen keskusmuistisoitteeksi useimmiten laitteistolla
- MMU tukee valittua virtuaalimuistin toteutustapaa

Tietokoneen rakenne -kurssi

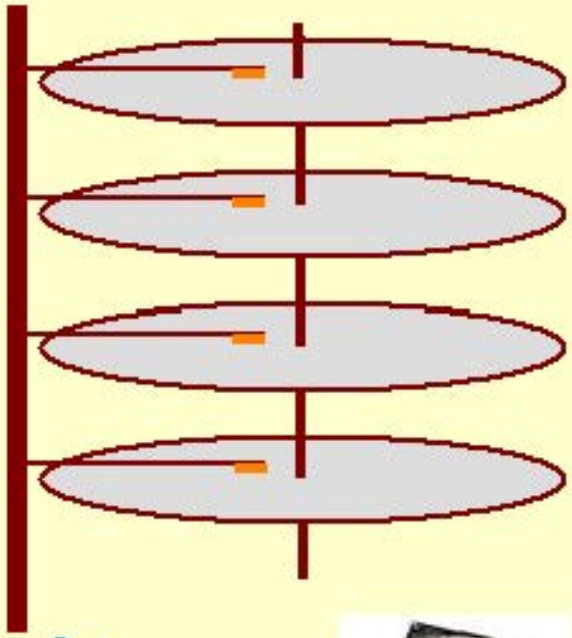
Käyttöjärjestelmäkurssit

Virtuaalimuistia ei käsitellä tällä kurssilla tämän tarkempaa. Lisätietoja laitteistotuesta eli TLB:n toteutuksesta saa kurssilla Tietokoneen rakenne. Virtuaalimuistijärjestelmän ohjelmistoa käsitellään taas tyhjentävämmin käyttöjärjestelmäkursseilla.

Magneettinen levymuisti eli kiintolevy

Levykkö 0.15-1000 GB/levykkö

- pyörii nopeasti 3600-10800 rpm
- yksi tai useampi levyä 1-16 levyä/levykkö
- joka levypinnalla oma luku/kirjoituspää
- kaikki luku/kirjoituspäät samassa varressa, liikkuvat yhtä aikaa



Levypinta

- 2 per levy (joskus 1)
- samankeskisiä uria
- päällekkäiset urat muodostavat sylinterin
- yhdellä uralla usea sektori
  - pienin kerralla luettava/kirjoitettava alue

Levykössä on päällekkäin yksi tai useampi magneettisesti käsiteltävä levy. Tieto talletetaan bitteinä levyn pinnalle magnetismin avulla. Levyt pyörivät yleensä koko ajan vakionopeudella ja tietoa luetaan ja kirjoitetaan hakuvarren päässä olevalla luku/kirjoituspäällä. Joka levypinnalle on yleensä vain yksi luku/kirjoituspää. Kaikki yhden levykön hakuvarret ovat kiinni samassa varressa, jolloin ne liikkuvat kaikki aina yhtä aikaa. Yhdellä kertaa voidaan yleensä lukea/kirjoittaa kuitenkin vain yhdestä luku/kirjoituspäästä.

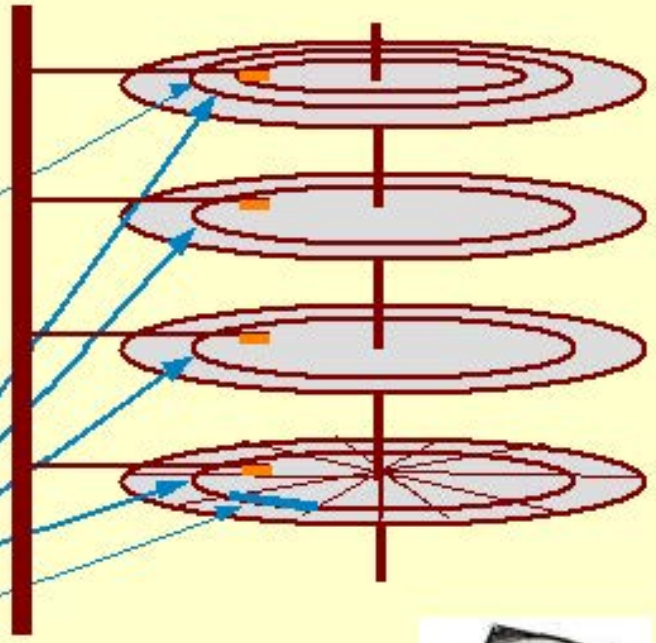
Magneettinen levymuisti eli kiintolevy

Levykkö

- pyörii nopeasti
- yksi tai useampi levyä
- joka levypinnalla oma luku/kirjoituspää
- kaikki luku/kirjoituspäät samassa varressa, liikkuvat yhtä aikaa

Levypinta

- 2 per levy (joskus 1)
- samankeskisiä uria 2000-3000 uraa/pinta
- päällekkäiset urat muodostavat sylinterin
- yhdellä uralla usea sektori 20-100 sekt/ura
  - pienin kerralla luettava/kirjoitettava alue 0.5-8KB/sektori



Levykön kullakin levyllä on yleensä molemmat pinnat käytössä, mutta isoissa levyköissä ulommaiset pinnat voivat olla käyttämättöminä. Kunkin levypinnan päällekkäin olevat urat muodostavat ns. sylinterin, koska niissä olevaa tietoa voidaan lukea/kirjottaa samalla hakuvarren asennolla. Hakuvarren liikuttelu vie merkittävästi aikaa, joten tällä seikalla on merkitystä levykön käytön optimoinnin kannalta. Kukin ura on jaettu sektoreihin, jotka ovat pienimpiä yksiköitä, joita levyltä voi kerralla lukea tai kirjoittaa. Levyltä ei esimerkiksi voi lukea vain 10 tavua jostain kohtaan, vaan luku aina kohdistuu yhteen tai useampaan sektoriin kerrallaan.

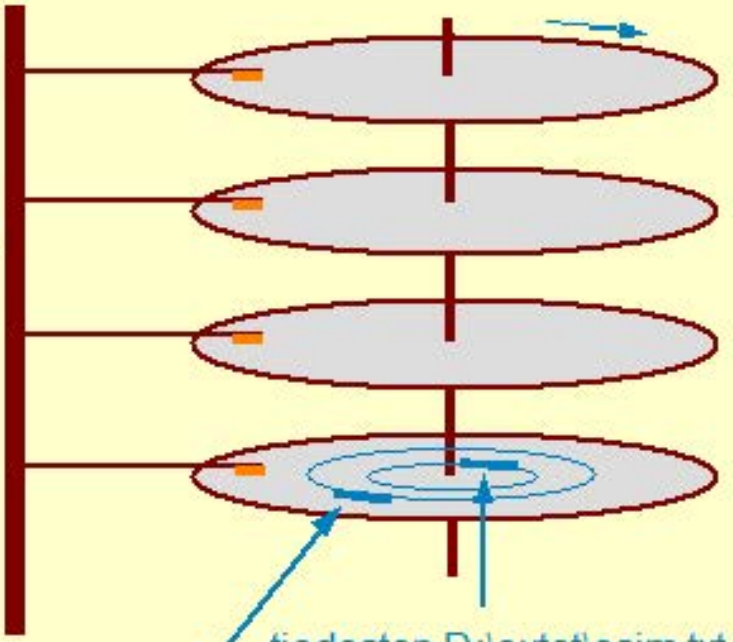
Levymuistin saantiaika

Tiedon osoite: levypinta + ura + sektori

- laiteajuri etsii tiedostojenhallinnan taulukoista tiedoston loogisen osoitteen (esim. D:\autot\esim.txt) perusteella

Saantiaika access time

- hakuvarren siirtoaika seek time
  - esim. 2-15 ms, keskimäärin 6.3 ms
- pyörähdysviive
  - esim. 8.33 ms rotational delay
  - satunnaiselle sektorille puolen kierroksen pyörähdysaika
- tiedon siirtoaika data transfer time
  - esim. 0.42 ms
  - pyörähdysaika / sektorien lkm



Tiedon hakeminen levyltä kestää hyvin kauan, millisekunteja. Kovalevy on liki ainoa osa tavallisessa pöytäkoneessa, missä on liikkuvia osia erilaisten tuulettimien lisäksi. Jotta tieto voitaisiin esimerkiksi lukea levyltä, meidän tulee ensin tietää luettavan sektorin osoite. Osoite kuvataan kolmikkona levypinta, ura ja sektori, jotka kaikki löytyvät käyttöjärjestelmän tiedostojärjestelmän taulukoista ohjelman käyttämän tiedostonimen perusteella. Jos tiedostonimessä on hakemistoja, niin nämä joudutaan ehkä ensin lukemaan levyltä halutun sektorin osoitteen löytämiseksi.

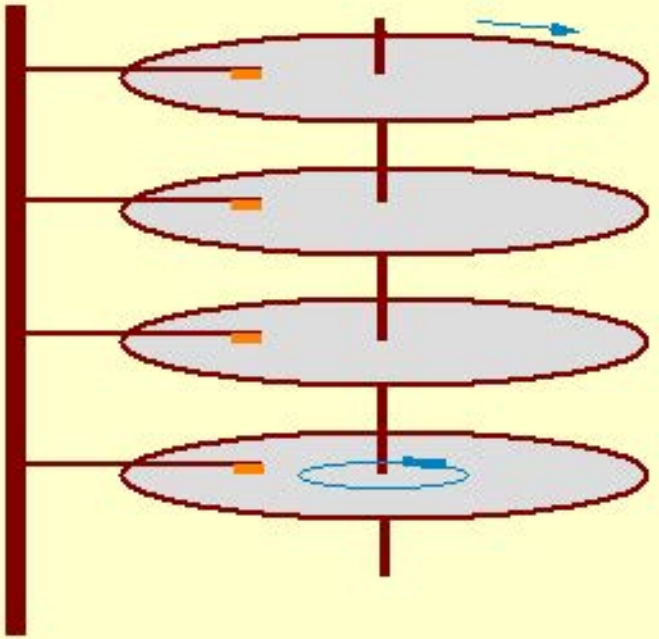
Levymuistin saantiaika

Tiedon osoite: levypinta + ura + sektori

- laiteajuri etsii tiedostojenhallinnan taulukoista tiedoston loogisen osoitteen (esim. D:\autot\esim.txt) perusteella

Saantiaika access time

- hakuvarren siirtoaika seek time
  - esim. 2-15 ms, keskimäärin 6.3 ms
- pyörähdysviive
  - esim. 8.33 ms rotational delay
  - satunnaiselle sektorille puolen kierroksen pyörähdysaika
- tiedon siirtoaika data transfer time
  - esim. 0.42 ms
  - pyörähdysaika / sektorien lkm

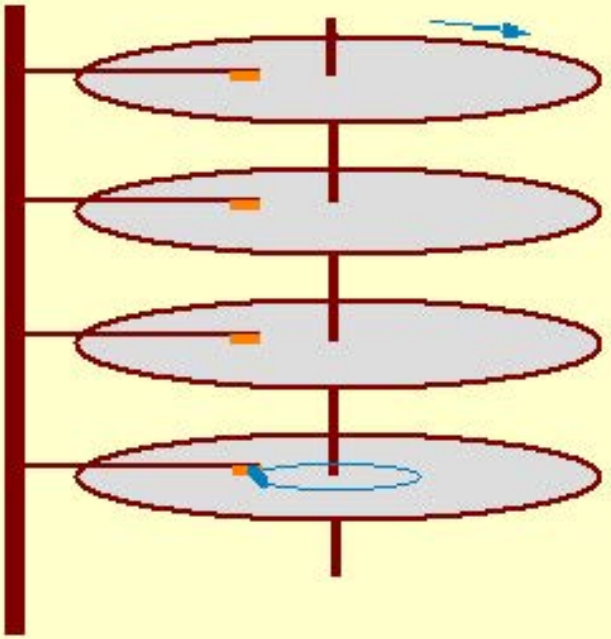


Oletetaan nyt, että halutun sektorin osoite tiedossa. Kaikista levypinnoista voidaan lukea yhtä helposti samoilla asetuksilla, joten sektorin levypinnalla ei alkuvaiheessa ole merkitystä. Ensimmäisenä meidän tulee siirtää hakuvarsi oikealle uralle. Tämä kuulostaa helpolta, mutta ei ole sitä. Levykössä vallitsee lähes tyhjiö, mutta ei ihan. Levykön sisällä pyörivät levyt saavat sisällä olevan kaasun pyörimään hyvin nopeasti, kun taas hakuvarret ovat paikallaan. Hakuvarret käyttäytyvät kaasuvirrassa kuten nopeasti lentävät lentokoneet ja niiden aerodynamiikkaa tutkitaan samoilla menetelmillä. Hakuvarret kiihtyvät huippuropeuteensa, matkaavat kohteen lähelle ja lopulta hidastavat ja pysähtyvät oikean uran kohdalle. Yhdenkin uran matkaamiseen kuluu helposti 2 ms.

Levymuistin saantiaika

Tiedon osoite: levypinta + ura + sektori

- laiteajuri etsii tiedostojenhallinnan taulukoista tiedoston loogisen osoitteen (esim. D:\autot\esim.txt) perusteella



Saantiaika access time

- hakuvarren siirtoaika seek time
  - esim. 2-15 ms, keskimäärin 6.3 ms
- pyörähdysviive
  - esim. 8.33 ms rotational delay
  - satunnaiselle sektorille puolen kierroksen pyörähdysaika
- tiedon siirtoaika data transfer time
  - esim. 0.42 ms
  - pyörähdysaika / sektorien lkm

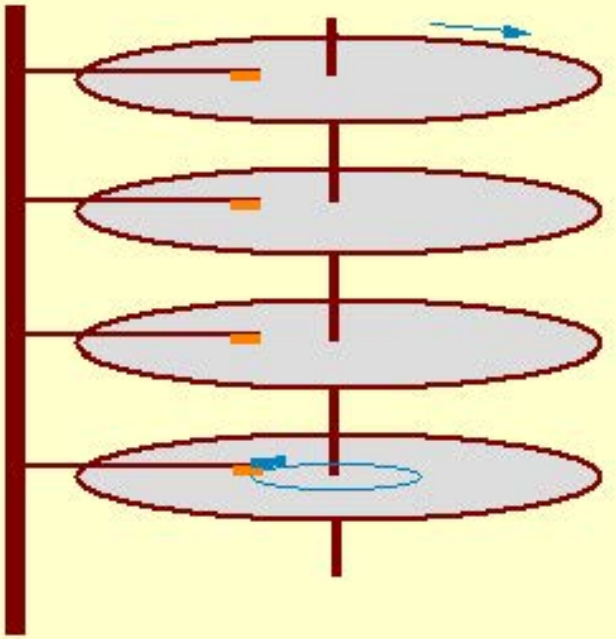
3600 rpm → 60 rps → 1/60s = 16.66 ms/kierros

Nyt kun hakuvarsi on paikallaan, niin meidän tulee odottaa, että haluttu sektori on pyörähtänyt lukupään kohdalle. Jos sektori on valittu satunnaisesti, niin tähän kuluu keskimäärin puolen kierroksen pyörähdysviiveen vaatima aika. Esimerkiksi, jos pyörähdysnopeus on 3600 rpm, niin yhteen kierrokseen kuluu 16.66 ms ja siten puoleen kierrokseen 8.33 ms. Tottakai levyn käyttöä pyritään optimoimaan, joten usein päästään pienempään pyörähdysviiveeseen kuin mitä tämä satunnainen sektori antaisi.

Levymuistin saantiaika

Tiedon osoite: levypinta + ura + sektori

- laiteajuri etsii tiedostojenhallinnan taulukoista tiedoston loogisen osoitteen (esim. D:\autot\esim.txt) perusteella



Saantiaika access time

- hakuvarren siirtoaika seek time
  - esim. 2-15 ms, keskimäärin 6.3 ms
- pyörähdysviive
  - esim. 8.33 ms rotational delay
  - satunnaiselle sektorille puolen kierroksen pyörähdysaika
- tiedon siirtoaika data transfer time
  - esim. 0.42 ms
  - pyörähdysaika / sektorien lkm

3600 rpm → 60 rps → 1/60s = 16.66 ms/kierros

16.66 ms / 40 sektoria = 0.42 ms

Lopuksi sektori pitää tietenkin vielä lukea. Lukemiseen kuluva aika on jälleen suoraan verrannollinen pyörähdysaikaan. Lukuaika on se aika, mikä kuluu kun sektori pyörähtää lukupään ohitse. Lukemiseen käytettävä lukupää valitaan luettavan sektorin levypinnan mukaan. Koko sektorin saantiaika (access time) koostuu siis pääasiassa muista viipeistä kuin itse tiedon siirrosta. Tämän vuoksi usein tehdäänkin niin, että samalla kertaa luetaan useampi kuin yksi sektori. Esimerkissä käsiteltiin sektorin lukemista. Kirjoittaminen tapahtuu tietenkin täysin samalla tavalla.

Tiedoston talletus levylle

1 sektori/lohko
lohko 1 KB
tiedosto 3.6 KB
tilanvaraus 4 KB

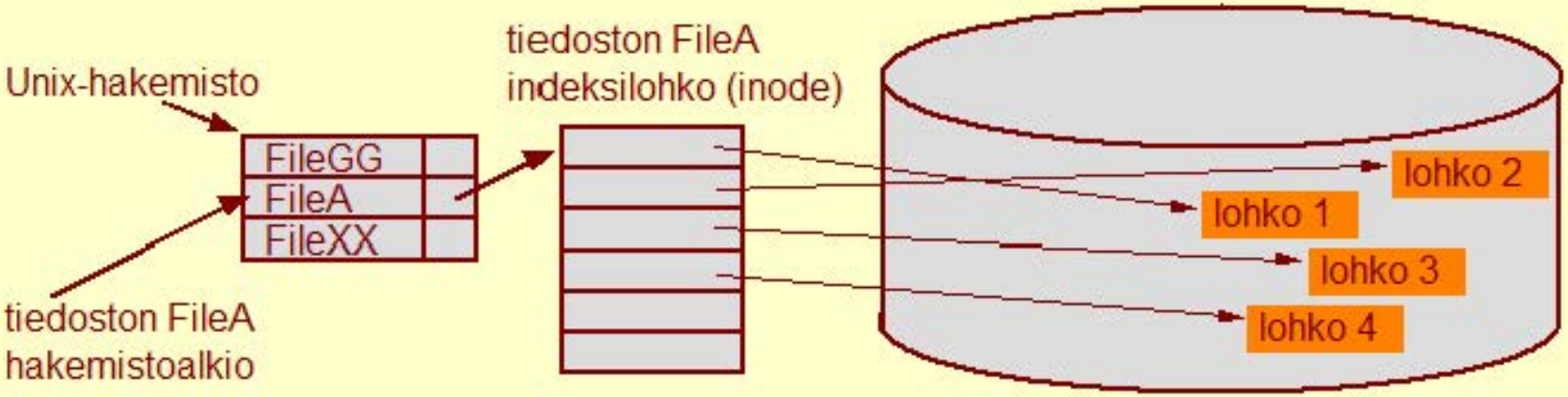
Tiedosto koostuu useista lohkoista (disk block)

- lohko koko on levysektorin koon monikerta
  - lohkoon kuuluvat sektorit ovat peräkkäisiä, yhdellä kertaa luettavia
- tiedoston viimeinen lohko on vain osittain käytössä
  - keskimääräinen hukkatila on puoli sektoria

hukkatilaa 0.4 KB

Hakemistossa on on tieto tiedoston käyttämistä lohkoista

- lohkot luetaan/kirjoitetaan annetussa järjestyksessä



Kukin tiedosto koostuu yhdestä tai useammasta lohkosta. Kukin tiedostolohko on yhden tai useamman levysektorin pituinen. Levytilaa varataan kokonaisina lohkoina aina vähän enemmän kuin tarvitaan ja keskimäärin joka tiedoston kohdalla levytilaa menee 'hukkaan' puoli sektoria. Tällä on paljon merkitystä, kun sektorin kokoa haluttaisiin nopeussyistä kasvattaa, esimerkiksi kokoon 16 tai 64 KB. Samaan tiedostoon kuuluvat sektorit pyritään sijoittamaan levylle niin, että niiden käsittely olisi nopeata. Niiden ei kuitenkaan tarvitse olla peräkkäisiä sektoreita taltiolla - jos ne eivät ole, niin tiedoston käsittely vaan kestään vähän kauemmin.

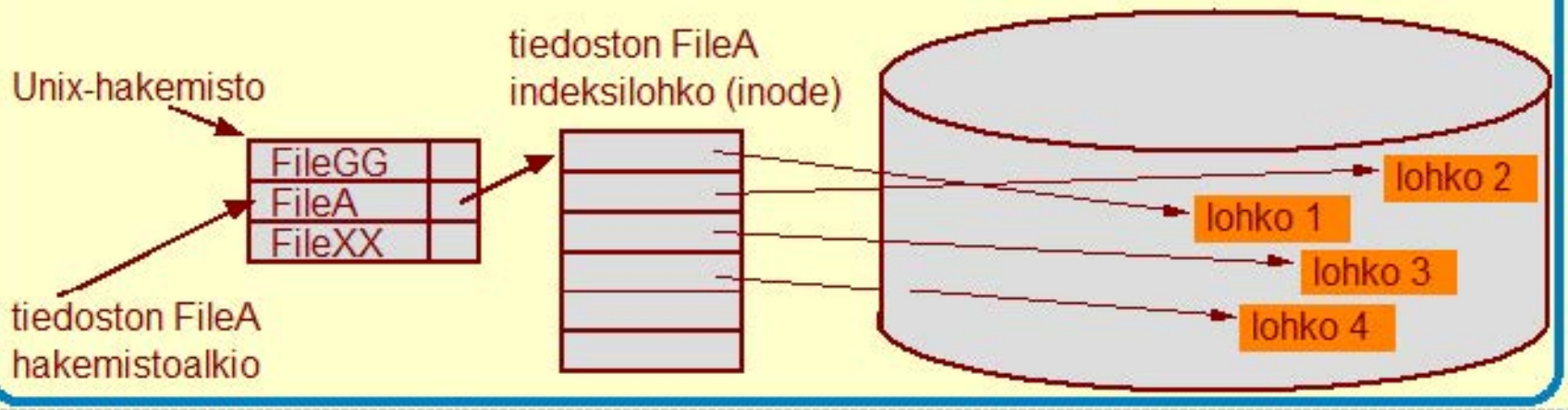
Tiedoston talletus levylle

Tiedosto koostuu useista lohkoista (disk block)

- lohko koko on levysektorin koon monikerta
  - lohkoon kuuluvat sektorit ovat peräkkäisiä, yhdellä kertaa luettavia
- tiedoston viimeinen lohko on vain osittain käytössä
  - keskimääräinen hukkatila on puoli sektoria

Hakemistossa on on tieto tiedoston käyttämistä lohkoista

- lohkot luetaan/kirjoitetaan annetussa järjestyksessä



Eri tiedostojärjestelmät pitävät kirjaa tiedostojen sijainneista eri tavoin. Yhden käyttöjärjestelmän sisällä voidaan käyttää useita erilaisia tiedostojärjestelmiä, joista kukin huolehtii kirjanpidosta omalla tavallaan. Unixissa on usein käytössä menetelmä, jossa hakemistot muodostuvat hakemistoalkioista, joissa on kustakin tiedostosta niiden nimen lisäksi muuta hallintotietoa ja viite kyseisen tiedoston indeksilohkoon. Indeksilohkossa on lueteltuna kaikkien sen tiedoston käyttämien levylohkojen osoitteet. Tällä tavoin tiedoston levylohkot voidaan sitten lukea levyltä yksi kerrallaan tai vaikkapa kaikki varmuuden vuoksi yhdellä kertaa muistipuskuriin käsittelyä varten.

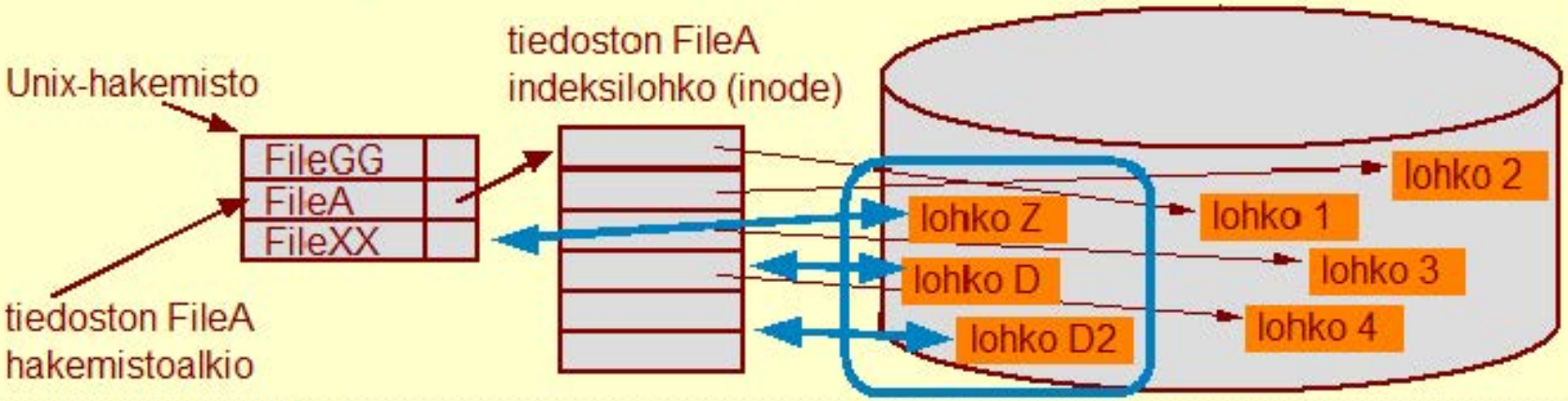
Tiedoston talletus levylle

Tiedosto koostuu useista lohkoista (disk block)

- lohko koko on levysektorin koon monikerta
  - lohkoon kuuluvat sektorit ovat peräkkäisiä, yhdellä kertaa luettavia
- tiedoston viimeinen lohko on vain osittain käytössä
  - keskimääräinen hukkatila on puoli sektoria

Hakemistossa on on tieto tiedoston käyttämistä lohkoista

- lohkot luetaan/kirjoitetaan annetussa järjestyksessä



Hakemistot ja indeksilohkot ovat tietenkin myös alkuaan sijainneet levyllä. Hakemisto on ihan tavallinen tiedosto, mutta sen sisäinen rakenne on vain ennalta määritelty. Indeksilohko voidaan samalla tavalla tallettaa tavallisena tiedostona, mutta myös sen rakenne on ennaltamäärätty ja ainoastaan käyttöjärjestelmän tiedostojenhallintarutiinit voivat sitä käsitellä. Jos tiedoston indeksilohko tuhoutuu vaikkapa levyvirheen vuoksi, niin kyseisen tiedoston lohkoihin ei enää pääse käsiksi, vaikka ne kaikki vielä olisivatkin hyvässä tallessa levyllä. Tämän vuoksi indeksilohko on usein talletettu varmuuden vuoksi kahteen kertaan levylle.

Tiedostot vs. virtuaalimuistin tukimuisti

Levy voi olla ositettu useaan eri osioon (partitioon)

- kukin osio käyttää omaa tiedostojärjestelmäänsä

Käyttöjärjestelmä on yhdessä partitiossa

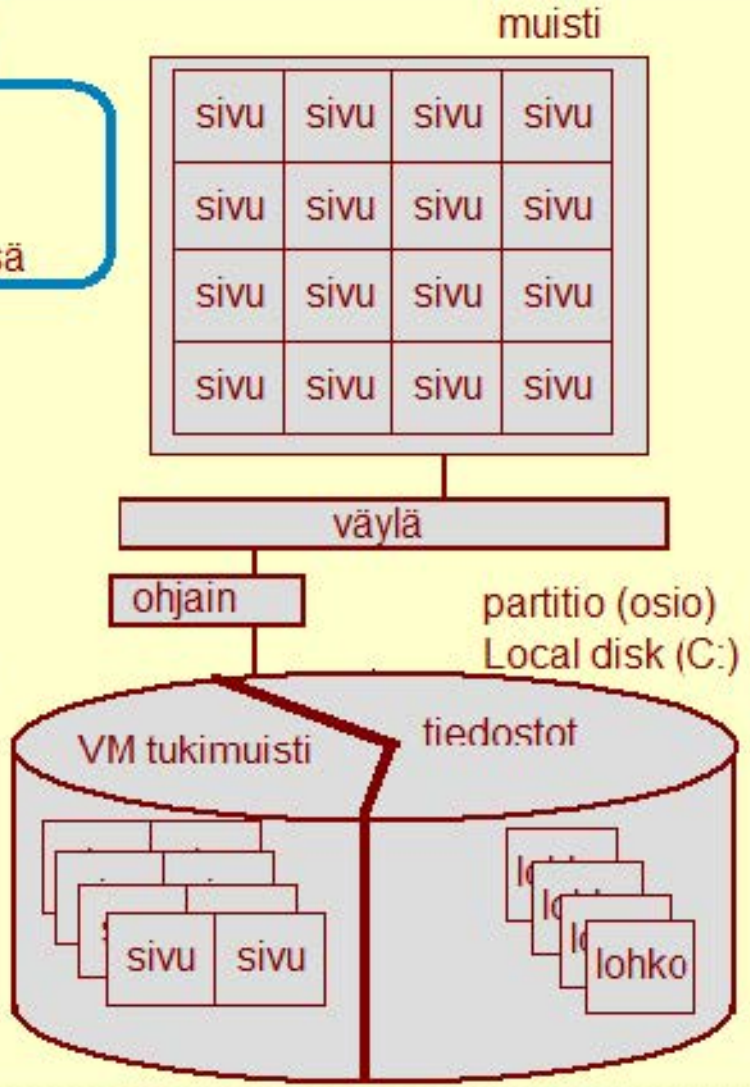
- samaa osiota voi käyttää myös tiedostoihin
- virtuaalimuistin tukimuisti

Virtuaalimuistin voi rakentaa tiedostojärjestelmän päälle

- virtuaalimuistin sivukehys on yksi tai useampia levylohkoja

Tiedostojärjestelmän voi rakentaa virtuaalimuistin päälle

- levylohko on yksi tai useampi virtuaalimuistin sivukehystä
- muistiinkuvattu tiedosto



Kukin kovalevy on ositettu yhteen tai useampaan partitioon, joista kuhunkin voidaan alustaa oma tiedostojärjestelmänsä. Esimerkiksi Windows-koneen levy voi olla ositettu kahteen partitioon, joista toinen ('Local disk (C:)') voi olla vanhempi FAT-partitio ja toinen ('user (D:)') uudemman teknologian NTFS-partitio. Käyttöjärjestelmä on toteutettu yhteen partitioon ('Local disk (C:)') ja muut partitiot voivat olla käyttäjien tiedostoja tai jotain muuta tarkoitusta varten. On myös mahdollista, että joku toinenkin partitio ('Linux (E:)') sisältää käyttöjärjestelmän, jolloin järjestelmän alustuksen yhteydessä voidaan valita, kumpaa käyttöjärjestelmää halutaan sillä kertaa käyttää.

Tiedostot vs. virtuaalimuistin tukimuisti

Levy voi olla ositettu useaan eri osioon (partitioon)

- kukin osio käyttää omaa tiedostojärjestelmäänsä

Käyttöjärjestelmä on yhdessä partitiossa

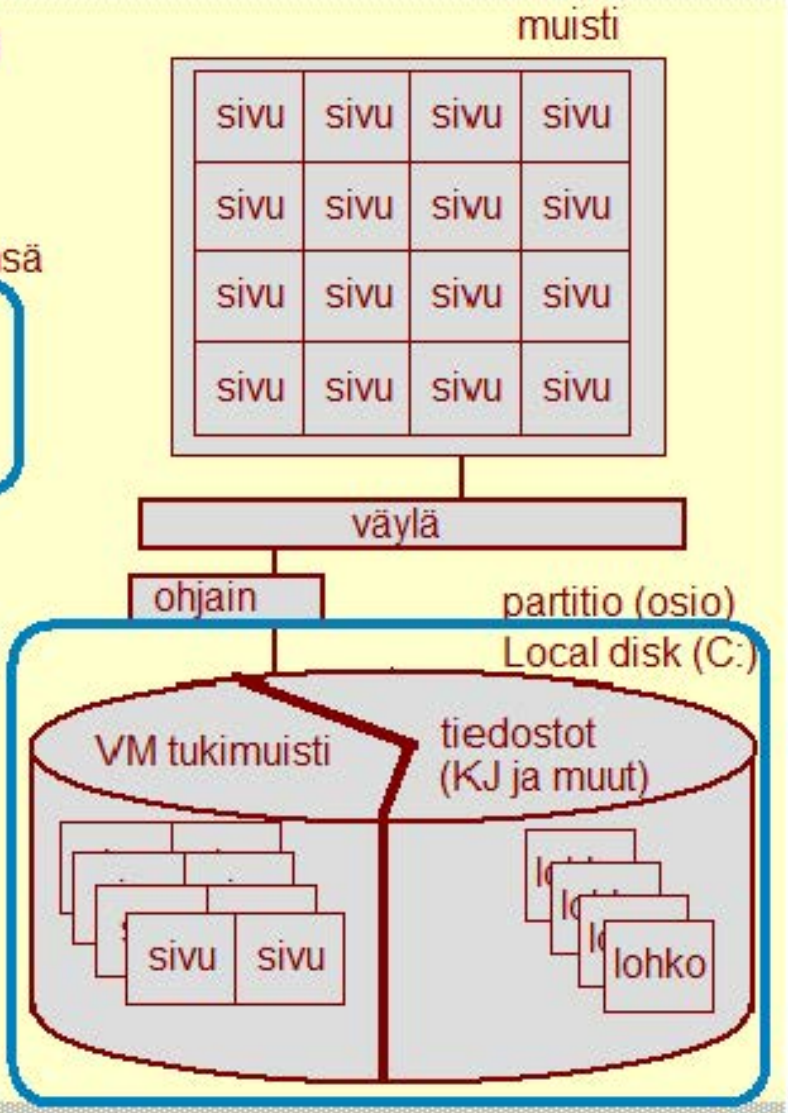
- samaa osiota voi käyttää myös tiedostoihin
- virtuaalimuistin tukimuisti

Virtuaalimuistin voi rakentaa tiedostojärjestelmän päälle

- virtuaalimuistin sivukehys on yksi tai useampia levylohkoja

Tiedostojärjestelmän voi rakentaa virtuaalimuistin päälle

- levylohko on yksi tai useampi virtuaalimuistin sivukehystä
- muistiinkuvattu tiedosto



Käyttöjärjestelmän levypartitiota voidaan käyttää myös tavallisiin tiedostoihin, mutta usein käyttöjärjestelmä halutaan erottaa omaksi partitiokseen, jolloin sen levytilan hallinta on helpompaa. Käyttöjärjestelmän partitiossa suuri osa on varattu virtuaalimuistin tukimuistille, koska kaikkien suoritettavana olevien prosessien koko muistiavaruuksien täytyy mahtua sinne. Käyttöjärjestelmän muistinhallinta yleensä säätelee tukimuistin kokoa automaattisesti, mutta sitä voi myös säätää käsin, jos osaa ja uskaltaa. Käyttöjärjestelmä voi myös dynaamisesti suoritusaikana kasvattaa tukimuistin kokoa, ainakin joissakin käyttöjärjestelmissä.

Tiedostot vs. virtuaalimuistin tukimuisti

Levy voi olla ositettu useaan eri osioon (partitioon)

- kukin osio käyttää omaa tiedostojärjestelmäänsä

Käyttöjärjestelmä on yhdessä partitiossa

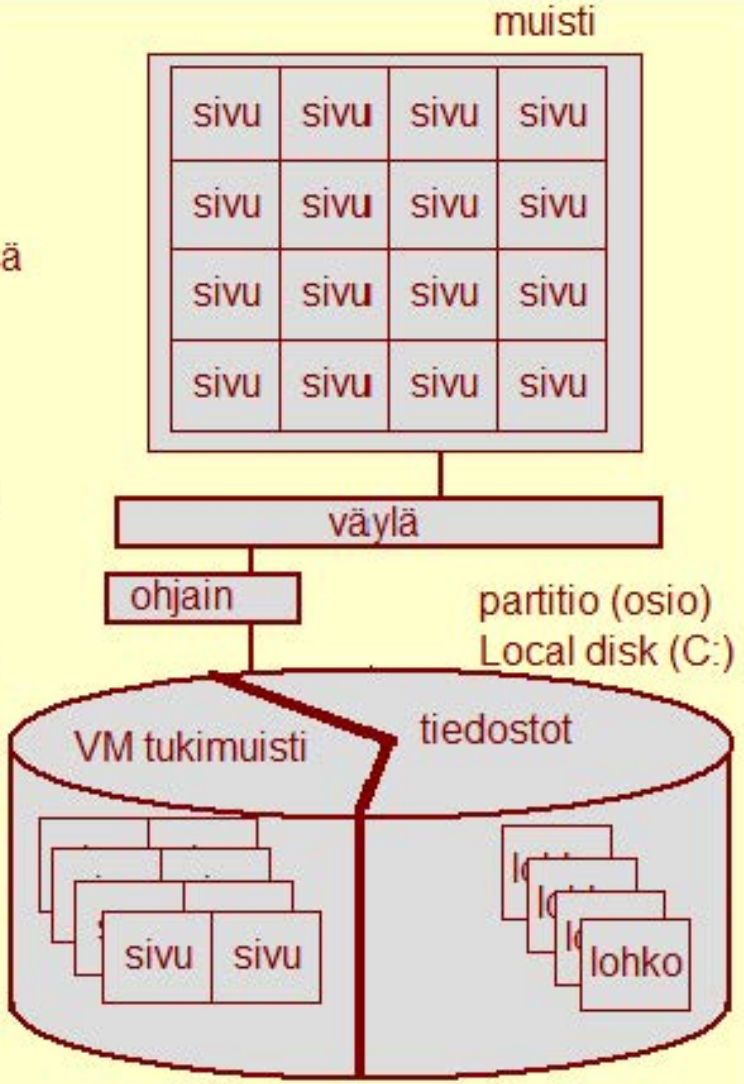
- samaa osiota voi käyttää myös tiedostoihin
- virtuaalimuistin tukimuisti

Virtuaalimuistin voi rakentaa tiedostojärjestelmän päälle

- virtuaalimuistin sivukehys on yksi tai useampia levylohkoja

Tiedostojärjestelmän voi rakentaa virtuaalimuistin päälle

- levylohko on yksi tai useampi virtuaalimuistin sivukehystä
- muistiinkuvattu tiedosto



Sekä virtuaalimuistijärjestelmä että tiedostojärjestelmä käyttävät samoja levyjä samalla tavalla. Virtuaalimuistin sivut ovat aivan vastaavassa asemassa tiedostojärjestelmän levylohkojen kanssa, koska niitä molempia luetaan ja kirjoitetaan fyysisille levyille kokonaisuuksina. Ei ole järkevää toteuttaa kahta liki samanlaista järjestelmää rinnakkain, joten yleensä toinen niistä toteutetaan ensin ja sitten toinen sen ensimmäisen avulla. Esimerkiksi, jos tiedostojärjestelmä on ensin toteutettu, niin virtuaalimuistin tukimuisti on helppo toteuttaa jo olemassaolevan tiedostojärjestelmän 'päälle'.

Tiedostot vs. virtuaalimuistin tukimuisti

Levy voi olla ositettu useaan osaan (partitioon)

- kukin osio käyttää omaa tiedostojärjestelmäänsä

Käyttöjärjestelmä on yhdessä partitiossa

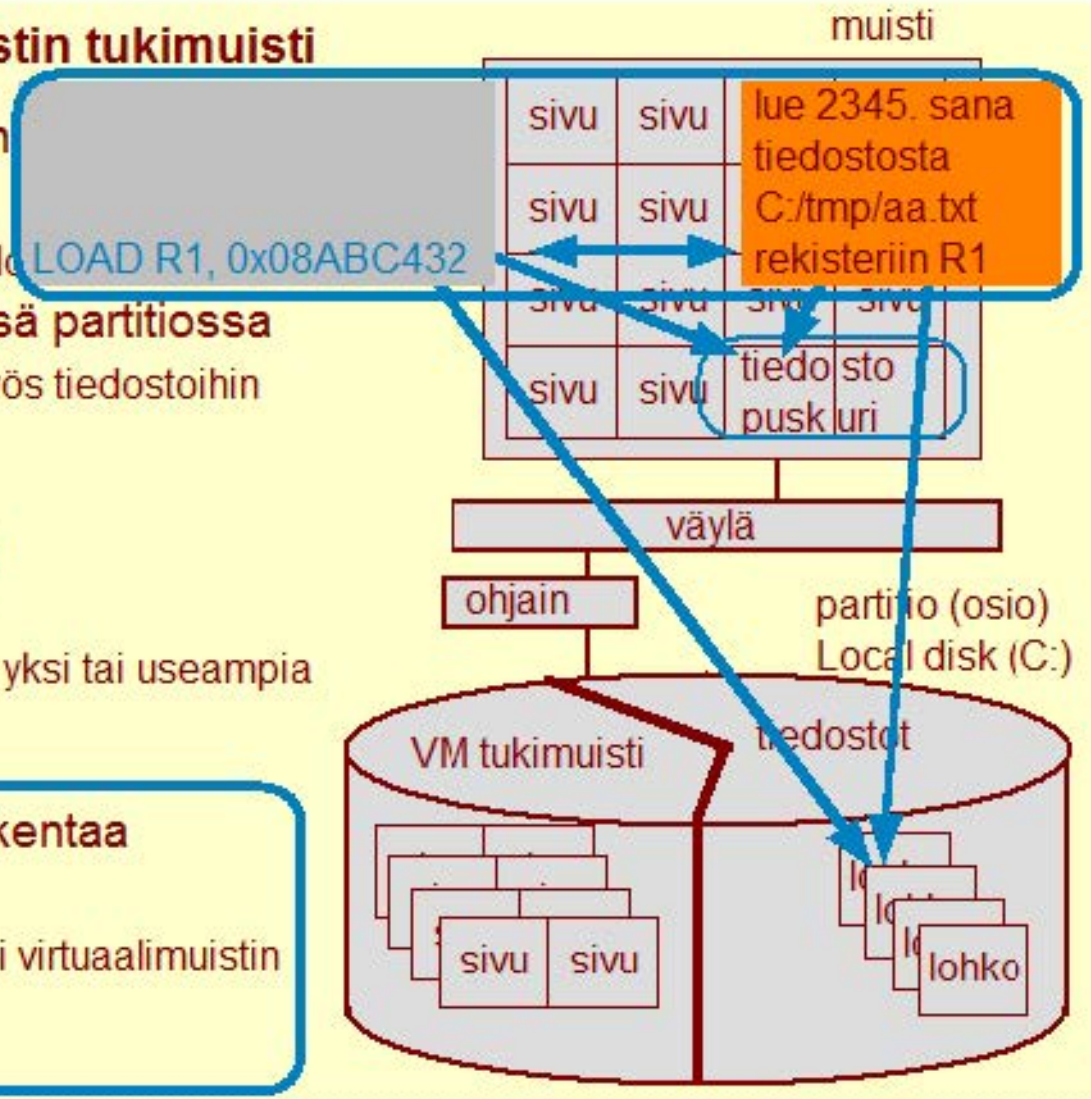
- samaa osiota voi käyttää myös tiedostoihin
- virtuaalimuistin tukimuisti

Virtuaalimuistin voi rakentaa tiedostojärjestelmän päälle

- virtuaalimuistin sivukehys on yksi tai useampia levylohkoja

Tiedostojärjestelmän voi rakentaa virtuaalimuistin päälle

- levylohko on yksi tai useampi virtuaalimuistin sivukehystä
- muistiinkuvattu tiedosto



Usein toteutus tehdään kuitenkin toisinpäin. Ensin tehdään virtuaalimuisti, ja sitten sen 'päälle' tiedostojärjestelmä. Tällaisessa systeemissä on mahdollista käsitellä tiedostoja kahdella eri tavalla: joko tavallisella tavalla tiedostojärjestelmän kautta tai sitten muistiinkuvattuna suoraan allaolevan virtuaalimuistijärjestelmän kautta. Tiedostoon pääsee nopeammin käsiksi esimerkiksi virtuaalimuistiosoitteen 0x08ABC432 avulla sen sijaan, että viitattaisiin esimerkiksi partition 'C:' hakemiston 'tmp' tiedoston 'aa.txt' sanaan 2345. Molemmissa tapauksissa viitattava tieto voi sijaita jo valmiiksi keskusmuistissa, mutta virtuaalimuistin kautta se löytyy paljon nopeammin kuin tiedostojärjestelmän taulukkoja läpikäyden.

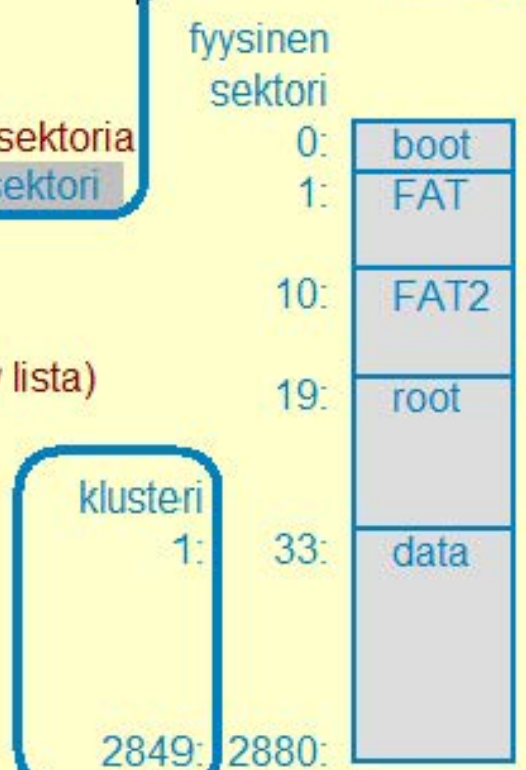
DOS/FAT-levyke

DOS-levyke

DOS klusteri = levylohko

- yksi levy, yksi pinta, ei hermeettisesti suljettu, 1.44 MB
- hidas: 300 rpm, 0.05 MB/s, keskim. hakuaika 90 ms
- talletus klustereissa, klusteri on 1 tai useampi peräkkäistä sektoria

tässä: klusteri = 1 sektori



FAT - File Allocation Table

- kertoo mitkä klusterit vapaana (alkion arvo nolla)
- kertoo mitkä klusterit käytössä millekin tiedostolle (linkitetty lista)
- kiinteä paikka levykkeellä, 2 kopiota

Tiedostot

- varsinaisen datan talletus

Hakemistot

- erikoistyyppinen tiedosto
- sisältää hakemistoalkion joka tiedostolle

Pöytäkoneissa käytettiin vielä 2000-luvun alussa pienten tiedostojen siirtoon koneesta toiseen muistitikkujen asemesta 1.44 MB levykkeitä, jotka yleensä oli formatoitu Microsoftin DOS/FAT tiedostojärjestelmän mukaisesti. Esimerkiksi Applen koneissa samoja levyjä formatoitiin myös Applen omalla tiedostojärjestelmällä, mutta DOS/FAT-tiedostojärjestelmä oli paljon yleisempi. Levyt formatoitiin levylohkoiksi eli klustereiksi, jotka muodostuivat yhdestä tai useammasta peräkkäisestä levysektorista. Levyt olivat hitaita ja naarmuille alttiita, koska niitä luettiin ja kirjoitettiin pölylle ja muulle lialle alttiissa luku/kirjoitusasemassa.

DOS/FAT-levyke

DOS-levyke

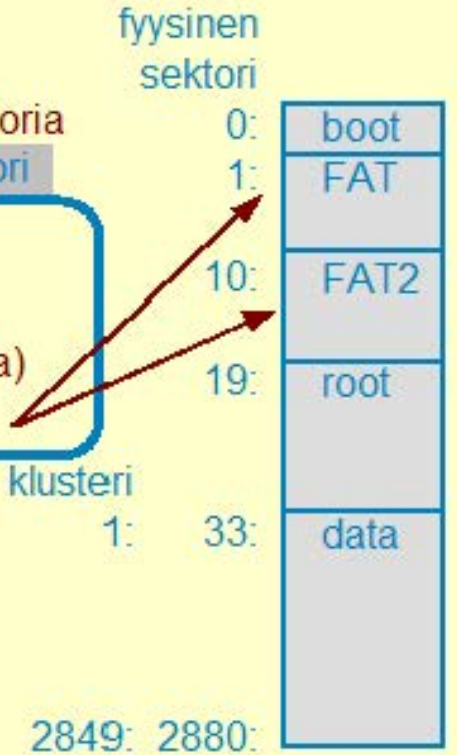
DOS klusteri = levylohko

- yksi levy, yksi pinta, ei hermeettisesti suljettu, 1.44 MB
- hidas: 300 rpm, 0.05 MB/s, keskim. hakuaika 90 ms
- talletus klustereissa, klusteri on 1 tai useampi peräkkäistä sektoria

tässä: klusteri = 1 sektori

FAT - File Allocation Table

- kertoo mitkä klusterit vapaana (alkion arvo nolla)
- kertoo mitkä klusterit käytössä millekin tiedostolle (linkitetty lista)
- kiinteä paikka levykkeellä, 2 kopiota



Tiedostot

- varsinaisen datan talletus

Hakemistot

- erikoistyyppinen tiedosto
- sisältää hakemistoalkion joka tiedostolle

FAT-tiedostojenhallintajärjestelmän ydin on FAT-taulu, jossa on yksinkertaisesti talletettuna tieto vapaista klustereista ja yhteen suuntaan linkitetyissä listoissa kunkin tiedoston varaamista klustereista. Ovelana oivalluksena samaa klusterin 12-bittistä indeksiä voidaan käyttää osoitteena sekä itse levykkeellä että myös FAT-taulukossa itsessään linkkiketjussa seuraavan alkion osoittamiseen. Tästä rakenteesta annetaan esimerkki ihan kohta. FAT-taulusta on levykkeellä myös varmuuskopio siltä varalta, että primäärinen FAT-taulu vahingoittuisi. DOS-levykkeet olivat kuuluisia helposta haavoittuvuudestaan.

DOS/FAT-levyke

DOS-levyke

DOS klusteri = levylohko

- yksi levy, yksi pinta, ei hermeettisesti suljettu, 1.44 MB
- hidas: 300 rpm, 0.05 MB/s, keskim. hakuaika 90 ms
- talletus klustereissa, klusteri on 1 tai useampi peräkkäistä sektoria

tässä: klusteri = 1 sektori

FAT - File Allocation Table

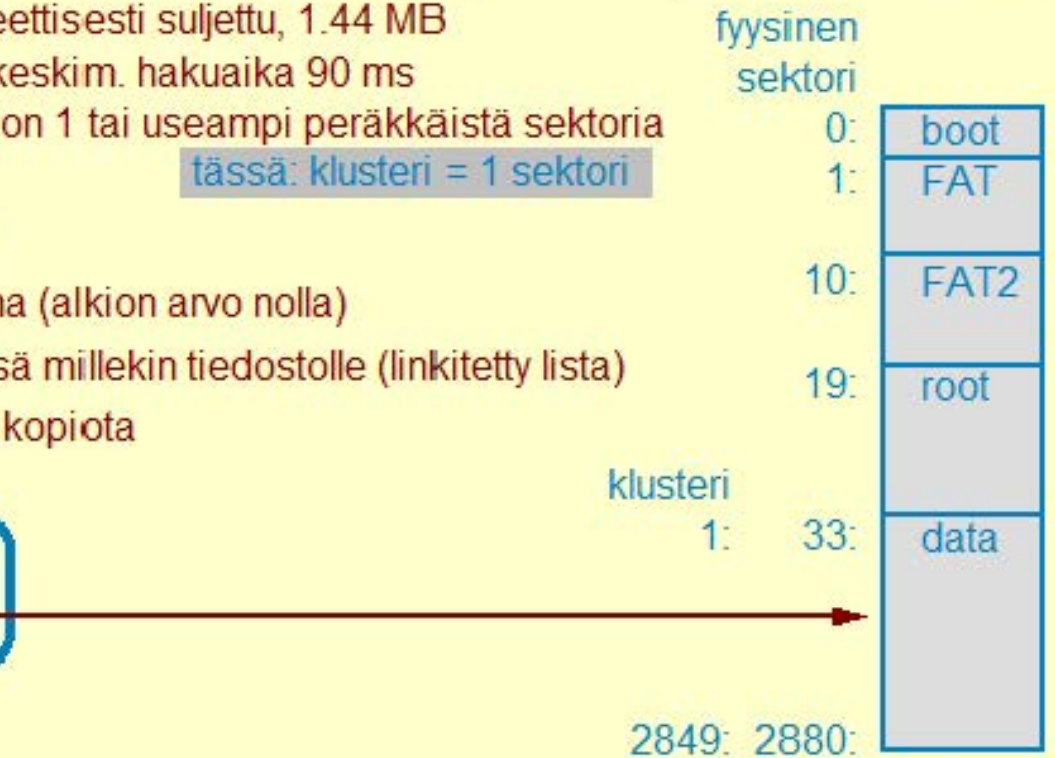
- kertoo mitkä klusterit vapaana (alkion arvo nolla)
- kertoo mitkä klusterit käytössä millekin tiedostolle (linkitetty lista)
- kiinteä paikka levykkeellä, 2 kopiota

Tiedostot

- varsinaisen datan talletus

Hakemistot

- erikoistyyppinen tiedosto
- sisältää hakemistoalkion joka tiedostolle



Pääosa levykkeestä on varattu tiedostoille ja hakemistoille. Tilan varaus kullekin tiedostolle tapahtuu kokonaisina klustereina ja varatut klusterit on merkitty FAT-tauluun. Hukkatila yhden tiedoston tai hakemiston osalta on siten keskimäärin puoli klusteria.

DOS/FAT-levyke

DOS-levyke

DOS klusteri = levylohko

- yksi levy, yksi pinta, ei hermeettisesti suljettu, 1.44 MB
- hidas: 300 rpm, 0.05 MB/s, keskim. hakuaika 90 ms
- talletus klustereissa, klusteri on 1 tai useampi peräkkäistä sektoria

tässä: klusteri = 1 sektori

FAT - File Allocation Table

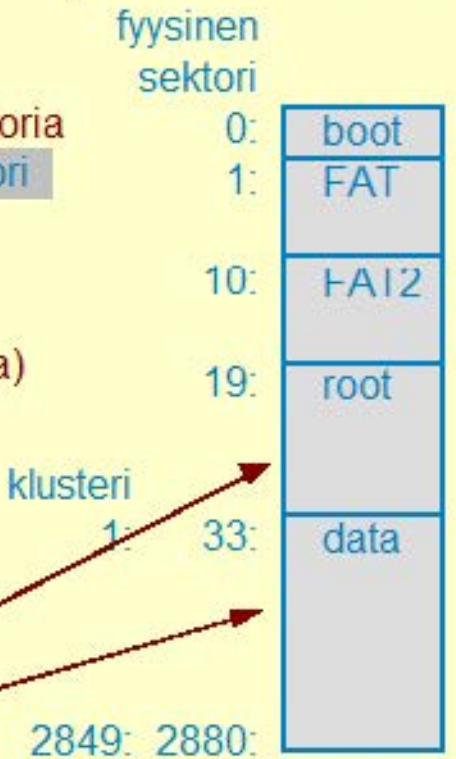
- kertoo mitkä klusterit vapaana (alkion arvo nolla)
- kertoo mitkä klusterit käytössä millekin tiedostolle (linkitetty lista)
- kiinteä paikka levykkeellä, 2 kopiota

Tiedostot

- varsinaisen datan talletus

Hakemistot

- erikoistyyppinen tiedosto
- sisältää hakemistoalkion joka tiedostolle



Tiedostot on organisoitu hakemistoihin, joista juuri- eli root-hakemiston paikka on aina kiinteässä kohdassa heti FAT-taulukoiden jälkeen. Hakemiston rakenne on kiinteä ja sieltä on helppo etsiä tiedostoa tai alihakemistoa nimen perusteella. Hakemistoalkiosta löytyy tiedoston tai alihakemiston nimen lisäksi yleistä hallintotietoa ja linkki FAT:iin, josta löytyy kyseisen tiedoston tai alihakemiston varaamat klusterit. Hakemistot ovat tietenkin vain erityisrakenteen omaavia tiedostoja, joita tiedostojärjestelmä tulkitsee hakemistoina.

Hakemistoalkio ja FAT

Tiedoston hakemistoalkio

- nimi, tyyppi, koko, muutos pvm ja klo
- attribuutit: invisible, read-only, ...
- ensimmäisen klusterin osoite (indeksi):
  - tiedoston ensimmäisen klusterin osoite levykkeellä
  - tiedoston seuraavan klusterin osoitteen FAT-alkion osoite

UNIX: ls -al

WINDOWS: dir mysubdir

FAT-alkion arvo

- 12-bittinen
  - rajaa klustereiden määrän 4096'een
- erikoisarvot
  - arvo 0x000 (nolla): käyttämätön klusteri, vapaa uusille tiedostoille
  - arvo 0xFF0-0xFF6 (4080-4086): varattu erikoiskäyttöön, esim. boot-lohko sektorissa 0
  - arvo 0xFF7 (4087): huono clusteri, ei käytössä enää
  - arvo 0xFF8-0xFFF (4088-4095): EOC (end of chain), ketjun loppu
- tavallinen arvo 0x001-0xFEF (1-4079)
  - seuraavan klusterin osoite ketjussa ja samalla sitä seuraavan klusterin osoitteen FAT-alkion osoite

FAT-tiedostojärjestelmässä tiedostot löytyvät hakemistoissa olevista hakemistoalkioista. Hakemistoalkiossa on kaikki hallintotieto kyseisestä tiedostosta ja ainoastaan varsinainen tieto puuttuu. Tiedoston tyyppinä on joko alihakemisto tai tavallinen tiedosto. Hakemiston hakemistoalkioden sisältöä voidaan lukea esimerkiksi ls- tai dir-komennoilla, jotka sitten listaavat käyttäjälle halutut tiedot kustakin halutusta hakemistoalkiosta. Kaikki dir-komennon tulostamat tiedot löytyvät hakemistoalkiosta, eikä itse tiedostoja tarvitse lukea tässä yhteydessä.

Hakemistoalkio ja FAT

Tiedoston hakemistoalkio

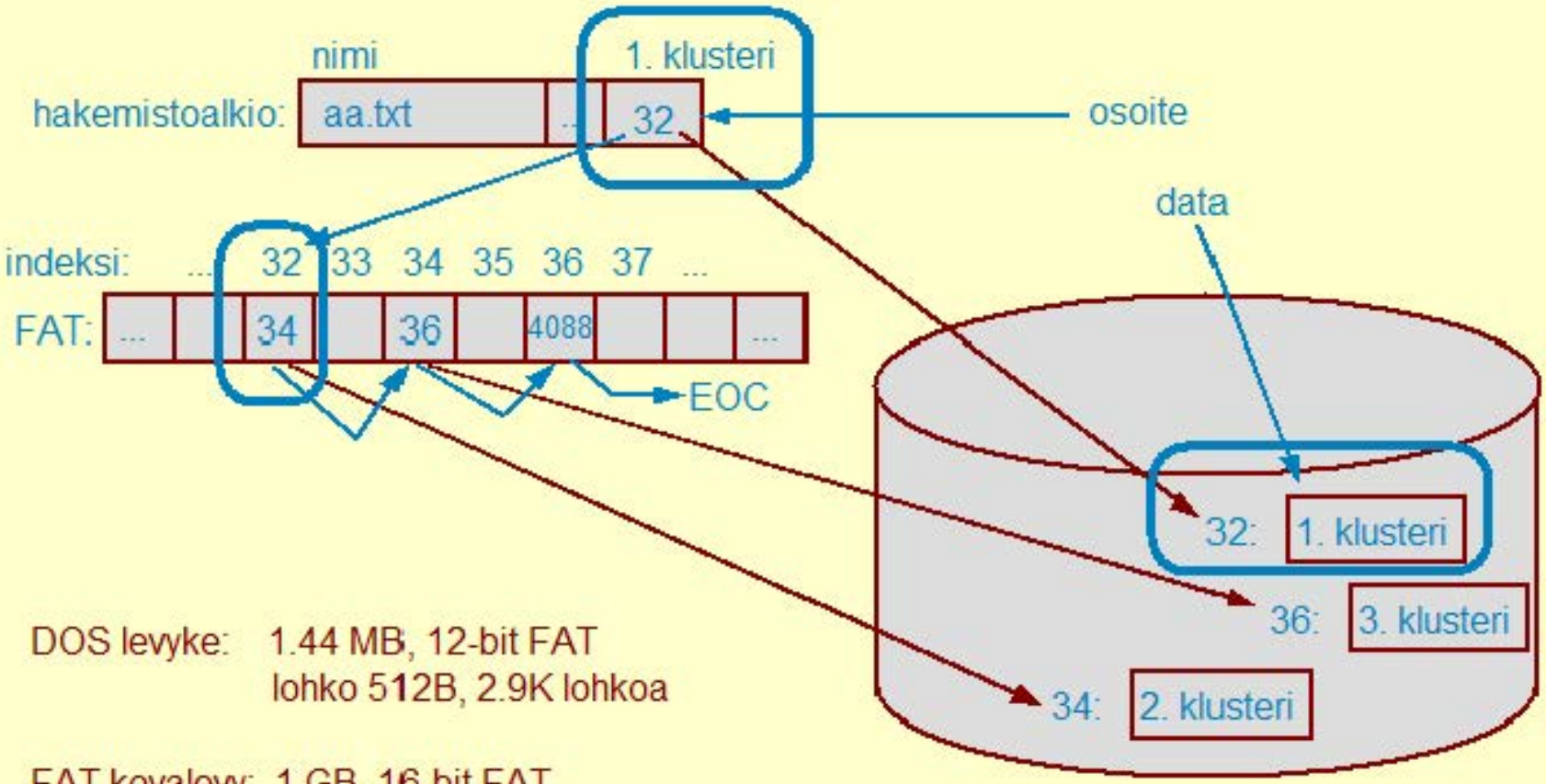
- nimi, tyyppi, koko, muutos pvm ja klo
- attribuutit: invisible, read-only, ...
- ensimmäisen klusterin osoite (indeksi):
  - tiedoston ensimmäisen klusterin osoite levykkeellä
  - tiedoston seuraavan klusterin osoitteen FAT-alkion osoite

FAT-alkion arvo

- 12-bittinen
  - rajaa klustereiden määrän 4096'een
- erikoisarvot
  - arvo 0x000 (nolla): käyttämätön klusteri, vapaa uusille tiedostoille
  - arvo 0xFF0-0xFF6 (4080-4086): varattu erikoiskäyttöön, esim. boot-lohko sektorissa 0
  - arvo 0xFF7 (4087): huono clusteri, ei käytössä enää
  - arvo 0xFF8-0xFFF (4088-4095): EOC (end of chain), ketjun loppu
- tavallinen arvo 0x001-0xFEF (1-4079)
  - seuraavan klusterin osoite ketjussa ja samalla sitä seuraavan klusterin osoitteen FAT-alkion osoite

Tavallisen levykkeen käyttöön sopivan FAT tiedostojärjestelmän FAT-alkiot ovat 12-bittisiä, jolloin suurin mahdollinen viitattavien klustereiden määrä on tietenkin 4096. FAT-alkion arvo on yleensä pelkkä indeksi, joka ilmaisee seuraavan klusterin sijainnin levyllä. Samaa indeksi ilmaisee sitten myös sitä seuraavan klusterin osoitteen sijainnin FAT-taulussa. Seuraavan sivun esimerkki valaisee tilannetta. FAT-alkion erikoisarvoilla ilmaistaan käyttämättömät ja vialliset k lusterit. Erikoisarvoilla voidaan myös ilmaista, jos tällä levykkeellä olevia tietoja halutaan käyttää koko käyttöjärjestelmän alustamiseen.

FAT-esimerkki

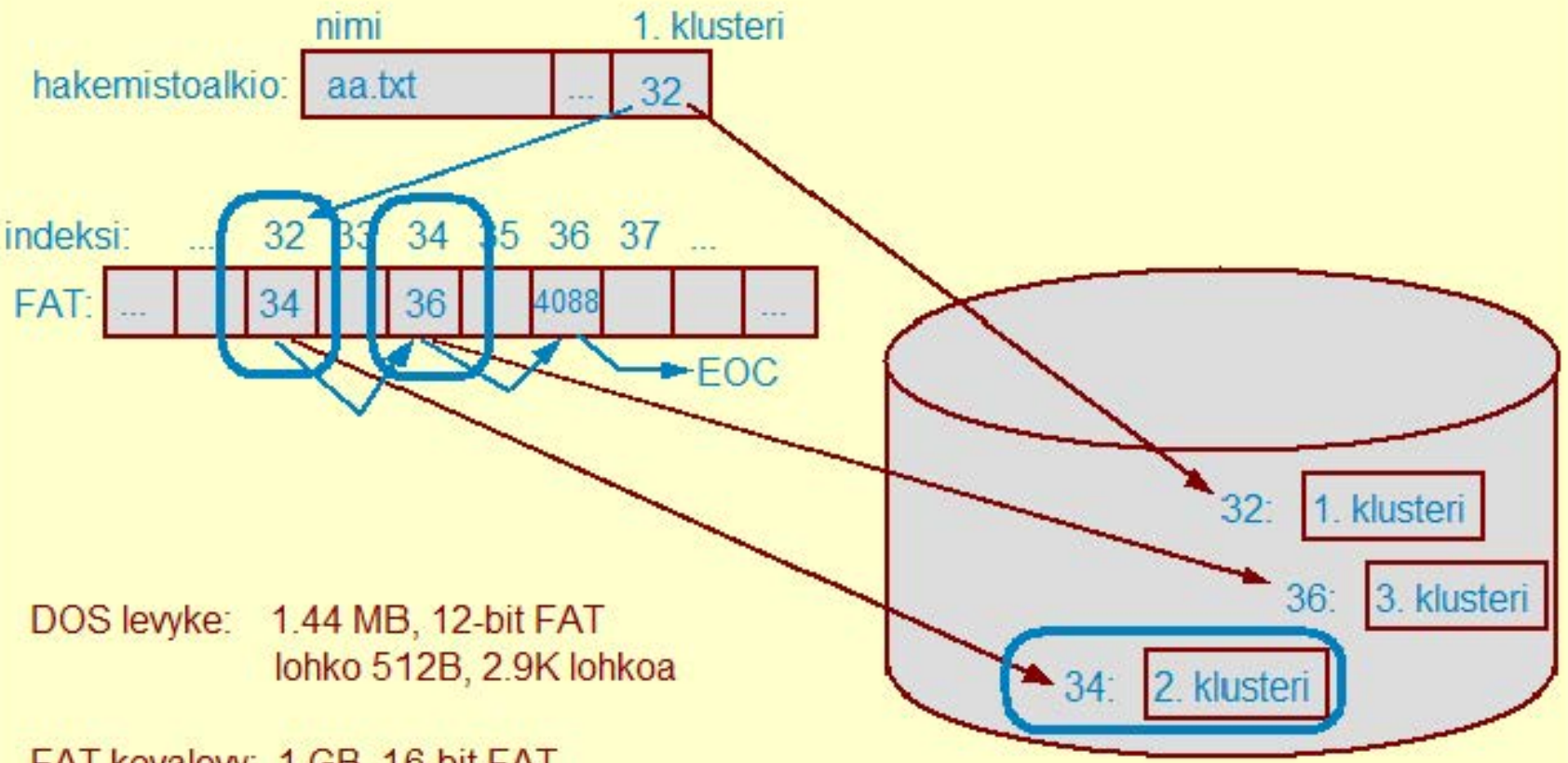


DOS levyke: 1.44 MB, 12-bit FAT
lohko 512B, 2.9K lohkoa

FAT kovalevy: 1 GB, 16-bit FAT
lohko 64 KB, 64K lohkoa ???

Tässä esimerkissä käsittelemme tiedostoa aa.txt, joka on talletettu levylle 3 klusterissa.
Tiedostojenhallintajärjestelmä on käynyt läpi hakemistoja hakemistopuussa, kunnes lopulta ollaan päädytty alihakemistoon, jossa kyseisen tiedoston hakemistoalkio on. Muiden tietojen lisäksi hakemistoalkiossa on tiedoston aa.txt ensimmäisen klusterin osoite 32. Klusterissa 32 (eli sektorissa 64, jos kussakin klusterissa on 1 sektori) on siis tiedoston aa.txt 512 ensimmäistä tavua. FAT-alkiossa 32 on seuraavan klusterin osoite eli 34.

FAT-esimerkki

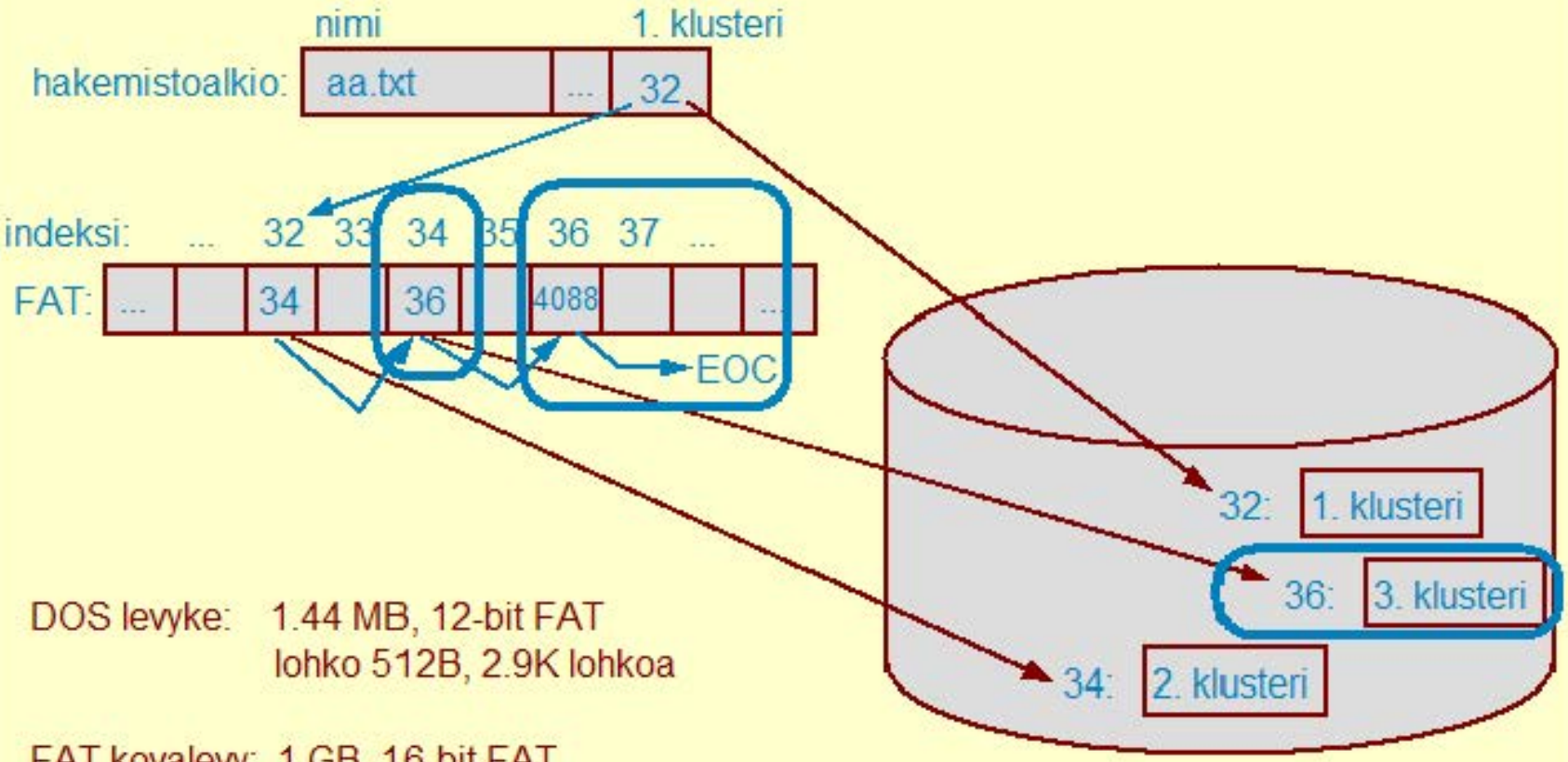


DOS levyke: 1.44 MB, 12-bit FAT
lohko 512B, 2.9K lohkoa

FAT kovalevy: 1 GB, 16-bit FAT
lohko 64 KB, 64K lohkoa ???

Tiedoston aa.txt seuraavat 512 tavua löytyvät sitten levykkeen klusterista 34 ja vastaavasti sitä seuraavan klusterin osoite löytyy FAT-alkiosta 34.

FAT-esimerkki

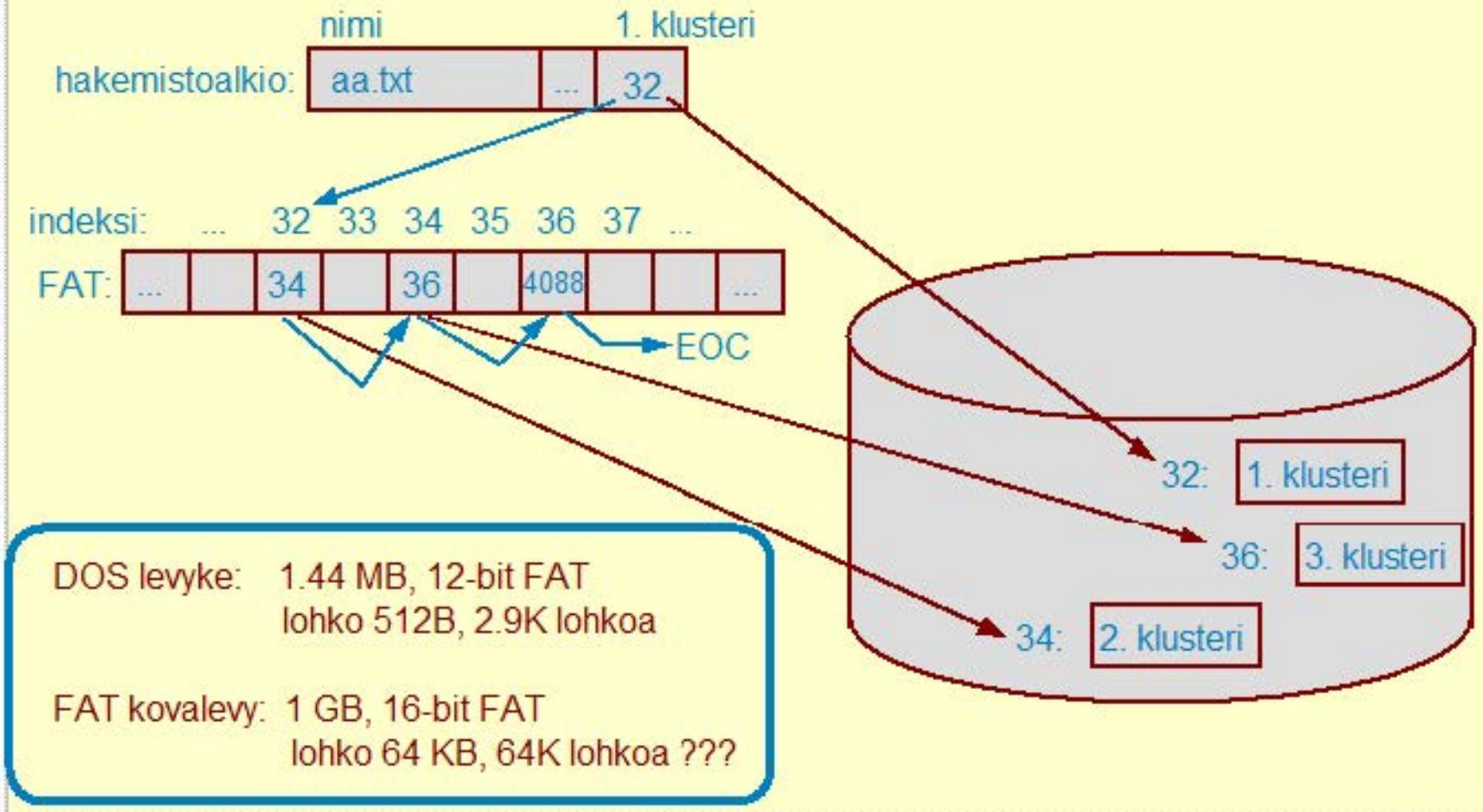


DOS levyke: 1.44 MB, 12-bit FAT
lohko 512B, 2.9K lohkoa

FAT kovalevy: 1 GB, 16-bit FAT
lohko 64 KB, 64K lohkoa ???

Tiedoston aa.txt loput tavut löytyvät klusterista 36. Emme tiedä, kuinka monta näistä 512 tavusta klusterissa 36 on oikeasti käytössä, mutta tiedostojenhallinta voi päätellä sen hakemistoalkiossa olevasta tiedoston koosta. FAT-alkiossa 36 on arvona 4088 eli end of clusters, joten tämä oli viimeinen tiedostolle aa.txt varattu klusteri. Ketju päättyy tähän.

FAT-esimerkki

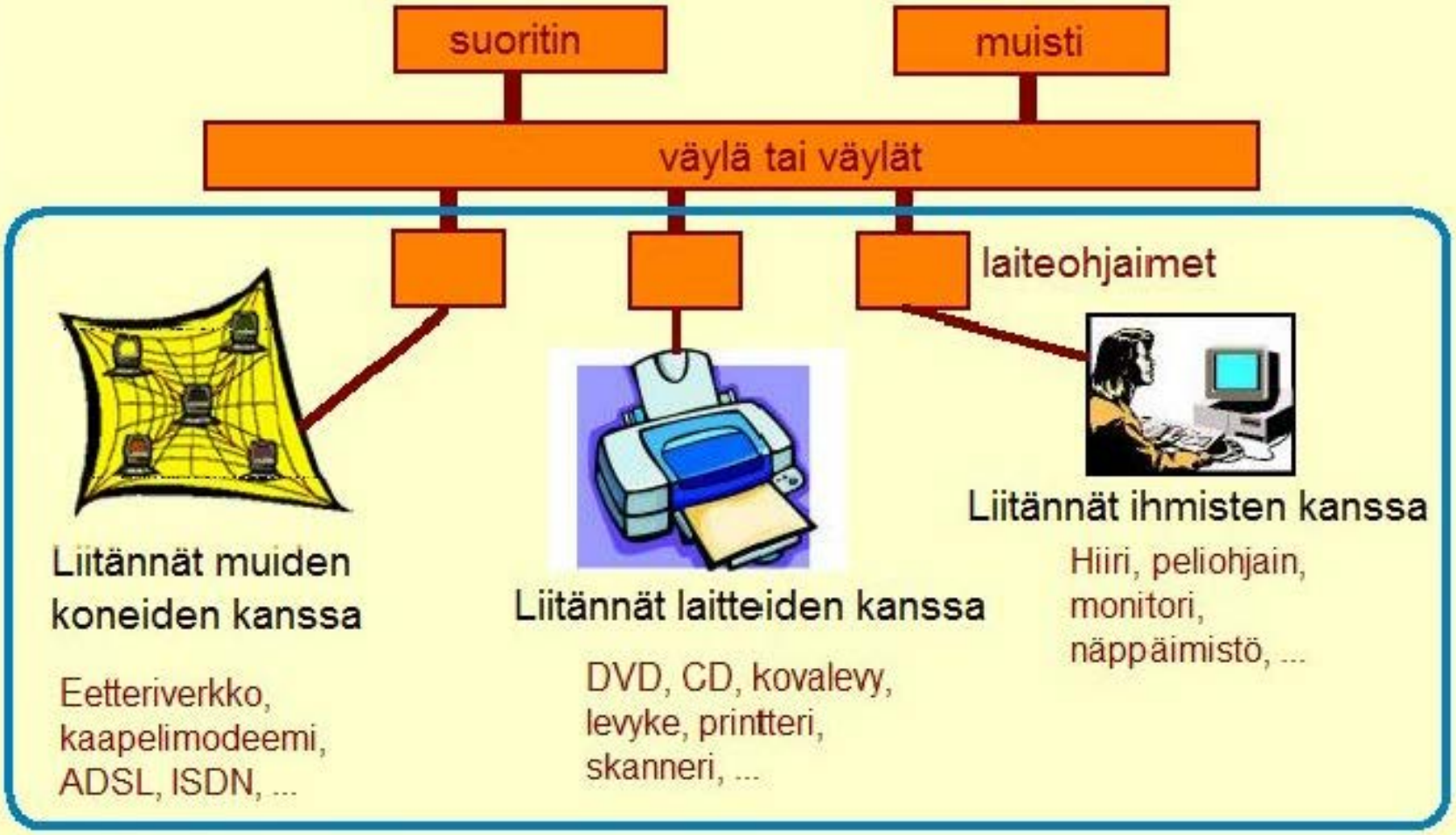


DOS levyke: 1.44 MB, 12-bit FAT
lohko 512B, 2.9K lohkoa

FAT kovalevy: 1 GB, 16-bit FAT
lohko 64 KB, 64K lohkoa ???

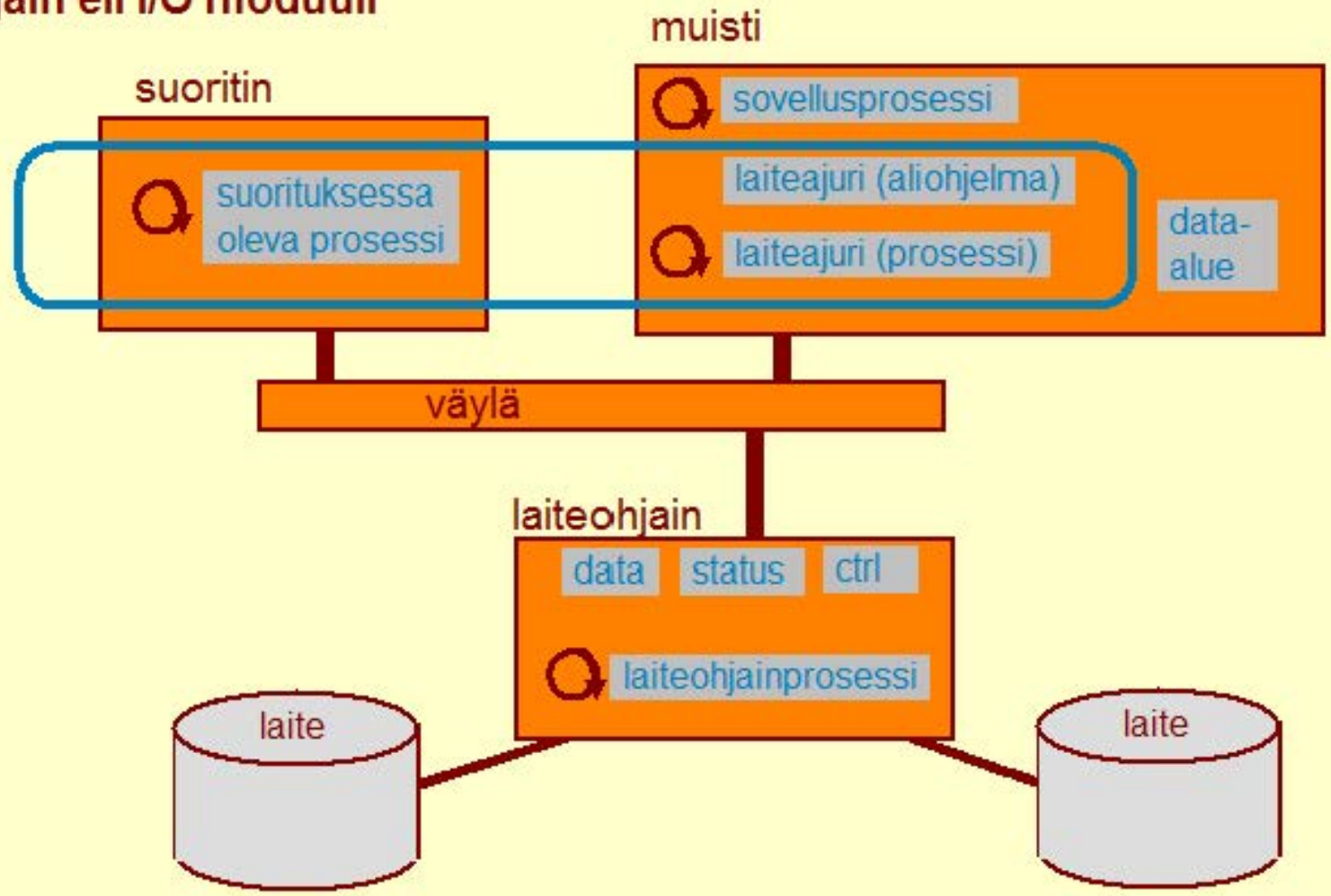
FAT-tiedostojärjestelmä sopii hyvin pienille levykkeille ja sitä on käytetty laajennettuna myös isommille levyille. Isommille levyille FAT-alkion pituutta tulee kuitenkin pidentää, jotta FAT-tauluista ei tulisi liian suuria. Tässä tulee kuitenkin pian raja vastaan ja esimerkiksi jo 1 GB levyille FAT ei välttämättä ole paras vaihtoehto. Uudemmat Windows-järjestelmät käyttävätkin suurille levyille useimmiten uudempaa NTFS eli New Technology File System tiedostojärjestelmää. NTFS'ään voitte tutustua tarkemmin kirjallisuudessa ja käyttöjärjestelmäkursseilla.

I/O laitteiden liittäminen järjestelmään



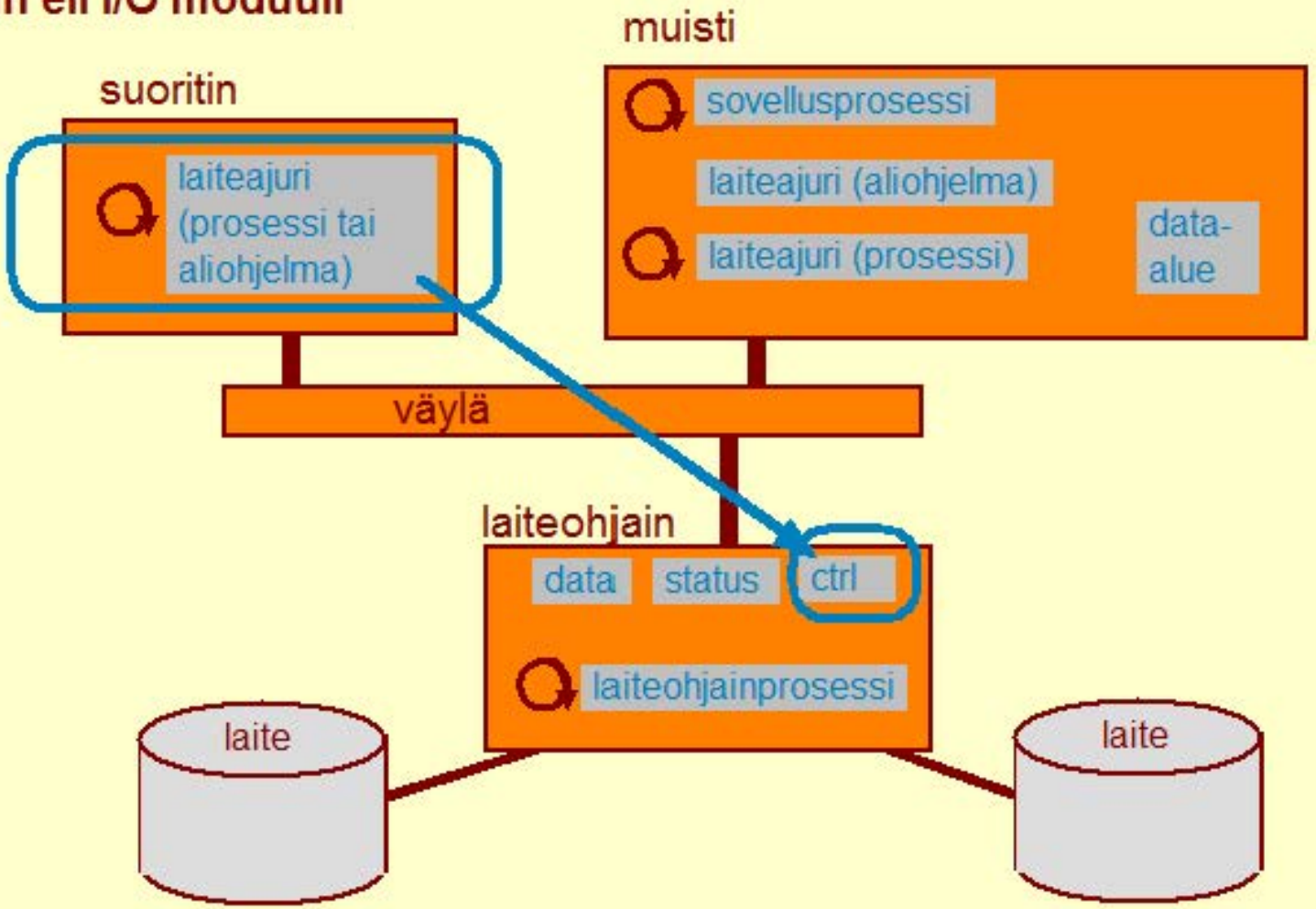
Vaikka järjestelmään voidaan liittää hyvin erilaisia ja eri nopeuksisia laitteita, niiden kaikkien liitos on silti samanlainen. Kullekin laitetyypille on omanlaisensa laiteohjain, joka voidaan liittää järjestelmän väylähierkiaan. Nopeampien laitteiden väyläohjaimet laitetaan lähemmäs muistiväylää ja hitaampien laitteiden kauemmas muistiväylästä hitaampiin aliväyliin, jotta ne eivät hidastaisi nopeampien laitteiden toimintaa. Fyysisen yhteensopivuuden lisäksi kuhunkin laiteohjaimeen liittyy käyttöjärjestelmän liitospalikka juuri tälle laitteelle eli kyseisen laitteen laiteajuri. Laiteajuri on normaali käyttöjärjestelmän palvelurutiini.

Laiteohjain eli I/O moduuli



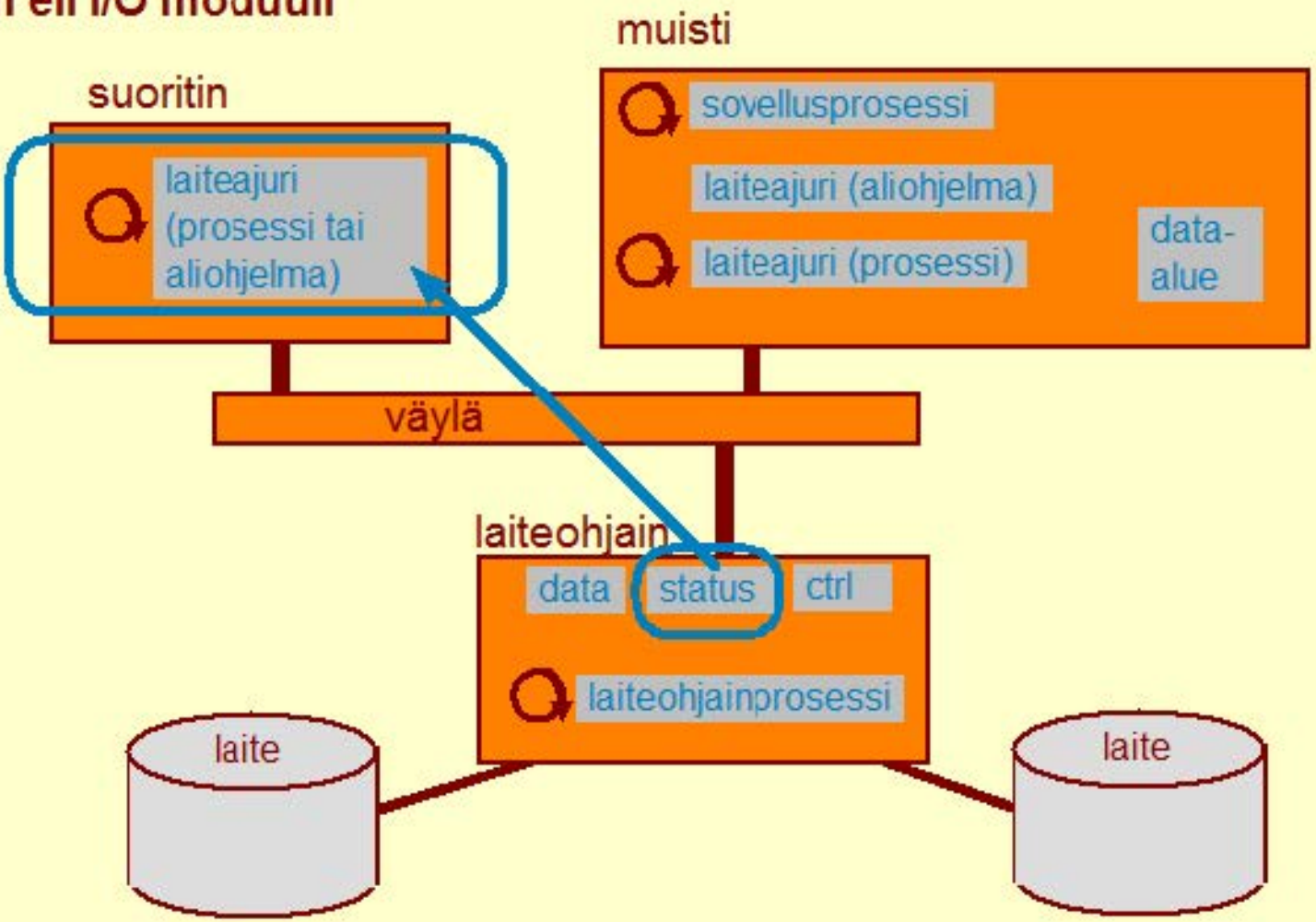
Kun suorituksessa oleva prosessi haluaa tehdä I/O:ta, se kutsuu laiteajuria. Laiteajuri voi olla toteutettu omana prosessinaan tai sitten aliohjelmana. Laiteajuri pääsee suoritukseen joko omana prosessinaan tai sitten aliohjelmana I/O:ta haluavalle prosessille. Joka tapauksessa laiteajuri suorittaa samalla (tai samantasoisella) suorittimella kuin mitä I/O:ta haluava prosessikin suoritti. Laiteajuri voi tietenkin suorittaa ainoastaan silloin, kun sillä on suoritusvuoro.

Laiteohjain eli I/O moduuli



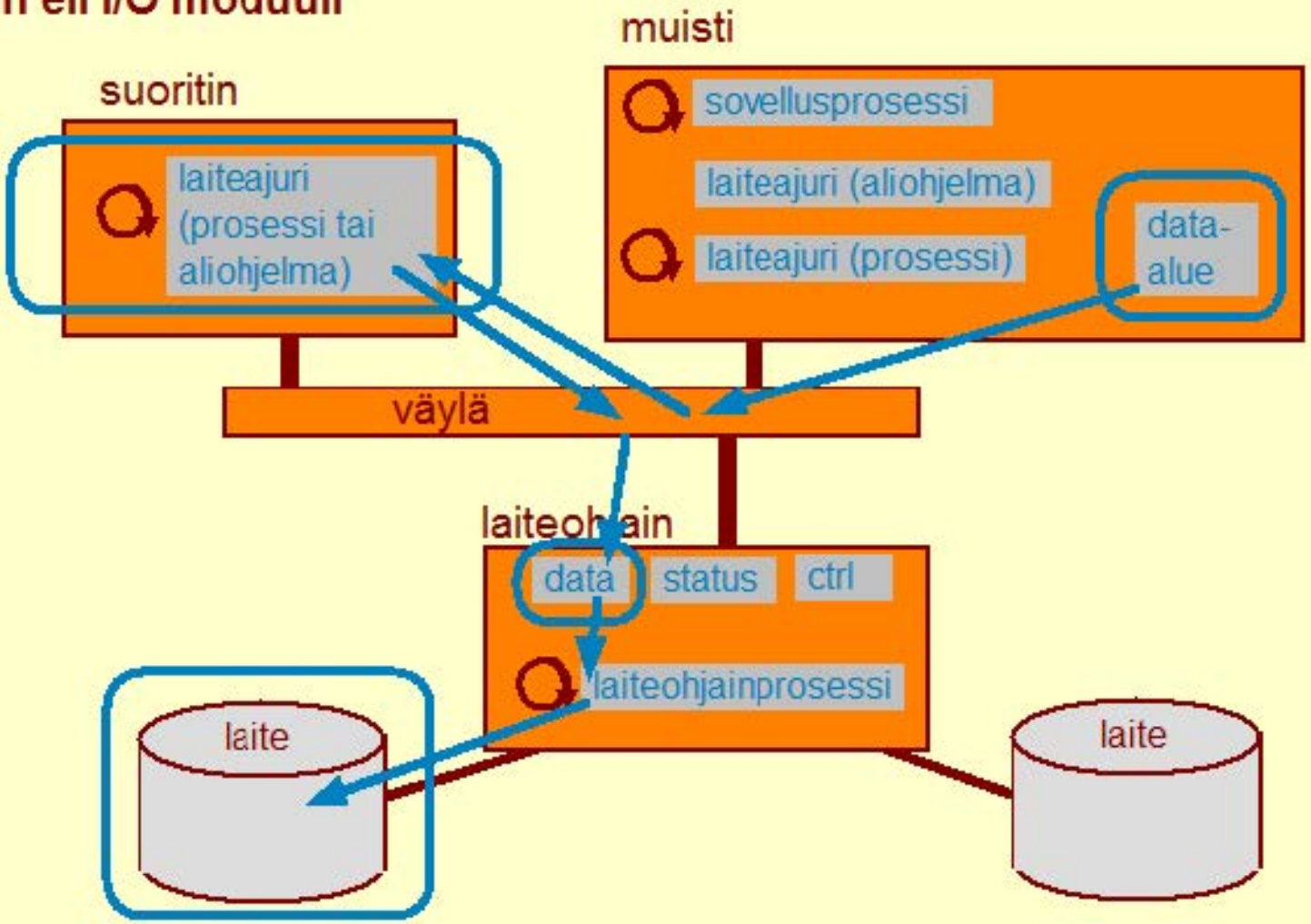
Laiteajuri ohjaa oheislaitteen toimintaa laiteohjaimella olevan kontrollirekisterin avulla. Nimitys 'rekisteri' on tässä vähän ehkä harhaanjohtava, koska kysymyksessä on vain laiteohjaimella sijaitseva muistialue, johon voidaan viitata (ehkä etuoikeutetuilla) konekäskyillä. Laiteajuri antaa siis kontrollirekisteriin komentoja, joita laiteohjain sitten toteuttaa. Kontrollirekisterin muodon ei tarvitse olla vakio, mutta se voi olla vaikkapa vain muutama kokonaisluku.

Laiteohjain eli I/O moduuli



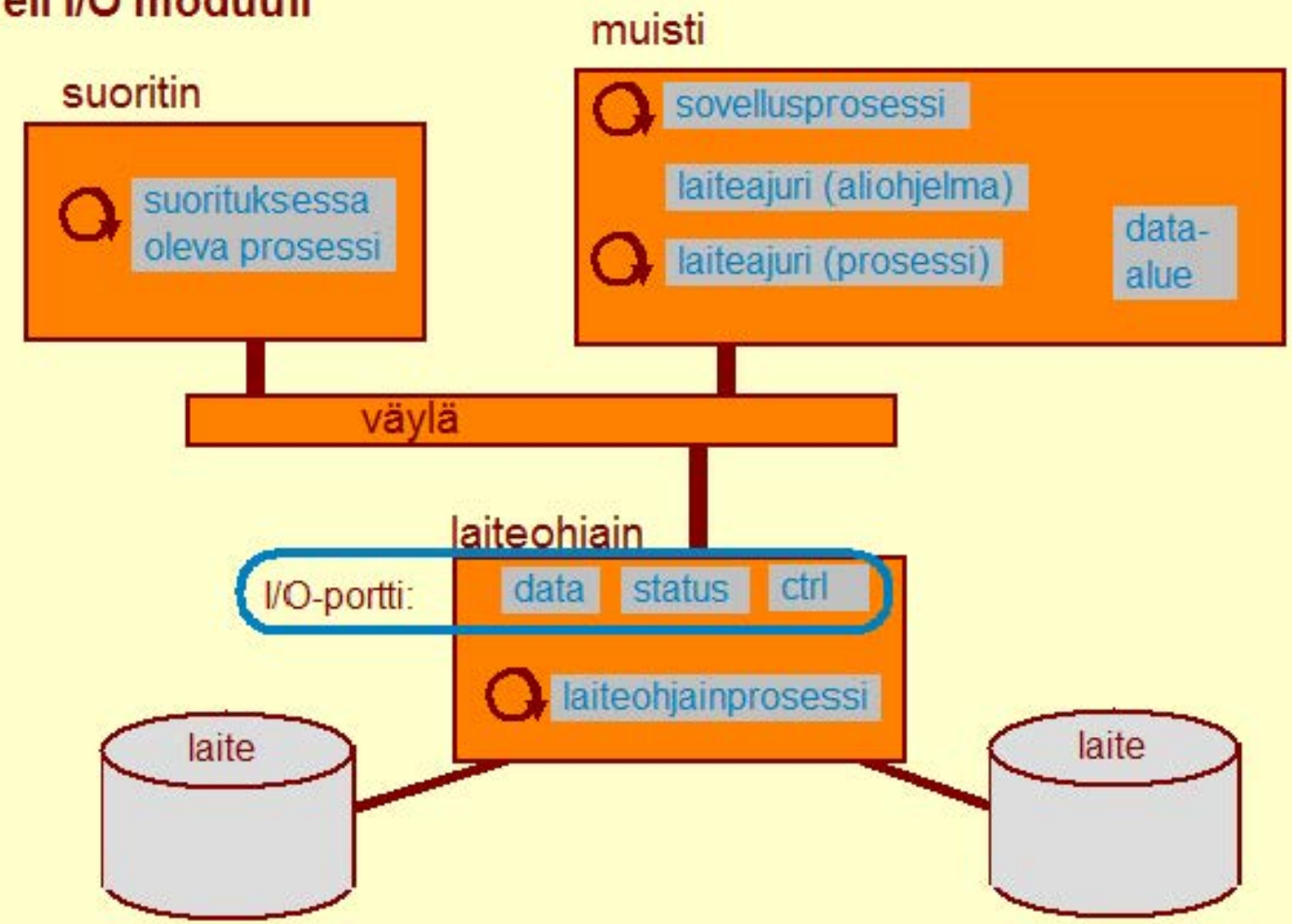
Laiteajuri voi lukea laiteohjaimen tilaa erityisestä statusrekisteristä, joka myöskin voi olla vain yksi tai kaksi kokonaislukua. Tällä tavoin laiteohjain voi kertoa laiteajurille omasta tilastaan. Tyypillinen viesti voisi olla vaikkapa 'laite vapaa' tai 'tehtävä suoritettu', mitkä kumpikin voidaan helposti koodata pienillä kokonaisluvuilla. Laiteajuri ei tietenkään voi lukea statusrekisterin arvoa ellei se ole sillä hetkellä suorituksessa. Odotustilassa olevan laiteajurin voi herättää esimerkiksi I/O-laitekeskeytyksellä, jos vain laiteohjain on tarpeeksi älykäs sen tekemään.

Laiteohjain eli I/O moduuli



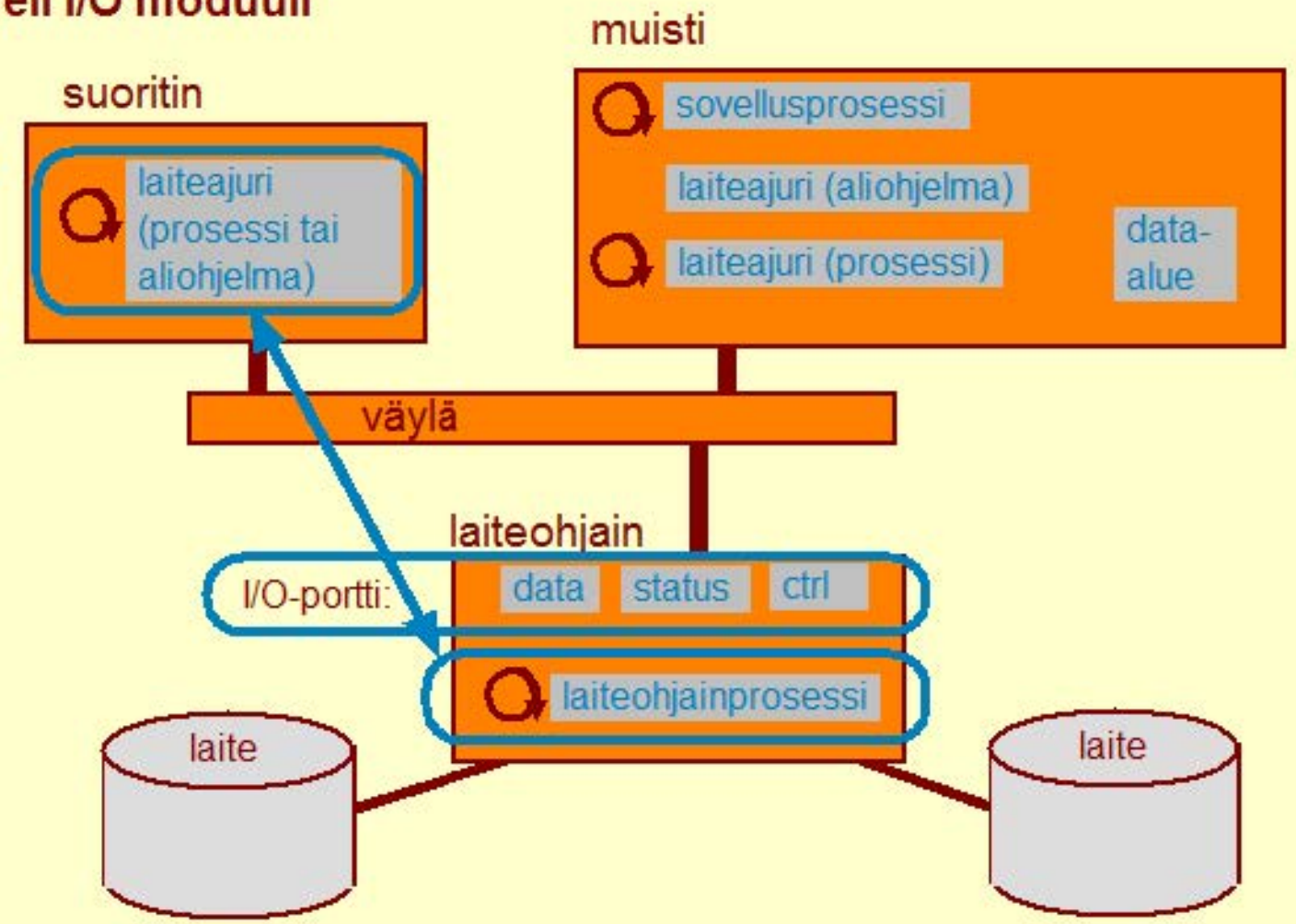
Varsinainen tiedonsiirto keskumuistista laitteelle (tai päin vastoin) tapahtuu laiteohjaimen datarekisterin kautta. Yleensä ajurin pitää aina ensin lukea data muistista suorittimen rekisteriin, ja sitten vasta kopioda se laiteohjaimen datarekisteriin, josta laiteohjaimella suorittava laiteohjainprosessi sitten sen aikanaan siirtää varsinaiselle laitteelle. Datarekister on usein usean sanan tai jopa usean kilotavun mittainen. Esimerkiksi kovalevyjen laiteohjaimen datarekisteri sisältää usein montakin erillistä kilotavujen mittaista datapuskuria, yhden kullekin aukiolevalle tiedostolle.

Laiteohjain eli I/O moduuli



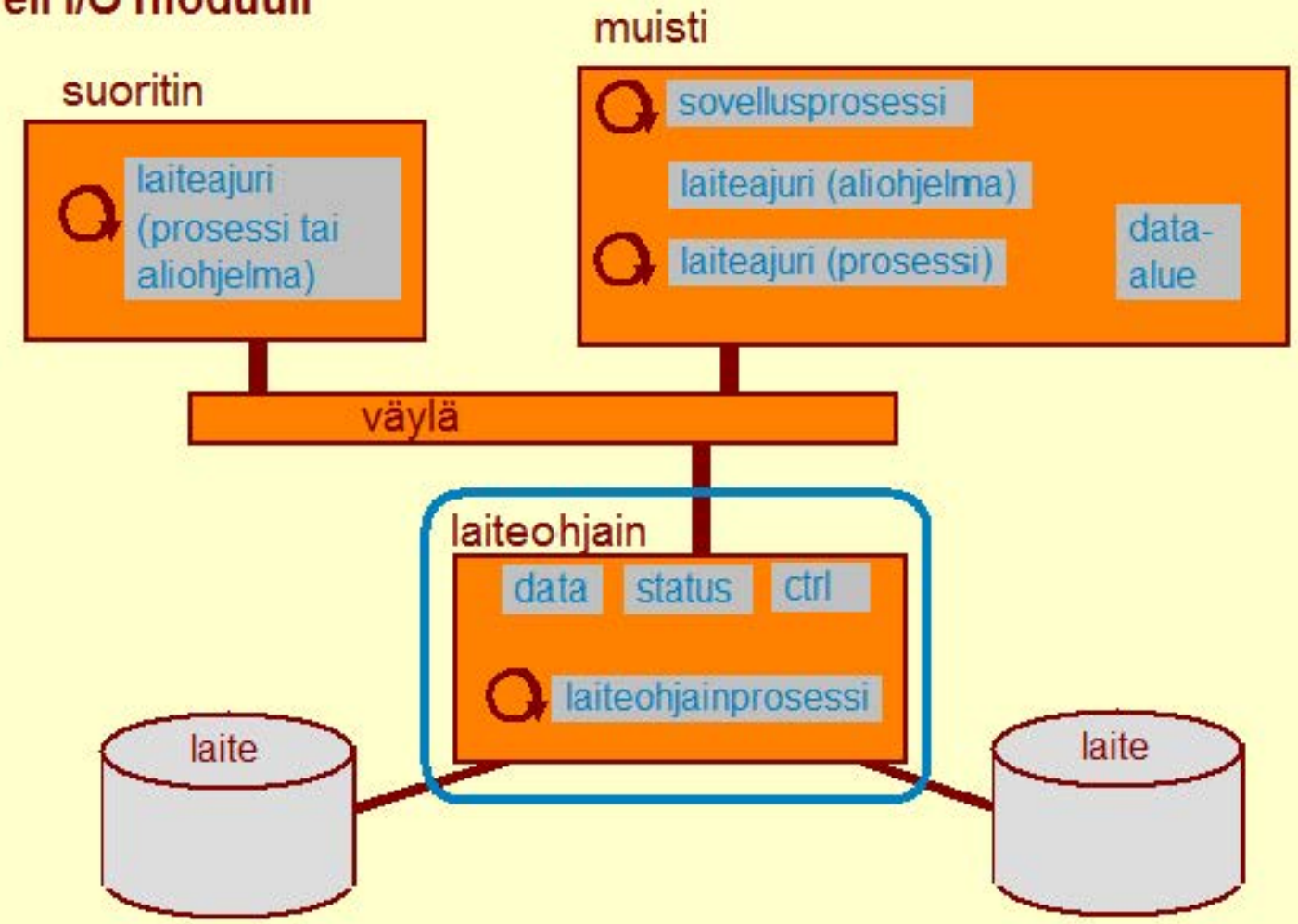
Kontrolli-, status- ja datarekisterit muodostavat ns. I/O-portin, jonka avulla minkä tahansa laitteen I/O-rajapinta määräytyy. Kunhan vain laiteajurilla on tiedossaan ja käytössään laitteen I/O-portti, niin ajurin lopputoteutus on aika suoraviivaista.

Laiteohjain eli I/O moduuli



Laiteohjain on itse asiassa aika lailla samanlainen suoritin kuin keskusyksikkökin. Erona kuitenkin on, että laiteohjaimella varsinaisen työn tekee aina vain yksi prosessi, laiteohjainprosessi, joka on aina suoritusvuorossa. Laiteohjainprosessi sitten toteuttaa I/O:n laiteajurin kanssa yhteistoiminnassa ja niiden välinen kommunikointi tapahtuu laiteohjaimen I/O-portin kautta. Kommunikcinnin tekee vaikeaksi se, että ainoastaan laiteohjainprosessi on aina suorituksessa, kun taas laiteajuri voi olla 'tajuttomana' odotustilassa.

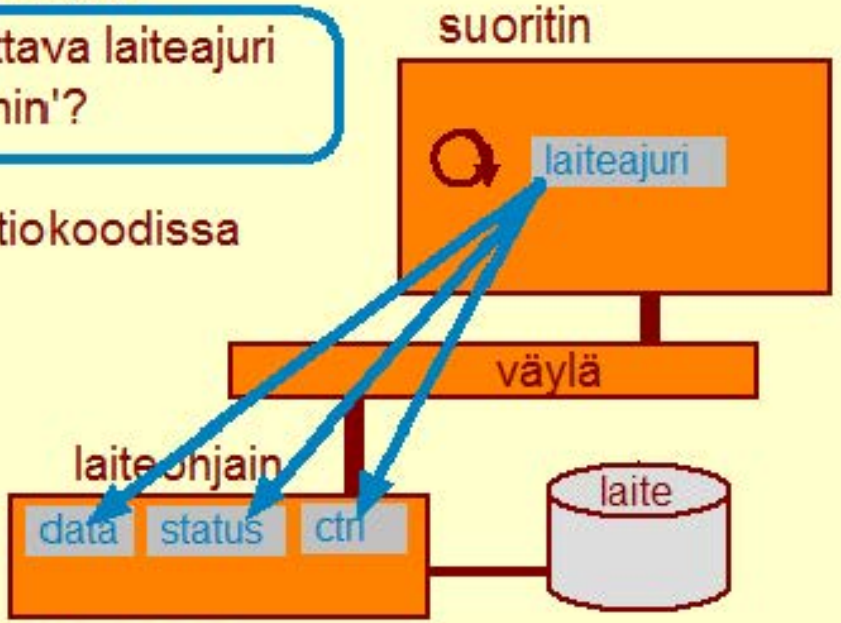
Laiteohjain eli I/O moduuli



Laiteohjain ja sillä suorittava laiteohjainprosessi voivat olla hyvinkin yksinkertaisia. Esimerkiksi näppäimistön laiteohjain ja laiteohjainprosessi ovat tällaisia. Toisaalta, laiteohjain voi sisältää jopa tehokkaamman suorittimen kuin varsinaisen laitteiston pääsuoritin on. Esimerkiksi näytönohjaimet sisältävät usein hyvin nopean grafiikkaan ja liikkuvan kuvan käsittelyyn optimoidun suorittimen ja hyvin nopeata muistia datarekisterin puskurien toteutukseen. Sen lisäksi siinä on myös paljon tavallista muistia puskureissa olevan kuvatiedon käsittelyyn. Laiteohjainprosessi on myöskin tällöin aika monimutkainen otus.

Laiteohjaimen rekistereihin viittaaminen

Ongelma: miten pääsuorittimella suorittava laiteajuri viittaa laiteohjaimella oleviin 'rekistereihin'?



Omat I/O-konekäskyt, toteutus operaatiokoodissa

- IN, OUT
- käskyssä laiteohjaimen (ja laiterekisterin) tunniste
- vaikea laajentaa uusiin laitteisiin
- toteutus käskykannan suorittimen yhteydessä

```
in r1, =KBD
```

Muistiinkuvattu I/O, toteutus osoitteen tulkinnassa

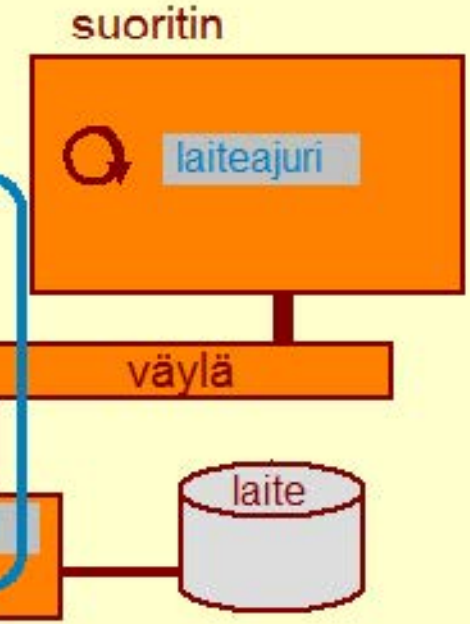
- ei tarvita erillisiä I/O konekäskyjä
- laiteohjaimen "rekisterit" viitattavissa kuten tavallinen muisti
- muistiosoitteen ensimmäiset bitit määräävät, mille laitteelle (tav. muisti vai laiteohjaimella oleva muisti) viite kohdistuu
- vie tilaa normaalilta muistiavaruudelta
- voidaan käyttää rinnan I/O-konekäskyjen kanssa

```
load  r1, =KbdRead
store r1, KbdCtrl
```


Laiteajuri siis kommunikoi laiteohjaimella suorittavan laiteohjainprosessin kanssa käyttäen laiteohjaimella toteutettua muistialuetta (eli 'rekistereitä'), joihin pitää siis pystyä jollain tavoin konekäskyissä viittaamaan. Kysymys on, että miten? Konekäskyissä pitää olla erityispiirteitä tätä tarkoitusta varten ja joka tapauksessa haluamme varmaan, että kyseiset viitteet voi tehdä vain etuoikeutetussa tilassa. Jos tavalliset sovellusprosessit voisivat suoraan manipuloida mitä tahansa laitetta, niin se olisi aivan liian riskialtista. Tosin se olisi ehkä hiukkasen nopeampaa, kun suorittimen tilaa ei tarvitsisi vaihtaa etuoikeutettuun ja takaisin.

Laiteohjaimen rekistereihin viittaaminen

Ongelma: miten pääsuorittimella suorittava laiteajuri viittaa laiteohjaimella oleviin 'rekistereihin'?



Omat I/O-konekäskyt, toteutus operaatiokoodissa

- IN, OUT — `in r1, =KBD`
- käskyssä laiteohjaimen (ja laiterekisterin) tunniste
- vaikea laajentaa uusiin laitteisiin
- toteutus käskykannan suorittimen yhteydessä

Muistiinkuvattu I/O, toteutus osoitteen tulkinnassa

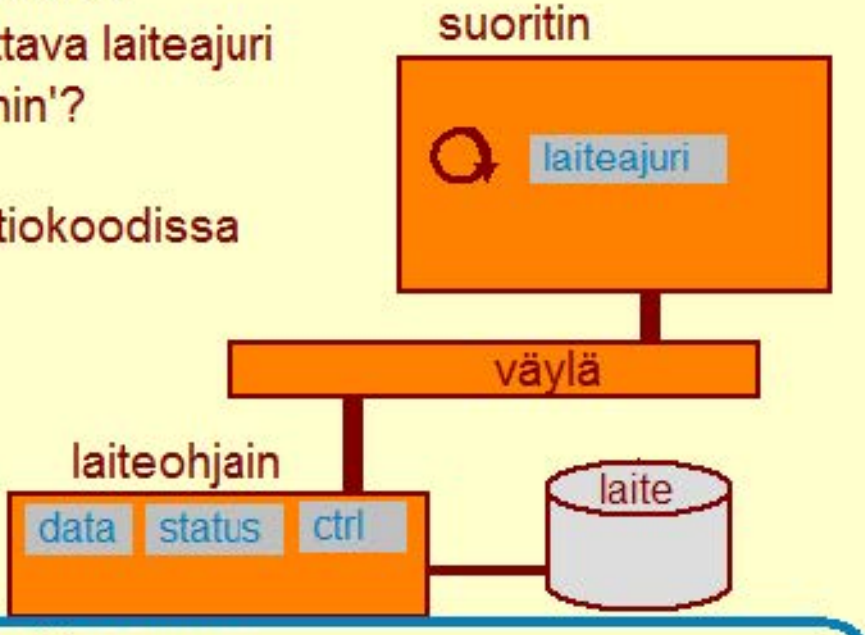
- ei tarvita erillisiä I/O konekäskyjä
- laiteohjaimen "rekisterit" viitattavissa kuten tavallinen muisti
- muistiosoitteen ensimmäiset bitit määräävät, mille laitteelle (tav. muisti vai laiteohjaimella oleva muisti) viite kohdistuu
- vie tilaa normaalilta muistiavaruudelta
- voidaan käyttää rinnan I/O-konekäskyjen kanssa

```
load  r1, =KbdRead
store r1, KbdCtrl
```


Omat konekäskyt kullekin laitetyypille on hyvin yksinkertainen toteuttaa. Ongelmana tässä vaihtoehdossa on, että se sopii vain hyvin yksinkertaisille laitteille, joiden tarkat speksit ovat selvillä jo suorittimen käskykannan suunnitteluvaiheessa. Esimerkiksi perusnäppäimistö voisi olla tällainen laite. Tätä lähestymistapaa on kuitenkin vaikea toteuttaa uusien laitteiden osalta. Jos uusi I/O-laite on kehitetty vasta suorittimen toteutuksen jälkeen, sen kaikki toiminnot pitäisi olla samanlaisia kun kyseisten konekäskyjen toteutuksessa on tehty.

Laiteohjaimen rekistereihin viittaaminen

Ongelma: miten pääsuorittimella suorittava laiteajuri viittaa laiteohjaimella oleviin 'rekistereihin'?



Omat I/O-konekäskyt, toteutus operaatiokoodissa

- IN, OUT
- käskyssä laiteohjaimen (ja laiterekisterin) tunniste
- vaikea laajentaa uusiin laitteisiin
- toteutus käskykannan suorittimen yhteydessä

```
in r1, =KBD
```

Muistiinkuvattu I/O, toteutus osoitteen tulkinnassa

- ei tarvita erillisiä I/O konekäskyjä
- laiteohjaimen "rekisterit" viitattavissa kuten tavallinen muisti
- muistiosoitteen ensimmäiset bitit määräävät, mille laitteelle (tav. muisti vai laiteohjaimella oleva muisti) viite kohdistuu
- vie tilaa normaalilta muistiavaruudelta
- voidaan käyttää rinnan I/O-konekäskyjen kanssa

```
load  r1, =KbdRead
store r1, KbdCtrl
```

```
KbdRead  EQU 1
KbdCtr   EQU 0x80000001
KbdStat  EQU 0x80000002
```


Yleisempi ratkaisu on muistiinkuvattu I/O, jossa osa prosessin käyttämästä osoitteiden joukosta eli osoiteavaruudesta on varattu eri I/O-laitteiden porteille. Muistisoitteen ensimmäiset bitit sitten ratkaisevat, mille laitteelle ja siis minkä laiteohjaimen muistialueelle muistiviite kohdistuu. Tämä ratkaisu on täysin yleinen ja salli uusien minkä tahansa tyyppisten laitteiden liittämisen järjestelmään milloin vain. Menetelmän heikkoutena on tietenkin viitattavissa olevan tavallisen muistin määrän pieneneminen. Jos viitttava fyysinen muistialue valitaan muistiosoitteen kahden ensimmäisen bitin avulla, niin tavallisen viitattavar muistialueen koko supistuu 75%.

I/O tyypit

Suora I/O

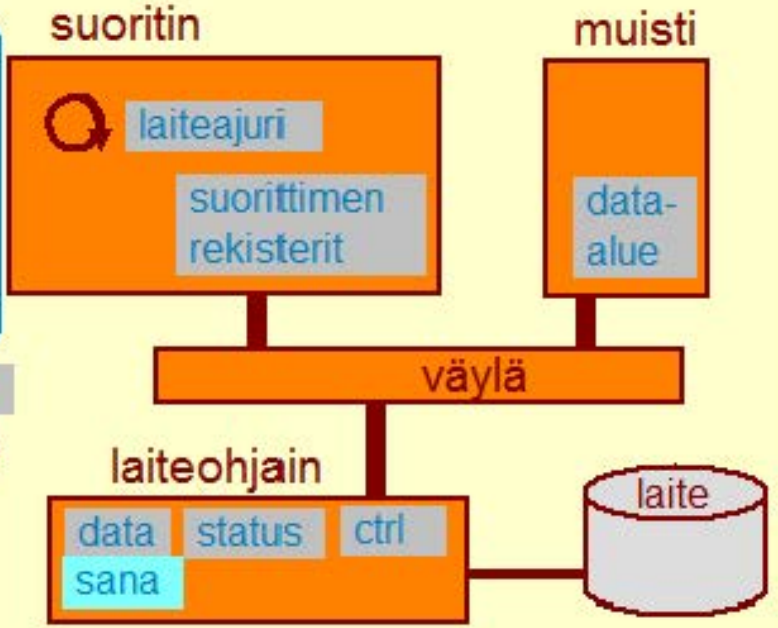
direct I/O, programmed I/O

- ajuri odottaa suorittaen, kunnes laiteohjain statusrekisterin avulla ilmoittaa I/O-tehtävän valmistuneen
- ajuri siirtää tiedon sana kerrallaan muistin ja laiteohjaimen data-rekisterin välillä

Epäsuora I/O

indirect I/O, interrupt driven I/O

- ajuri odottaa odotustilassa, kunnes laiteohjain I/O-laitekeskeytyksen avulla ilmoittaa I/O-tehtävän valmistuneen
- ajuri siirtää tiedon sana kerrallaan muistin ja laiteohjaimen data-rekisterin välillä



DMA

direct memory access I/O

- ajuri antaa laiteohjainprosessille kokonaisia tehtäviä, joihin sisältyy tiedon siirto
- ajuri odottaa odotustilassa, kunnes kokonainen I/O- työ on valmis
- laiteohjainprosessi siirtää tiedon suoraan muistin ja laiteohjaimen datarekisterin välillä
- laiteohjainprosessi ilmoittaa ajurille I/O-työn valmistumisesta I/O-laitekeskeytyksellä
- laiteohjain on selvästi älykkäämpi kuin suorassa/epäsuorassa I/O:ssa

Laiteohjaimia on monenlaisia ja niiden 'älykkyys-asteelle' on ainakin kolme perustasoa. Yksinkertaisin laiteohjain tukee ainoastaan suoraa I/O:ta, jossa laiteajuri tekee lähes kaiken työn ja varaa suorittimen itselleen koko I/O:n ajaksi. Tulostusta varten ajuri ensin asettaa tulostettavan tiedon laiteohjaimen data-rekisteriin ja sitten tulostuskomennon laiteohjaimen kontrollirekisteriin. Tämän jälkeen ajuri odottaa tiukassa loopissa, kunnes se havaitsee laiteohjaimen statusrekisteristä, että annettu I/O-tehtävä on valmis. Ja sitten sama toistuu, kunnes koko I/O-työ on saatu loppuun. Kaikki data kiertää siis suorittimen rekistereiden kautta ja kulkee kaksi kertaa väylän kautta.

I/O tyypit

Suora I/O

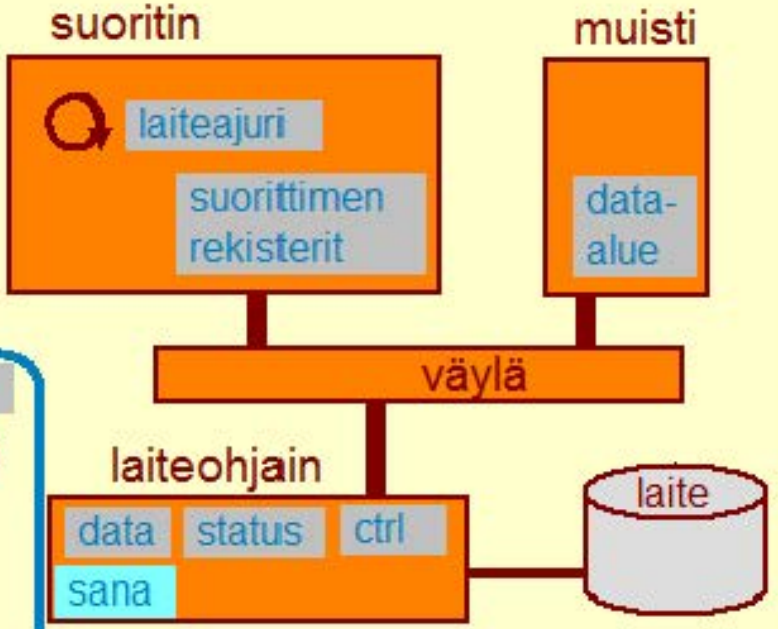
direct I/O, programmed I/O

- ajuri odottaa suorittaen, kunnes laiteohjain statusrekisterin avulla ilmoittaa I/O-tehtävän valmistuneen
- ajuri siirtää tiedon sana kerrallaan muistin ja laiteohjaimen data-rekisterin välillä

Epäsuora I/O

indirect I/O, interrupt driven I/O

- ajuri odottaa odotustilassa, kunnes laiteohjain I/O-laitekeskeytyksen avulla ilmoittaa I/O-tehtävän valmistuneen
- ajuri siirtää tiedon sana kerrallaan muistin ja laiteohjaimen data-rekisterin välillä



DMA

direct memory access I/O

- ajuri antaa laiteohjainprosessille kokonaisia tehtäviä, joihin sisältyy tiedon siirto
- ajuri odottaa odotustilassa, kunnes kokonainen I/O- työ on valmis
- laiteohjainprosessi siirtää tiedon suoraan muistin ja laiteohjaimen datarekisterin välillä
- laiteohjainprosessi ilmoittaa ajurille I/O-työn valmistumisesta I/O-laitekeskeytyksellä
- laiteohjain on selvästi älykkäämpi kuin suorassa/epäsuorassa I/O:ssa

Tästä selvästi fiksumpi laiteohjain osaa myös aiheuttaa I/O-laitekeskeytyksiä eli sen väyläliitoksessa on mukana myös piuhat I/O-laitekeskeytysjohtimille. Ajuri voi nyt I/O-tehtävän annettuaan mennä odotustilaan ja suoritin voi ottaa jonkun muun prosessin suoritukseen I/O-tehtävän suorituksen ajaksi. Lopulta, kun laiteohjaimella suorittava laiteohjainprosessi on saanut annetun I/O-tehtävän valmiiksi, se ensin asettaa tilarekisterin arvon osoittamaan I/O-tehtävän valmistumista ja sitten aiheuttaa I/O-laitekeskeytyksen. Laiteajuri lopettaa odotuksen ja suoritukseen päästyään tutkii laiteohjaimen tilarekisterin ja havaitsee I/O-tehtävän valmistuneen. Ajuri voi sitten jatkaa uudella tehtävällä.

I/O tyypit

Suora I/O

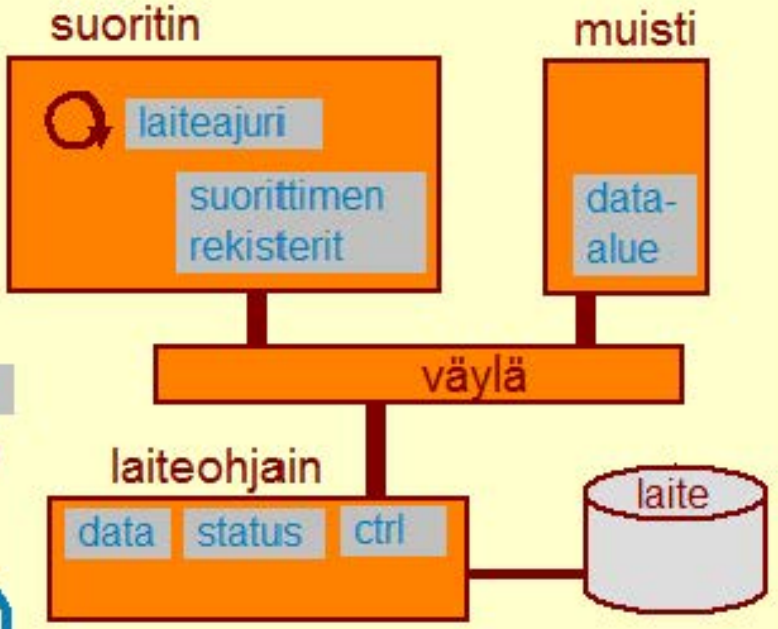
direct I/O, programmed I/O

- ajuri odottaa suorittaen, kunnes laiteohjain statusrekisterin avulla ilmoittaa I/O-tehtävän valmistuneen
- ajuri siirtää tiedon sana kerrallaan muistin ja laiteohjaimen data-rekisterin välillä

Epäsuora I/O

indirect I/O, interrupt driven I/O

- ajuri odottaa odotustilassa, kunnes laiteohjain I/O-laitekeskeytyksen avulla ilmoittaa I/O-tehtävän valmistuneen
- ajuri siirtää tiedon sana kerrallaan muistin ja laiteohjaimen data-rekisterin välillä



DMA

direct memory access I/O

- ajuri antaa laiteohjainprosessille kokonaisia tehtäviä, joihin sisältyy tiedon siirto
- ajuri odottaa odotustilassa, kunnes kokonainen I/O- työ on valmis
- laiteohjainprosessi siirtää tiedon suoraan muistin ja laiteohjaimen datarekisterin välillä
- laiteohjainprosessi ilmoittaa ajurille I/O-työn valmistumisesta I/O-laitekeskeytyksellä
- laiteohjain on selvästi älykkäämpi kuin suorassa/epäsuorassa I/O:ssa

Myös epäsuorassa I/O:ssa tieto siirtyy muistin ja laiteohjaimen välillä ainostaan ajurin toimesta. Kaikki tieto kulkee suoritimen rekistereiden kautta sana kerrallaan. Tiedon siirrossa tarvitaan myös väylää kaksi kertaa, sekä muistin ja suorittimen rekistereiden välillä että suorittimen rekistereiden ja laiteohjaimen data-rekisterin välillä. Tämä on selvästi sekä suoran I/O:n että epäsuoran I/O:n yksi heikkous. Toisaalta, monessa yksinkertaisessa laitteessa, kuten vaikkapa kämmenmikrossa tai kännykässä, tällainen yksinkertainen I/O on aivan tarkoituksenmukainen. Se sopii myös oikein hyvin vaikkapa pöytäkoneen näppäimistölle.

I/O tyypit

Suora I/O

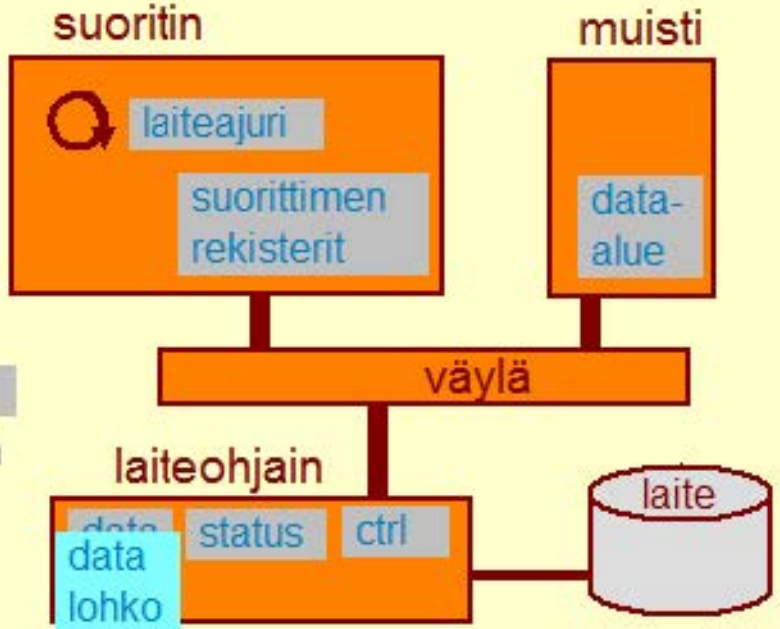
direct I/O, programmed I/O

- ajuri odottaa suorittaen, kunnes laiteohjain statusrekisterin avulla ilmoittaa I/O-tehtävän valmistuneen
- ajuri siirtää tiedon sana kerrallaan muistin ja laiteohjaimen data-rekisterin välillä

Epäsuora I/O

indirect I/O, interrupt driven I/O

- ajuri odottaa odotustilassa, kunnes laiteohjain I/O-laitekeskeytyksen avulla ilmoittaa I/O-tehtävän valmistuneen
- ajuri siirtää tiedon sana kerrallaan muistin ja laiteohjaimen data-rekisterin välillä



DMA

direct memory access I/O

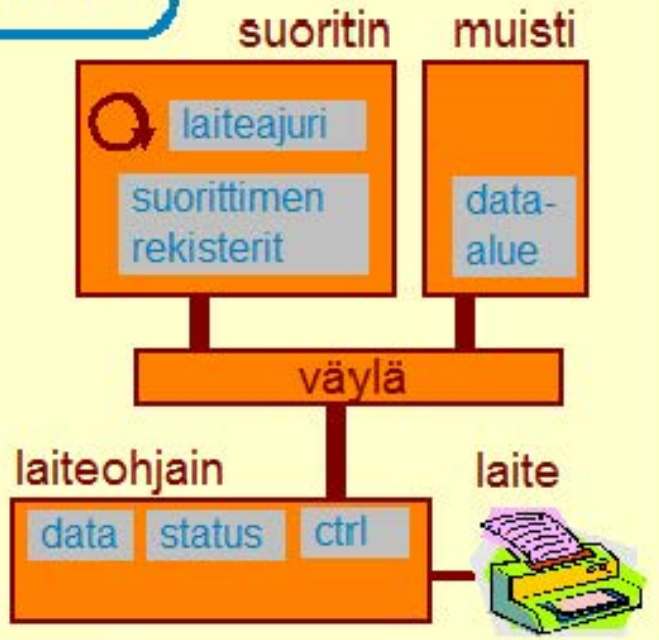
- ajuri antaa laiteohjainprosessille kokonaisia tehtäviä, joihin sisältyy tiedon siirto
- ajuri odottaa odotustilassa, kunnes kokonainen I/O- työ on valmis
- laiteohjainprosessi siirtää tiedon suoraan muistin ja laiteohjaimen datarekisterin välillä
- laiteohjainprosessi ilmoittaa ajurille I/O-työn valmistumisesta I/O-laitekeskeytyksellä
- laiteohjain on selvästi älykkäämpi kuin suorassa/epäsuorassa I/O:ssa

DMA-laiteohjaimet ovat selvästi aikaisempia kehittyneempiä. Ne toimivat rinnakkain ajurin kanssa ja suorittavat itsenäisesti huomattavasti laajempia I/O-tehtäviä. DMA-laitteelle voidaan antaan kokonaisia I/O-töitä, jolloin laiteajurille jää paljon vähemmän tekemistä, jolloin taas vapautunutta suoritinaikaa voidaan käyttää muiden prosessien ajamiseen. DMA-ohjain osaa käsitellä myös keskusmuistia ja siten siirtää tietoa suoraan muistin ja laiteohjaimen välillä. Tästä on myös se merkittävä etu, että väylää tarvitaan vain yhden kerran kunkin sanan tiedonsiirtoon. DMA-ohjaimen muistiväylän käyttö aiheuttaa kilpailua muistiväylästä ja sillä tavoin vähän hidastaa suorittimen toimintaa

Laiteajuri esimerkki: kirjoittimen laiteajuri ttk-91 -koneelle

Laitteella voidaan tulostaa kokonaislukuja yksi kerrallaan

Muistiinkuvattu, suora I/O,
ajuri etuoikeutetussa tilassa



Laiteportti

- kontrollirekisteri,
  muistiosoite 0x100000 = 1048576
- status- eli tilarekisteri,
  muistiosoite 0x100001 = 1048577
- datarekisteri,
  muistiosoite 0x100002 = 1048578

Kutsu:

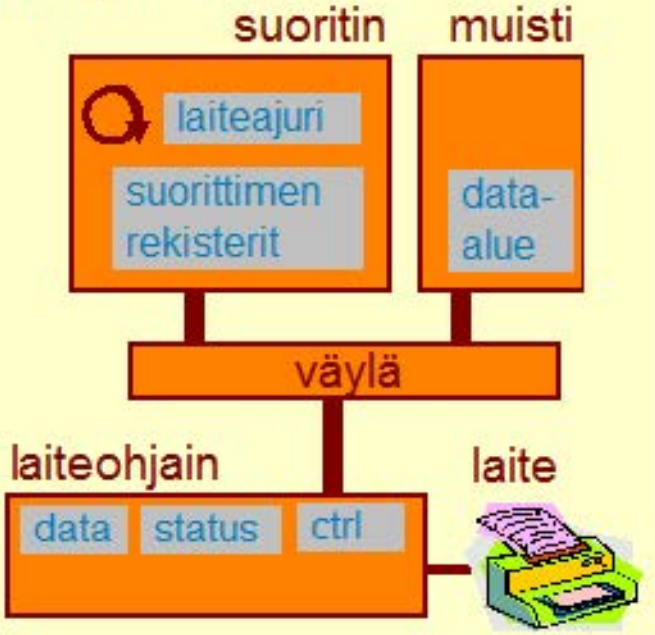
```
push   sp, =0       ; space for return value
push   sp, X        ; parameter value to be printed
svc    sp, =print   ; returns success=0 or failure=1
pop    sp, r1       ; return value
jnzer  r1, TakeCareOfTrouble ; problem handling
```


Ajuri on siis käyttöjärjestelmään kuuluva ohjelmiston osa, joka osaa ohjata laiteohjaimella suorittavaa laiteohjainprosessia tekemään I/O:n. Tässä esimerkissä teteutamme ajurin uudelle ttk-91 -koneen tulostimelle, joka osaa tulostaa annettuja kokonaislukuja yksi kerrallaan. Joskus käytössä olleet printerit olivat vielä yksinkertaisempia: ne pystyvät tulostamaan vain yhden merkin kerrallaan.

Laiteajuri esimerkki: kirjoittimen laiteajuri ttk-91 -koneelle

Laitteella voidaan tulostaa kokonaislukuja yksi kerrallaan

Muistiinkuvattu, suora I/O,
ajuri etuoikeutetussa tilassa



Laiteportti

- kontrollirekisteri,
  muistiosoite 0x100000 = 1048576
- status- eli tilarekisteri,
  muistiosoite 0x100001 = 1048577
- datarekisteri,
  muistiosoite 0x100002 = 1048578

Kutsu:

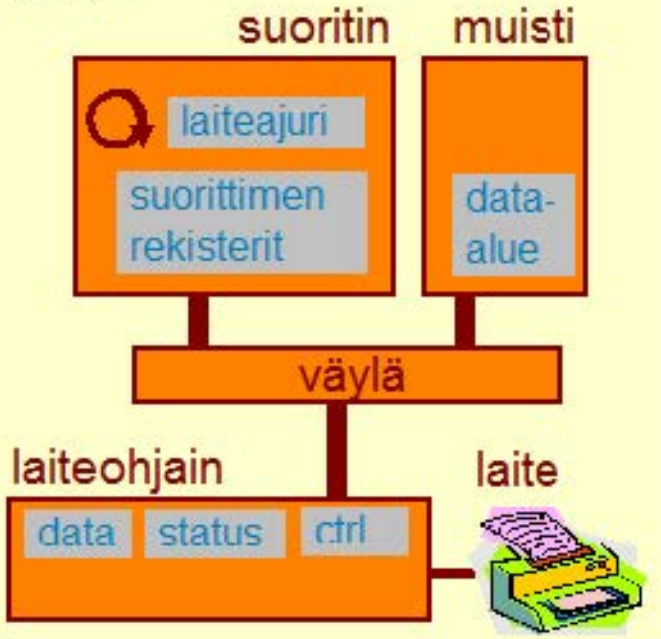
```
push   sp, =0       ; space for return value
push   sp, X        ; parameter value to be printed
svc    sp, =print   ; returns success=0 or failure=1
pop    sp, r1       ; return value
jnzer  r1, TakeCareOfTrouble ; problem handling
```


Ajuri toteutetaan käyttäen muistiinkuvattua I/O:ta, mikä siis tarkoittaa, että ajuri ohjaa laiteohjaimen toimintaa tavallisilla muistiinviittauskäskyillä. Osa muistiavaruudesta on nyt varattu I/O-laitteiden porteille ja tavallista viitattavaa muistia on siltä osin vähemmän. Tulostus toteutetaan mahdollisimman yksinkertaisesti suorana I/O:na, jolloin ajuri pitää suorittimen hallussaan koko tulostustehtävän ajan. Laiteohjaimen rekisterit on toteutettu sellaiseen osaan muistiavaruutta, johon voi viitata ainoastaan etuoikeutetussa tilassa, joten myös laiteajurin täytyy suorittaa etuoikeutetussa tilassa.

Laiteajuri esimerkki: kirjoittimen laiteajuri ttk-91 -koneelle

Laitteella voidaan tulostaa kokonaislukuja yksi kerrallaan

Muistiinkuvattu, suora I/O,
ajuri etuoikeutetussa tilassa



Laiteportti

- kontrollirekisteri,
  muistiosoite 0x100000 = 1048576
- status- eli tilarekisteri,
  muistiosoite 0x100001 = 1048577
- datarekisteri,
  muistiosoite 0x100002 = 1048578

Kutsu:

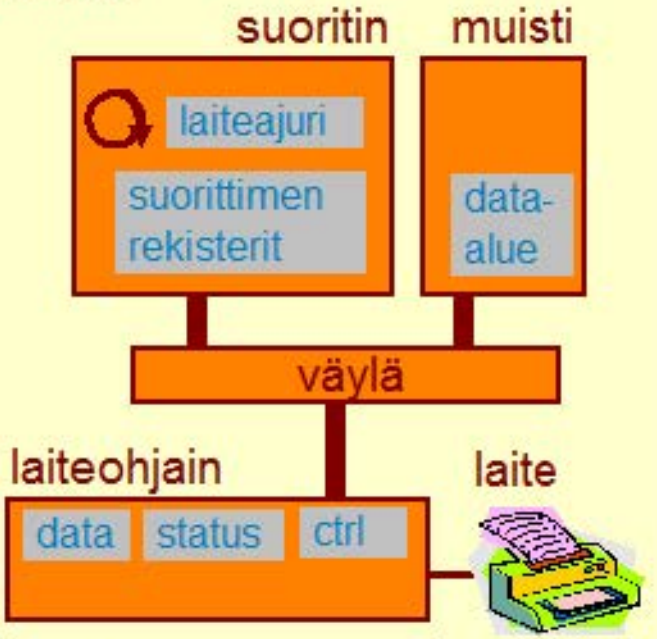
```
push   sp, =0       ; space for return value
push   sp, X        ; parameter value to be printed
svc    sp, =print   ; returns success=0 or failure=1
pop    sp, r1       ; return value
jnzer  r1, TakeCareOfTrouble ; problem handling
```


Laitteen portti on toteutettu kolmen yhden sanan 'rekisterin' avulla, muistiosoitteissa 0x100000-0x100002. Osoitteet ovat liian suuria mahtuakseen ttk-91 koneen konekäskyn osoiteosaan, joten niissä olevaan tietoon tulee aina viitata epäsuoran tiedonosoitusmoodin kautta. Muutoin niihin viittaminen tapahtuu ihan samoin kuin muihinkin muistipaikkoihin. Tässä on yksinkertaisuuden vuoksi otaksuttu, että ttk-91 koneen osoitevaruus olisi 21-bittinen, jolloin I/O-porttien osoitteissa bitti 21 on 1.

Laiteajuri esimerkki: kirjoittimen laiteajuri ttk-91 -koneelle

Laitteella voidaan tulostaa kokonaislukuja yksi kerrallaan

Muistiinkuvattu, suora I/O,
ajuri etuoikeutetussa tilassa



Laiteportti

- kontrollirekisteri,
  muistiosoite 0x100000 = 1048576
- status- eli tilarekisteri,
  muistiosoite 0x100001 = 1048577
- datarekisteri,
  muistiosoite 0x100002 = 1048578

Kutsu:

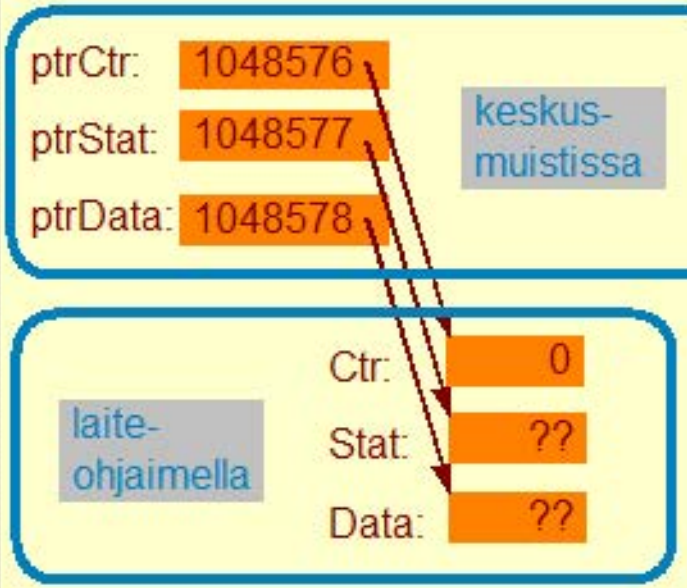
```
push   sp, =0       ; space for return value
push   sp, X        ; parameter value to be printed
svc    sp, =print   ; returns success=0 or failure=1
pop    sp, r1       ; return value
jnzer  r1, TakeCareOfTrouble ; problem handling
```


Halutessaan tulostaa kokonaisluvun uudella printterillä sovellusohjelma kutsuu ajuria tekemään työn. Koska ajurin pitää suorittaa koodinsa etuoikeutetussa tilassa, pitää kutsun tapahtua SVC-käskyllä tavallisen aliohjelmakutsun asemesta. Oletamme tässä, että parametrien ja paluu-arvon välitys tapahtuu SVC:llä samalla tavalla kuin tavallisten aliohjelmien yhteydessä. Itse ajurin toteutus näytetään seuraavalla sivulla.

Esimerkki: ttk-91 -koneen kirjoittimen laiteajurin toteutus

Oletus: muistisoitteissa korkeintaan 21 merkitsevää bittiä

Oletus: SVC- ja IRET-toteutukset samalla tavalla kuin CALL ja EXIT




```
ptrCtr   DC 1048576     ; control reg address
ptrStat  DC 1048577     ; status reg address
ptrData  DC 1048578     ; data reg address
retVal   EQU -3   ; return value addr in activation record
parData  EQU -2   ; param addr in activation record

print   push sp, r1          ; save r1
        load  r1, parData(FP)
        store r1, @ptrData   ; put data into data register
        load  r1, =0
        store r1, @ptrStat   ; clear status register
        load  r1, =1
        store r1, @ptrCtr    ; give print command
wait    load r1, @ptrStat    ; check status register
        jzer  r1, wait       ; not ready, wait
        load  r1, =0         ; return 'success'
        store  r1, retval(FP)
        pop    sp, r1        ; recover r1
        iret   sp, =1
```

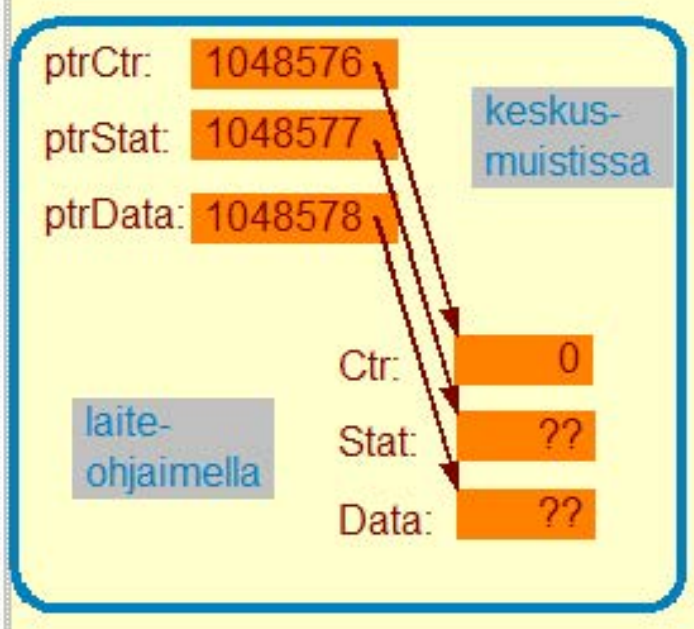
Ei virheitä, ei time-out'ia!
Täydellisempi toteutus esimerkkiohjelmssa driver.k91

Ajurin toteutuksessa lähtökohtana on siis, että otaksumme SVC- ja IRET-käskyjen olevan toteutettu vastaavalla tavalla kuin aliohjelmien kutsu- ja paluukäskyt. Erona vain on kutsuttavan rutiinin nimeämistapa ja suorittimen suoritustilan vaihto SVC:n yhteydessä etuoikeutettuun ja takaisin aikaisempaan IRET-käskyn yhteydessä. Viitattavaa muistia on nyt siis sekä keskusmuistissa että laiteohjaimella. Molempien laitteiden 21-bittiset muistiosoitteet näyttävät samanlaisilta, mutta laiteohjaimilla ensimmäinen bitti on 1 kun tavallisilla keskumuistiosoitteilla se on 0.

Esimerkki: ttk-91 -koneen kirjoittimen laiteajurin toteutus

Oletus: muistisoitteissa korkeintaan 21 merkitsevää bittiä

Oletus: SVC- ja IRET-toteutukset samalla tavalla kuin CALL ja EXIT




```
ptrCtr   DC 1048576     ; control reg address
ptrStat  DC 1048577     ; status reg address
ptrData  DC 1048578     ; data reg address
retVal   EQU -3   ; return value addr in activation record
parData  EQU -2   ; param addr in activation record

print   push sp, r1          ; save r1
        load  r1, parData(FP)
        store r1, @ptrData   ; put data into data register
        load  r1, =0
        store r1, @ptrStat   ; clear status register
        load  r1, =1
        store r1, @ptrCtr    ; give print command
wait    load r1, @ptrStat    ; check status register
        jzer  r1, wait       ; not ready, wait
        load  r1, =0         ; return 'success'
        store  r1, retval(FP)
        pop    sp, r1        ; recover r1
        iret   sp, =1
```

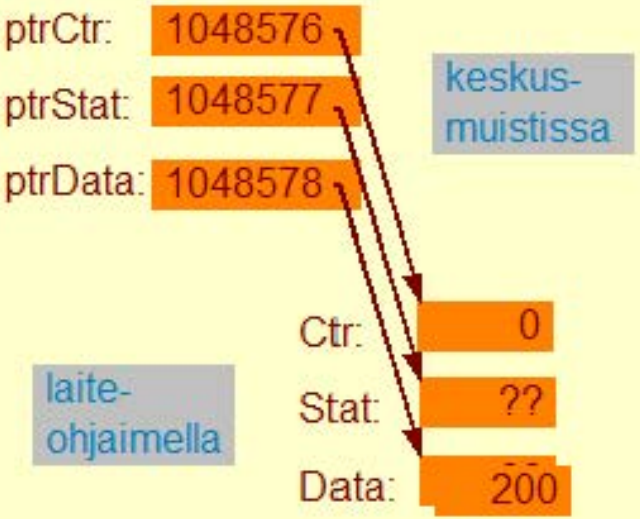
Ei virheitä, ei time-out'ia!
Täydellisempi toteutus esimerkkiohjelmssa driver.k91

Palvelurutiinin Print määrittelyt ovat hyvin samanlaisia kuin minkä tahansa aliohjelman. Laitteen I/O-portin osoitteet määritellään ensin suurina vakioina, joiden kautta laiteohjaimen rekistereihin päästään myöhemmin käsiksi epäsuoran viittauksen avulla. Arvoparametri parData:ssa on tulostettava luku ja Print palauttaa arvonaan retVal luvun 0 tulostuksen onnistuttua ja luvun 1 muutoin. Tässä yksinkertaistetussa esimerkissä tulostus aina onnistuu, joten paluuarvoksi tulee aina lopulta 0.

Esimerkki: ttk-91 -koneen kirjoittimen laiteajurin toteutus

Oletus: muistisoitteissa korkeintaan 21 merkitsevää bittiä

Oletus: SVC- ja IRET-toteutukset samalla tavalla kuin CALL ja EXIT




```
ptrCtr   DC 1048576     ; control reg address
ptrStat  DC 1048577     ; status reg address
ptrData  DC 1048578     ; data reg address
retVal   EQU -3  ; return value addr in activation record
parData  EQU -2  ; param addr in activation record

print    push sp, r1          ; save r1
         load  r1, parData(FP)
         store r1, @ptrData   ; put data into data register
         load  r1, =0
         store r1, @ptrStat   ; clear status register
         load  r1, =1
         store r1, @ptrCtr    ; give print command
wait     load r1, @ptrStat    ; check status register
         jzer  r1, wait       ; not ready, wait
         load  r1, =0         ; return 'success'
         store  r1, retval(FP)
         pop    sp, r1        ; recover r1
         iret   sp, =1
```

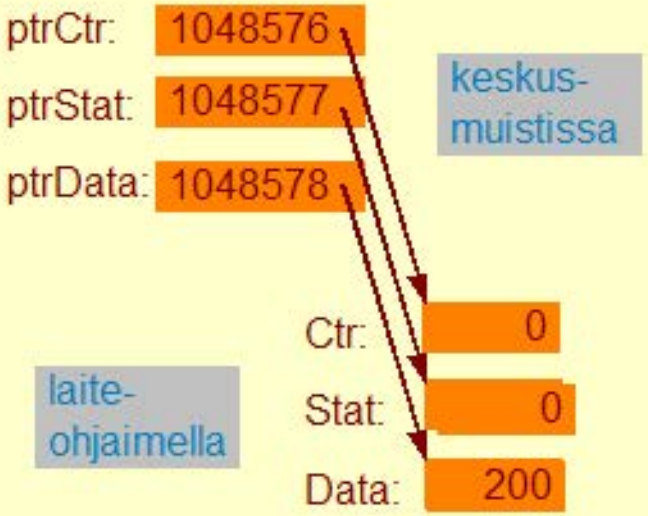
Ei virheitä, ei time-out'ia!
Täydellisempi toteutus esimerkkiohjelmssa driver.k91

Rutiini Print aloittaa tallettamalla ainoan käyttämänsä rekisterin (r1) arvon pinoon, jonka jälkeen se kopioi parametrina annetun luvun arvon laiteohjaimen data-rekisteriin. Laiteohjaimen kontrollirekisterin arvo on nolla, mikä ilmaisee laitteen olevan vapaa ja valmis annettavan tehtävän suoritukseen. Todellisuudessa laiteajurin pitäisi tietenkin tarkistaa laitteen tila, ja ehkä käynnistää se tai tehdä jotain muita toimintoja, jos tilana ei ole 'valmis suoritukseen'. Yksinkertaisuuden vuoksi nämä tarkistukset on jätetty tässä pois.

Esimerkki: ttk-91 -koneen kirjoittimen laiteajurin toteutus

Oletus: muistisoitteissa korkeintaan 21 merkitsevää bittiä

Oletus: SVC- ja IRET-toteutukset samalla tavalla kuin CALL ja EXIT




```
ptrCtr  DC 1048576     ; control reg address
ptrStat DC 1048577     ; status reg address
ptrData DC 1048578     ; data reg address
retVal  EQU -3  ; return value addr in activation record
parData EQU -2  ; param addr in activation record

print   push sp, r1          ; save r1
        load  r1, parData(FP)
        store r1, @ptrData   ; put data into data register
        load  r1, =0
        store r1, @ptrStat   ; clear status register
        load  r1, =1
        store r1, @ptrCtr    ; give print command
wait    load r1, @ptrStat    ; check status register
        jzer  r1, wait       ; not ready, wait
        load  r1, =0         ; return 'success'
        store  r1, retval(FP)
        pop    sp, r1        ; recover r1
        iret   sp, =1
```

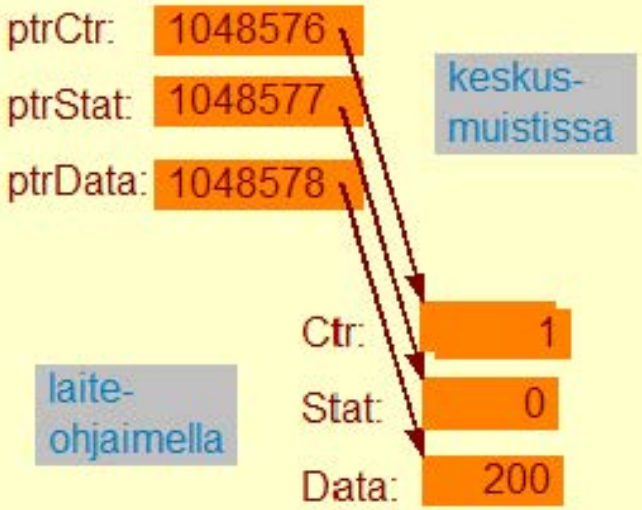
Ei virheitä, ei time-out'ia!
Täydellisempi toteutus esimerkkiohjelmssa driver.k91

Seuraavaksi statusrekisterin arvo nollataan, koska laiteohjainprosessin tiedetään asettavan se arvoon 1, kun annettu I/O-tehtävä on tehty. Haluamme varmistua siitä, että alkuaan statusrekisteri indikoi tilaa 'ei valmis'.

Esimerkki: ttk-91 -koneen kirjoittimen laiteajurin toteutus

Oletus: muistisoitteissa korkeintaan 21 merkitsevää bittiä

Oletus: SVC- ja IRET-toteutukset samalla tavalla kuin CALL ja EXIT




```
ptrCtr   DC 1048576     ; control reg address
ptrStat DC  1048577     ; status reg address
ptrData DC 1048578      ; data reg address
retVal   EQU  -3  ; return value addr in activation record
parData EQU -2  ; param addr in activation record

print   push sp, r1          ; save r1
        load  r1, parData(FP)
        store r1, @ptrData  ; put data into data register
        load  r1, =0
        store r1, @ptrStat    ; clear status register
        load  r1, =1
        store r1, @ptrCtr      ; give print command
wait    load r1, @ptrStat     ; check status register
        jzer  r1, wait          ; not ready, wait
        load  r1, =0            ; return 'success'
        store  r1, retval(FP)
        pop    sp, r1           ; recover r1
        iret    sp, =1
```

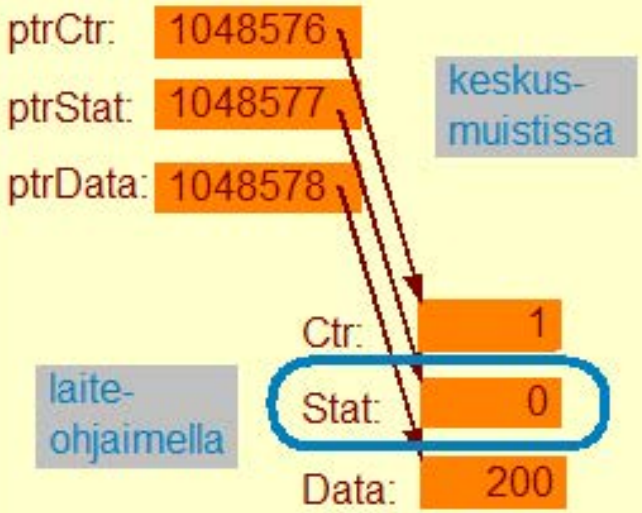
Ei virheitä, ei time-out'ia!
Täydellisempi toteutus esimerkkiohjelmssa driver.k91

Lopulta ajuri antaa laiteohjaimelle (eli siis laiteohjainprosessille) tulostuskomennon kirjoittamalla arvon 1 kontrollirekisteriin. Laiteohjainprosessi on koko ajan odottanut tiukassa loopissa vain lukien tämän kontrollirekisterin arvoa, ja havaitsee välittömästi uuden työtehtävän ilmaantuneen. Laiteohjainprosessi aloittaa tulostustehtävän, joka suorittimen ja ajurin mittakaavassa kestää ehkä hyvinkin kauan aikaa, vaikkapa 2-50 miljoonan konekäskyn suorittamisen ajan. Tiedämme ainoastaan, että tulostuksen valmistuttua laiteohjainprosessi kirjoittaa luvun 1 laiteohjaimen tilarekisteriin.

Esimerkki: ttk-91 -koneen kirjoittimen laiteajurin toteutus

Oletus: muistisoitteissa korkeintaan 21 merkitsevää bittiä

Oletus: SVC- ja IRET-toteutukset samalla tavalla kuin CALL ja EXIT




```
ptrCtr   DC 1048576     ; control reg address
ptrStat DC  1048577     ; status reg address
ptrData DC 1048578      ; data reg address
retVal   EQU  -3   ; return value addr in activation record
parData EQU -2   ; param addr in activation record

print   push sp, r1          ; save r1
        load  r1, parData(FP)
        store r1, @ptrData  ; put data into data register
        load  r1, =0
        store r1, @ptrStat    ; clear status register
        load  r1, =1
        store r1, @ptrCtr      ; give print command
wait    load r1, @ptrStat     ; check status register
        jzer  r1, wait          ; not ready, wait
        load  r1, =0            ; return 'success'
        store  r1, retval(FP)
        pop    sp, r1           ; recover r1
        iret    sp, =1
```

Ei virheitä, ei time-out'ia!
Täydellisempi toteutus esimerkkiohjelmssa driver.k91

Laiteajuri siirtyy odottamaan laiteohjainprosessin viestiä ja lukee tiukassa loopissa koko ajan tilarekisterin arvoa. Odotus loopissa jatkuu, niin kauan kun tilarekisterin arvo pysyy nollana. Tulostuksen yhteydessä odotus kestää ehkä vain millisekunteja tai kymmeniä millisekunteja. Syötelaitteiden kuten esimerkiksi näppäimistön yhteydessä odotus voi olla sekunteja, minuutteja tai tunteja, koska laitteen nopeus on oikeasti kiinni laitetta käyttävästä ihmisestä.

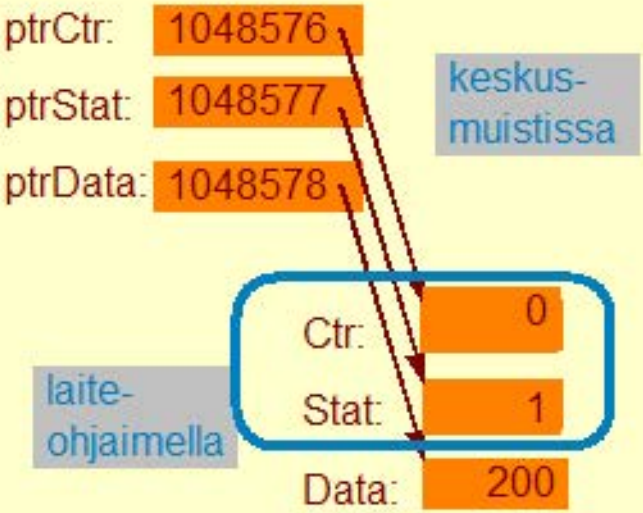
Esimerkki: ttk-91 -koneen kirjoittimen laiteajurin toteutus

Oletus: SVC- ja IRET-toteutukset samalla tavalla kuin CALL ja EXIT

Oletus: muistisoitteissa korkeintaan 21 merkitsevää bittiä

```
ptrCtr   DC 1048576     ; control reg address
ptrStat  DC 1048577     ; status reg address
ptrData  DC 1048578     ; data reg address
retVal   EQU -3   ; return value addr in activation record
parData  EQU -2   ; param addr in activation record

print    push sp, r1          ; save r1
         load  r1, parData(FP)
         store r1, @ptrData   ; put data into data register
         load  r1, =0
         store r1, @ptrStat   ; clear status register
         load  r1, =1
         store r1, @ptrCtr    ; give print command
wait     load r1, @ptrStat    ; check status register
         jzer  r1, wait       ; not ready, wait
         load  r1, =0         ; return 'success'
         store  r1, retval(FP)
         pop    sp, r1        ; recover r1
         iret    sp, =1
```



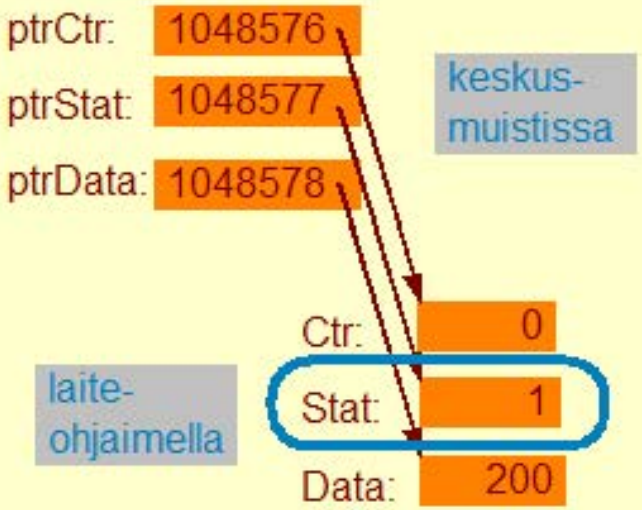
Ei virheitä, ei time-out'ia!
Täydellisempi toteutus esimerkkiohjelmssa driver.k91

Lopulta laiteohjaimella suorittava laiteohjainprosessi saa työnsä valmiiksi eli se on tulostanut annetun luvun. Se signaloi asiasta laiteajurille sovitulla tavalla eli asettamalla laiteohjaimen tilarekisterin arvoksi 1.
Laiteohjainprosessi asettaa myös kontrollirekisterin arvoksi nolla. Tällä tavoin se ilmaisee laiteajurille, että se on valmis seuraavaan tehtävään.

Esimerkki: ttk-91 -koneen kirjoittimen laiteajurin toteutus

Oletus: muistisoitteissa korkeintaan 21 merkitsevää bittiä

Oletus: SVC- ja IRET-toteutukset samalla tavalla kuin CALL ja EXIT




```
ptrCtr   DC 1048576     ; control reg address
ptrStat  DC 1048577     ; status reg address
ptrData  DC 1048578     ; data reg address
retVal   EQU -3  ; return value addr in activation record
parData  EQU -2  ; param addr in activation record

print    push sp, r1          ; save r1
         load  r1, parData(FP)
         store r1, @ptrData   ; put data into data register
         load  r1, =0
         store r1, @ptrStat   ; clear status register
         load  r1, =1
         store r1, @ptrCtr    ; give print command
wait     load r1, @ptrStat    ; check status register
         jzer  r1, wait       ; not ready, wait
         load  r1, =0         ; return 'success'
         store  r1, retval(FP)
         pop    sp, r1        ; recover r1
         iret   sp, =1
```

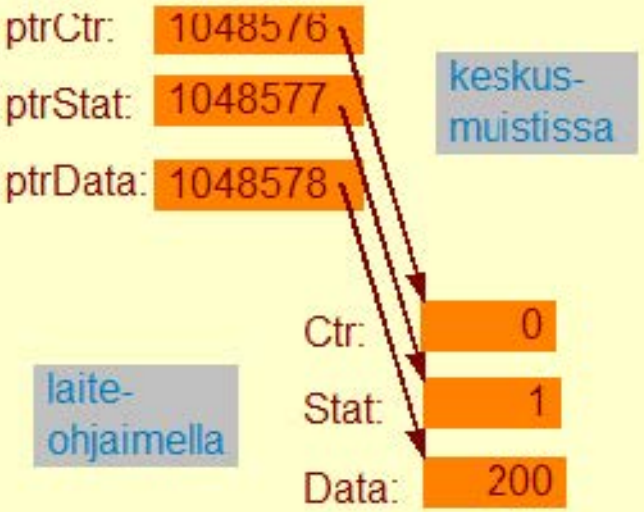
Ei virheitä, ei time-out'ia!
Täydellisempi toteutus esimerkkiohjelmssa driver.k91

Laiteajuri havaitsee nyt puolestaan tämän hyvin nopeasti, koska se ei muuta ole tehnytkään kuin pollannut tilarekisterin arvoa. Se voi nyt poistua odotusloopista ja palauttaa kontrollin kutsuvalle rutiinille.

Esimerkki: ttk-91 -koneen kirjoittimen laiteajurin toteutus

Oletus: muistisoitteissa korkeintaan 21 merkitsevää bittiä

Oletus: SVC- ja IRET-toteutukset samalla tavalla kuin CALL ja EXIT




```
ptrCtr   DC 1048576     ; control reg address
ptrStat DC  1048577     ; status reg address
ptrData DC 1048578      ; data reg address
retVal   EQU  -3  ; return value addr in activation record
parData EQU -2  ; param addr in activation record

print   push sp, r1          ; save r1
        load  r1, parData(FP)
        store r1, @ptrData  ; put data into data register
        load  r1, =0
        store r1, @ptrStat    ; clear status register
        load  r1, =1
        store r1, @ptrCtr      ; give print command
wait    load r1, @ptrStat     ; check status register
        izer  r1, wait          ; not ready, wait
        load  r1, =0            ; return 'success'
        store  r1, retval(FP)
        pop    sp, r1           ; recover r1
        iret    sp, =1
```

Ei virheitä, ei time-out'ia!
Täydellisempi toteutus esimerkkiohjelmssa driver.k91

Palvelurutiini Print asettaa nyt paluuarvokseen tulostuksen onnistumista indikoivan arvon 0, palauttaa työrekisterin r1 arvon ennalleen, ja siirtää kontrollin takaisin kutsuvaan sovellukseen. Iret-käskyn yhteydessä suorittimen tila palaa ennalleen eli käyttäjätilaan normaalien sovellusten ollessa kyseessä. Paluun yhteydessä pinosta otettiin vielä pois alkuperäisen parametrin viemä tila.

Tiedostojärjestelmä, tiedostojen hallintajärjestelmä

Osa käyttöjärjestelmää, hallitsee tiedostojen käyttöä

Valvoo oikeuksia tiedostoja avattaessa

- jotkut tiedostojärjestelmät tarkistavat käyttöoikeudet joka viitteellä

Muuntaa tiedostonimet fyysisiksi laiteosoitteiksi

Ylläpitää taulukoita, joista näkee, mitä kohtaa tiedostoa kukin prosessi on käsittelemässä

- sama tiedosto voi olla auki usealle prosessi samanaikaisesti

Lukee ja kirjoittaa tiedostoja suurina lohkoina (0.5-8 KB?)

- käyttäjätason prosessit käsittelevät tiedostoja tavuina, eikä niiden tarvitse tietää tiedostojen fyysistä rakennetta tai sijaintia
- pitää yllä suuria puskurialueita (jopa 50% keskusmuistista) auki oleville tiedostoille

Edellä kävimme läpi, miten kovalevyt toimivat. Levyillä olevaa tietoa käsitellään käyttöjärjestelmän tiedostojenhallintajärjestelmän eli tiedostojärjestelmän kautta. Tiedostojärjestelmään on keskitetty kaikki tiedostojen hallintaan liittyvät käyttöjärjestelmän palvelut. Tiedostojen hallintajärjestelmään voi itse asiassa kuulua useampikin tiedostojärjestelmä, jos yhdessä ja samassa tietokonejärjestelmässä käytetään useita erilaisia tiedostojärjestelmiä. Esimerkiksi luennoijan pöytäkoneen Windows-ympäristössä systeemi-partitio on NTFS, mutta user-partitiota sekä levykkeitä käytetään FAT-tiedostojärjestelmän kautta.

Tiedostojärjestelmä, tiedostojen hallintajärjestelmä

Osa käyttöjärjestelmää, hallitsee tiedostojen käyttöä

Valvoo oikeuksia tiedostoja avattaessa -rwxr-xr-x kerola grpb index.html

- jotkut tiedostojärjestelmät tarkistavat käyttöoikeudet joka viitteellä

Muuntaa tiedostonimet fyysisiksi laiteosoitteiksi

Ylläpitää taulukoita, joista näkee, mitä kohtaa tiedostoa kukin prosessi on käsittelemässä

- sama tiedosto voi olla auki usealle prosessi samanaikaisesti

Lukee ja kirjoittaa tiedostoja suurina lohkoina (0.5-8 KB?)

- käyttäjätason prosessit käsittelevät tiedostoja tavuina, eikä niiden tarvitse tietää tiedostojen fyysistä rakennetta tai sijaintia
- pitää yllä suuria puskurialueita (jopa 50% keskusmuistista) auki oleville tiedostoille

Tiedostojärjestelmä pitää kirjaa tiedostojen pääsynvalvonnasta. Useimmiten pääsy kontrolloidaan ainoastaan tiedostoa avattaessa, ja sitten myöhemmin tiedoston käytön yhteydessä valvotaan ainoastaan, että käyttötapa ei poikkea tiedoston avaamisen yhteydessä annetusta. Tämä ei aina riitä, koska joissakin (esimerkiksi sotilas-) organisaatioissa halutaan tiukempaa ja täsmällisempää tiedostojen pääsynvalvontaa. Esimerkiksi, jos henkilön lukuoikeus tiedostoihin poistetaan, niin sen tulisi tulla voimaan heti myös kaikille aukioleville tiedostoille.

Tiedostojärjestelmä, tiedostojen hallintajärjestelmä

Osa käyttöjärjestelmää, hallitsee tiedostojen käyttöä

Valvoo oikeuksia tiedostoja avattaessa

- jotkut tiedostojärjestelmät tarkistavat käyttöoikeudet joka viitteellä

Muuntaa tiedostonimet fyysisiksi laiteosoitteiksi

D:\user\kerola\kurssit\tito\luento.doc

↓

Samsung SP1604N, 30GB partitio
levy 2, pinta 1, ura 3335, sektori 55

Ylläpitää taulukoita, joista näkee, mitä kohtaa tiedostoa kukin prosessi on käsittelemässä

- sama tiedosto voi olla auki usealle prosessi samanaikaisesti

Lukee ja kirjoittaa tiedostoja suurina lohkoina (0.5-8 KB?)

- käyttäjätason prosessit käsittelevät tiedostoja tavuina, eikä niiden tarvitse tietää tiedostojen fyysistä rakennetta tai sijaintia
- pitää yllä suuria puskurialueita (jopa 50% keskusmuistista) auki oleville tiedostoille

Tiedostojärjestelmä antaa mukavar käyttöliittymän levyllä oleviin tiedostoihin. Käyttäjän (siis ohjelman) ei tarvitse tietää mitään tiedoston sisäisestä rakenteesta tai laittestotaltion toteutuksesta. Tiedoston käyttämiseksi sovellukselle riittää tietää sen sijainti hakemistopuussa eli yksinkertaisesti tiedoston nimi. Tiedoston avaamisen yhteydessä tiedostojärjestelmä muuttaa nimen fyysiseksi laitteisto-osoitteeksi (levy, pinta, ura, sektori).

Tiedostojärjestelmä, tiedostojen hallintajärjestelmä

Osa käyttöjärjestelmää, hallitsee tiedostojen käyttöä

Valvoo oikeuksia tiedostoja avattaessa

- jotkut tiedostojärjestelmät tarkistavat käyttöoikeudet joka viitteellä

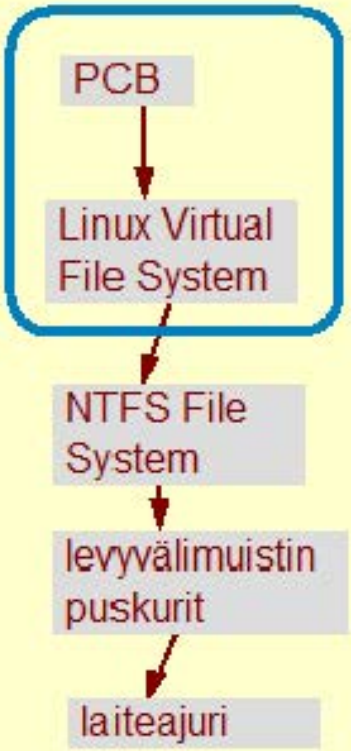
Muuntaa tiedostonimet fyysisiksi laiteosoitteiksi

Ylläpitää taulukoita, joista näkee, mitä kohtaa tiedostoa kukin prosessi on käsittelemässä

- sama tiedosto voi olla auki usealle prosessi samanaikaisesti

Lukee ja kirjoittaa tiedostoja suurina lohkoina (0.5-8 KB?)

- käyttäjätason prosessit käsittelevät tiedostoja tavuina, eikä niiden tarvitse tietää tiedostojen fyysistä rakennetta tai sijaintia
- pitää yllä suuria puskurialueita (jopa 50% keskusmuistista) auki oleville tiedostoille



Tiedostojärjestelmässä pitää omissa tietorakenteissaan kirjaa siitä, mitkä tiedostot milläkin prosessilla on auki, millä oikeuksilla ne on auki ja mitä kohtaa kutakin tiedostoa ollaan nyt käsittelemässä. Sama tiedosto voi olla auki usealle prosessille samanaikaisesti ja nämä prosessit voivat käyttää yhteisiä tiedostopalvelimen puskurialueita. Samanaikaisuuden hallinta on tässäkin ongelmallista. Jos joku prosesseista haluaa muuttaa tiedoston tietoja, niin tiedostojärjestelmän tulee ratkaista, sallitaanko muutos lainkaan ja milloin mahdollisesti muuttuneet tiedot näkyvät muille.

Tiedostojärjestelmä, tiedostojen hallintajärjestelmä

Osa käyttöjärjestelmää, hallitsee tiedostojen käyttöä

Valvoo oikeuksia tiedostoja avattaessa

- jotkut tiedostojärjestelmät tarkistavat käyttöoikeudet joka viitteellä

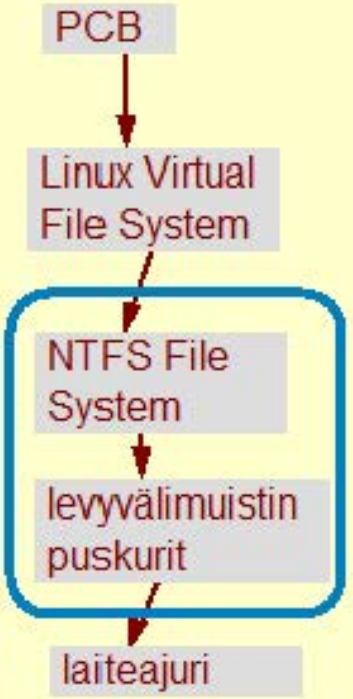
Muuntaa tiedostonimet fyysisiksi laiteosoitteiksi

Ylläpitää taulukoita, joista näkee, mitä kohtaa tiedostoa kukin prosessi on käsittelemässä

- sama tiedosto voi olla auki usealle prosessi samanaikaisesti

Lukee ja kirjoittaa tiedostoja suurina lohkoina (0.5-8 KB?)

- käyttäjätason prosessit käsittelevät tiedostoja tavuina, eikä niiden tarvitse tietää tiedostojen fyysistä rakennetta tai sijaintia
- pitää yllä suuria puskurialueita (jopa 50% keskusmuistista) auki oleville tiedostoille



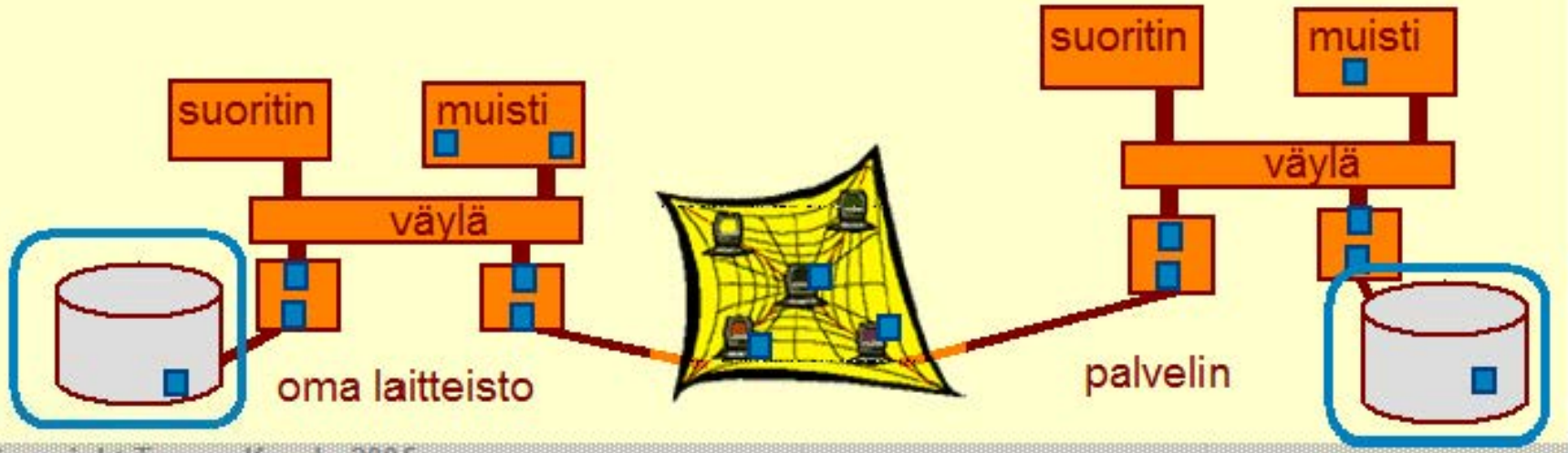
Tiedostojärjestelmä lukee tiedostoja suurina lohkoina, joiden koko on tietenkin levylohkon monikerta. Lohkot talletetaan suuriin puskurialueisiin muistissa eli levyvälimuistiin. Tällä tavoin suurin osa prosessin pyytämistä levy-I/O -operaatioista voidaan todellisuudessa toteuttaa suoraan puskurialueelta ilman, että laiteajuria vaivataan asiassa lainkaan. Sen lisäksi vielä laiteajurilla voi olla omia puskurialueitaan muistissa, jolloin myös laiteajurilta pyydetty levy-luku voidaan ehkä sekin toteuttaa ilman fyysistä levyn käsittelyä. Molemmissa tapauksissa levyn kirjoittaminen tuottaa ongelmia samanaikaisuuden hallinnan vuoksi.

Tiedostopalvelin

Verkossa tai lähiverkossa oleva palvelin

Tiedostojenhallinta hakee tiedot paikalliseen puskuriin

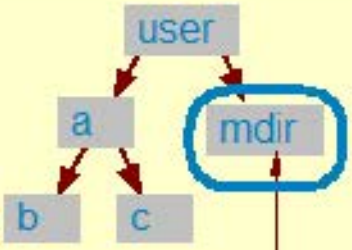
- puskuri omassa muistissa levyvälimuistissa
- etätiedostojen käyttö kuten omalla koneella olevien tiedostojen käyttö
- tiedoston tietoja puskuroitu moneen paikkaan matkan varrelle
- uuden tiedon haku aina lähimmästä puskurista, kirjoitus ongelmallista
- www-palvelimen proxy-palvelin on erikseen nimetty puskuripalvelin



Tiedostopalvelin on joko intranetissä tai muualla verkossa oleva palvelin, jolla olevia tiedostoja voi käyttöoikeuksien mukaisesti käyttää samalla tavalla kuin omalla koneella olevan levyn tiedostoja. Ohjelman ei tarvitse tietää, millä palvelimella käytettävä tiedosto sijaitsee, koska tiedostojenhallintajärjestelmä hoitaa yhteydet ja kopioinnit automaattisesti. Tiedostojen käyttö on tietenkin hitaampaa kuin paikallisella levyllä olevien tiedostojen käyttö. Parhaimmillaankin kyse on millisekuntien viipeestä, mutta se voi esimerkiksi www-palvelimien kohdalla olla sekunteja tai jopa minuutteja.

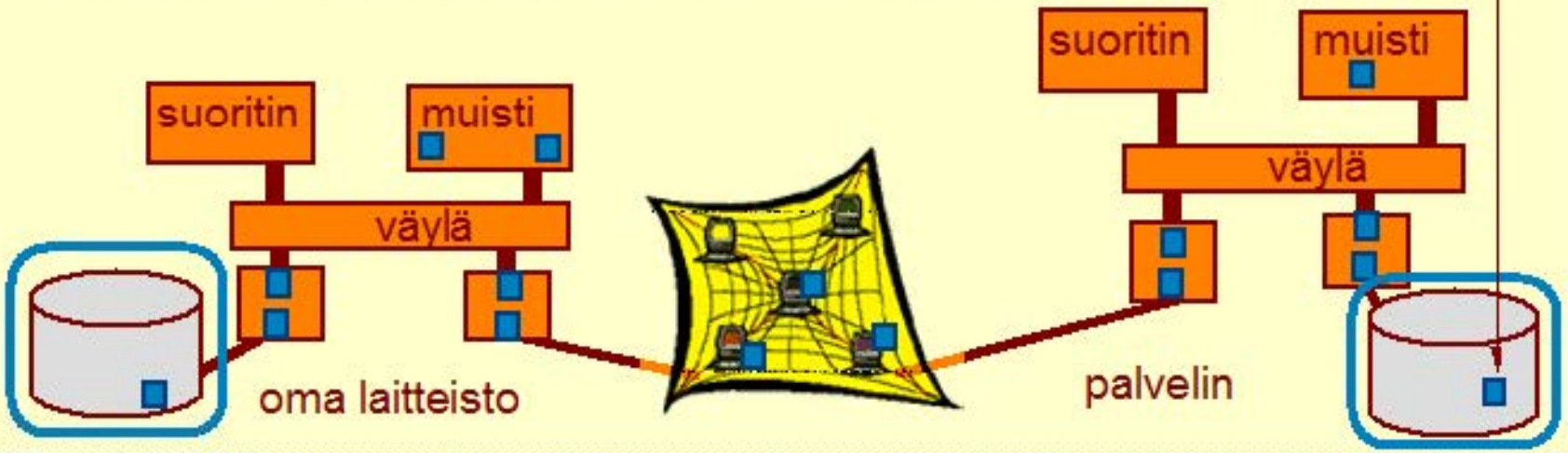
Tiedostopalvelin

Verkossa tai lähiverkossa oleva palvelin



Tiedostojenhallinta hakee tiedot paikalliseen puskuriin

- puskuri omassa muistissa levyvälimuistissa
- etätiedostojen käyttö kuten omalla koneella olevien tiedostojen käyttö
- tiedoston tietoja puskuroitu moneen paikkaan matkan varrelle
- uuden tiedon haku aina lähimmästä puskurista, kirjoitus ongelmallista
- www-palvelimen proxy-palvelin on erikseen nimetty puskuripalvelin



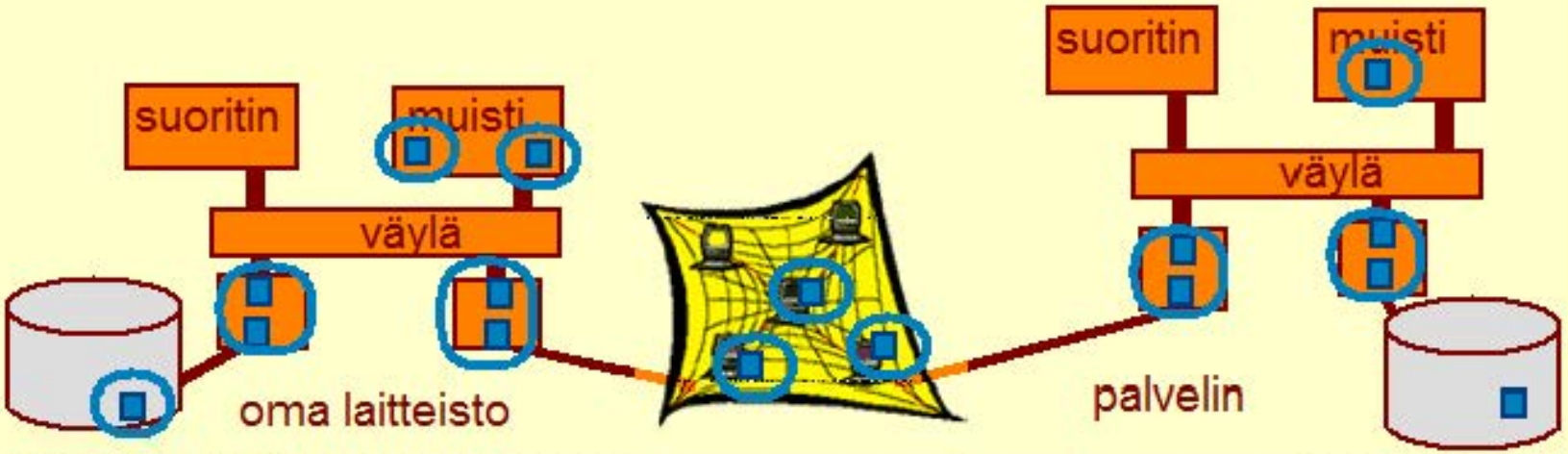
Etätiedostojen käyttö voi olla yhteydellistä, jossa palvelimella oleva hakemisto asemoidaan (mountataan) osaksi oman järjestelmän hakemistopuuta ja jonka jälkeen palvelimella olevia tiedostoja voidaan käyttää aivan paikallisten tiedostojen tapaan. Toisaalta, esimerkiksi verkkoselaimet toimivat eri tavoin. Jokainen verkkoselaus käsitellään erikseen ja tällöin haetaan vain yksi verkkosivu liitteineen omalle koneelle, jossa selain sitten esittää sen halutussa muodossa. Usea peräkkäinen selausoperaatio samalle palvelimelle voidaan yhdistää toisiinsa esimerkiksi omalle koneelle talletettavien piparien (cookies) avulla.

Tiedostopalvelin

Verkossa tai lähiverkossa oleva palvelin

Tiedostojenhallinta hakee tiedot paikalliseen puskuriin

- puskuri omassa muistissa levyvälimuistissa
- etätiedostojen käyttö kuten omalla koneella olevien tiedostojen käyttö
- tiedoston tietoja puskuroitu moneen paikkaan matkan varrelle
- uuden tiedon haku aina lähimmästä puskurista, kirjoitus ongelmallista
- www-palvelimen proxy-palvelin on erikseen nimetty puskuripalvelin



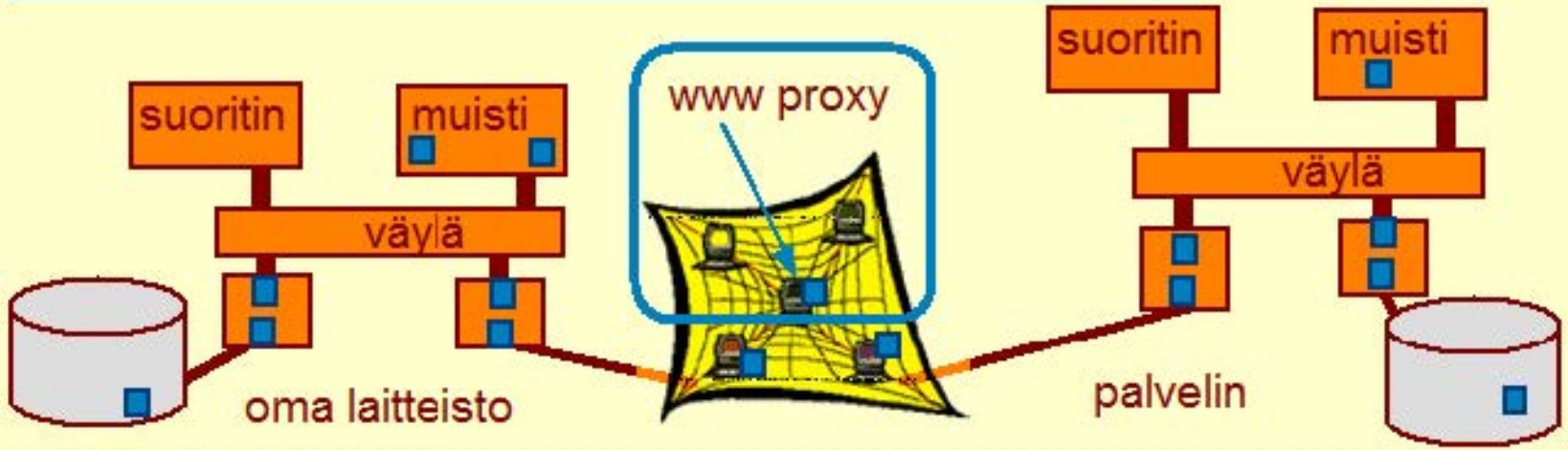
Levypalvelimien käyttöön liittyy huomattava määrä puskurointia. Etäkoneella käytettävästä tiedostosta voi olla palvelimella, verkossa ja omalla koneella yhteensä kymmeniä kopioita. Palvelimella kopioita on laiteohjaimien puskureissa ja tiedostovälimuistissa. Verkossa siitä on useita kopioita jokaisessa polun varrella olevassa koneessa. Omalla koneellakin siitä on useita kopioita eri laiteohjaimien puskureissa ja tiedostovälimuistissa. Voipa siitä olla kopio myös omalla kovalevylläkin. Puskurien avulla tiedon lukeminen usein nopeutuu, mutta kirjoittaminen tulee jälleen työlääksi. Onko varmaa, että kirjoitus menee perille asti kaikista puskureista? Milloinka eri puskurit 'vanhenevat' ja niiden sisältö päivitetään?

Tiedostopalvelin

Verkossa tai lähiverkossa oleva palvelin

Tiedostojenhallinta hakee tiedot paikalliseen puskuriin

- puskuri omassa muistissa levyvälimuistissa
- etätiedostojen käyttö kuten omalla koneella olevien tiedostojen käyttö
- tiedoston tietoja puskuroitu moneen paikkaan matkan varrelle
- uuden tiedon haku aina lähimmästä puskurista, kirjoitus ongelmallista
- www-palvelimen proxy-palvelin on erikseen nimetty puskuripalvelin



Verkkoselailua varten on kehitetty vielä erityinen proxy-palvelin, joka puskuroi kaikkea yhden organisaation verkkoliikennettä jonkin suurehkon erillisen palvelimen avulla. Kaikki selaimen viittaukset tapahtuvat nyt tuon proxy-palvelimen kautta. Jos viitattu sivu on proxy-palvelimella, niin se annetaan heti käyttöön. Jos sitä ei ole siellä, niin se haetaan muualta ja kopioidaan sitten proxy-palvelimelle sillä ajatuksella, että todennäköisesti sama tai joku muu tämän organisaation jäsen sitä tarvitsee kohta uudelleen. Esimerkiksi, tietojenkäsittelytieteen laitoksella käyttävät. Milloinka eri puskurit 'vanhenevat' ja niiden sisältö päivitetään?

Magneettiset levyt

- Kiintolevy
- 1.44 MB levyke
- ZIP-levyke
- Jaz-levyke

Kiintolevy
umpinainen, ei vaihdettava
tila: 0.15 - 1 TB
hakuaika: 5-15 ms
1-10 levyä: 1-20 levypintaa
nopeus: 4500-10800 rpm
siirtonopeus: 5-50 MB/sec

Hitachi Microdrive 3K6, 6 GB

LaCie Bigger Disk, 1 TB

Magneettisia levyjä on monenlaisia ja niiden tarkemmat speksit vaihtelevat paljon. Ne ovat kaikki myös kirjoitettavia medioita, eli yleensä niitä käsitellään luku/kirjoitusasemissa. Joissakin pelikoneissa tosin on käytetty ainoastaan lukuasemia. Tavallinen kovalevy on yleisin, mutta siinäkin kokoerot ovat huimia digikameraan tai kännykkään mahtuvasta mikrodrivesta 1 TB levykköihin. Kaikilla on huonona puolena tärähdysherkkyys ja tietenkin taltion kiinteä koko. Levykköä ei voi vaihtaa. Etuna on suuri taltion koko ja edullinen hinta muihin medioihin verrattuna. Kovalevyjä saa myös helposti siirrettävinä, mutta hitaampina USB-versioina.

Magneettiset levyt

Kiintolevy

1.44 MB levyke

ZIP-levyke

Jaz-levyke

1.44 MB levyke
vaihdettava levyke
tila: 1.44 MB
hakuaika: 90 ms
1 levy: 1 levypinta
nopeus: 300 rpm
siirtonopeus: 0.05 MB/sec

1.44 MB levykkeet eli floppyt olivat vielä muutama vuosi sitten jokaisen pöytäkoneen vakiokalustoa ja edelleenkin se löytyy useimmista koneista. Sen käyttö on vain kovin vähäistä, koska se on usein nykyään aivan liian pieni.

Magneettiset levyt

Kiintolevy

1.44 MB levyke

ZIP-levyke

Jaz-levyke

ZIP
vaihdettava kovalevy
tila: 250-750 MB
hakuaika: 30-50 ms
1 levy: 1 levypinta
nopeus: 3000 rpm
siirtonopeus: 0.5-1 MB/sec

ZIP-levyasemat ovat yhden yhtiön (Iomega) tuotemerkki, eikä yleinen taltiotyyppi!

ZIP-asemat suunniteltiin floppy-levykkeen korvaajaksi ja siihen ne aika pitkän aikaa olivatkin sopivia. Tällä hetkellä ne ovat jäämässä optisten levyjen ja muistitikkujen varjoon ja jäänevät pian unholaan floppy-asemien tavoin.

Magneettiset levyt

Kiintolevy

1.44 MB levyke

ZIP-levyke

Jaz-levyke

JAZ
vaihdettava kovalevy
tila: 1-2 GB
hakuaika: 10-15 ms
1 levy: 1 levypinta
nopeus: 3000 rpm
siirtonopeus: 5 MB/sec

JAZ-levyasemat ovat yhden yhtiön (Iomega) tuotemerkki, eikä yleinen taltiotyyppi!

JAZ-levykeasemat pyrkivät niinikään saamaan jalansijaa helposti siirrettävinä kovalevyinä, mutta nekin hävisivät markkinasodan muutamassa vuodessa usb-kovalevyille ja optisille levyille.

Optiset levyt

CD, CD-R, CD-RW

DVD, DVD-ROM,
DVD+R(W), DVD-R(W)

BD-ROM (Blu-ray Disc ROM),
HD-DVD (High Density DVD)

- read-only
- write-once
- R/W

vaihdettava optinen levy
read only, kerran tai useammin kirjoitettava
yksi pitkä spiralimainen ura ("LP-levy")

tila:	n. 650 MB
hakuaika:	90 ms
pyörimisnopeus:	200-9000 rpm
siirtonopeus:	0.1-2 MB/s

CD-romput ovat vanhin yleisessä käytössä oleva optinen media ja näillä näkymin ne tulevat olemaan käytössä vielä vuosia, vaikka ovatkin vähän pieniä kooltaan. Etuna niillä on magneettisiin medioihin verrattuna helppokäyttöisyys ja halpa hinta. CR-romppujen huonona puolena on niiden hitaus. CD-levyjä on useampaa mallia yleisessä käytössä. Useimmiten kuluttajien käytössä on joko valmiiksi poltetut tai kertakäyttöiset CD, joita voi kirjoittaa myös kotikoneiden CD-luku/kirjoitus laitteilla. On myös CD-versio, joka voidaan pyyhkiä puhtaaksi ja sitten uudelleenkirjoittaa.

Optiset levyt

CD, CD-R, CD-RW

DVD, DVD-ROM,
DVD+R(W), DVD-R(W)

BD-ROM (Blu-ray Disc ROM),
HD-DVD (High Density DVD)

- read-only
- write-once
- R/W

vaihdettava optinen levy
read only, kerran tai useammin kirjoitettava
rakenne kuten kovalevyllä

tila:	4.7-17 GB
hakuaika:	100-180 ms
pyörimisnopeus:	2000-8000 rpm
siirtonopeus:	2-8 MB/s

DVD-levyille mahtuu huomattavasti enemmän tietoa kuin CD:lle ja ne ovatkin korvaamassa CD-levyt useissa käyttötilanteissa. Esimerkiksi useimmat elokuvat mahtuvat sopivasti yhdelle DVD:lle, samoin kuin vähän paremmat tietokonepelit. DVD-levykin ovat vähän vielä pieniä, joten pitkät elokuvat tai uuden teknologian teräväpiirto-elokuvat eivät niille mahdu. DVD-levyjen yhtenä heikkoutena on myös standardoinnin epäonnistuminen, minkä seurauksena markkinoilla on kaksi kilpailevaa, mutta ei-yhteensopivaa, uudelleenkirjoitettavien DVD-levyjen standardia.

Optiset levyt

CD, CD-R, CD-RW

DVD, DVD-ROM,
DVD+R(W), DVD-R(W)

BD-ROM (Blu-ray Disc ROM),
HD-DVD (High Density DVD)

- read-only
- write-once
- R/W

seuraavan sukupolven vaihdettava optinen levy
read only, kerran tai useammin kirjoitettava?
rakenne kuten kovalevyllä

tila:	15-50 GB
hakuaika:	?
pyörimisnopeus:	?
siirtonopeus:	2-8 MB/s

Toshiba HD-DVD

Sony BR-ROM'in eri versioita

Uudet, vielä 'paremmat' pelit ja teräväpiirtoelokuvat tulevat vaatimaan jo seuraavan sukupolven optista levyä. Sille on kaksi kilpailijaa, BD-ROM ja HD-DVD. Esimerkiksi uusi Sony Playstation-3 tulee käyttämään BD-ROM levyjä. HD-DVD -levyjä voidaan valmistaa olemassaolevilla DVD-levyjen valmistuslinjoilla, joten ne tulevat olemaan halvempia. HD-DVD on DVD-levyjen valmistajien kehittäjien yhteisorganisaation, DVD Forumin hyväksymä seuraaja DVD-levyille. Luultavaa on siis, että DVD-levyistä tuttu usean standardin kilpailutilanne jatkuu myös tulevaisuudessa.

Flash-muisti levylaittena

Oikeasti toteutettu Flash-muistiteknologialla

- sisäinen rakenne Flash-muistia
- pysyväismuistia, ei tarvitse sähkövirtaa tiedon säilymiseen
- hitaampi kuin keskusmuisti
- paketoituna nopeus samaa luokkaa levyjen kanssa

Helpompi käyttää kuten magneettinen tai optinen levy

- paketoitu USB-liittymän (tms) kautta siirrettäväksi kovalevyksi
- käyttö kuten kovalevyllä
- ei voida käyttää kuten keskusmuistia
- ei tarvita omaa käyttöliittymää siirrettävälle keskusmuistille

Kilpailee siirrettävien magneettisten ja optisten levyjen kanssa

- ei ole altis tärinälle
- hyvä hinta/laatu -suhde
- tämän hetken media henkilökohtaisesti siirrettävälle tiedolle

Siirrettävät flash-muisti -teknologiaan perustuvat muistit kuuluvat yllättäen samaan kategoriaan siirrettävien magneetttisten ja optisten levyjen kanssa. Niiden todellinen sisäinen rakenne on tietenkin tavallinen flash-muisti. Flash-muisti on selvästi hitaampaa kuin keskusmuisti, mutta sen erinomaisena hyvänä puolena on tietojen säilyminen ilman sähkövirtaa. Suoraan muistiväylään liitettynä ne olisivat suht'koht nopeita, mutta yleensä nämä siirrettävät flash-muistit on paketoitu hitaampien väyläliitosten läpi, jolloin liitosten nopeus useinkin määrittelee käytännössä siirrettävien flash-muistien nopeuden.

Flash-muisti levylaittena

USB Flash muistitikku
32MB - 4GB

Oikeasti toteutettu Flash-muistiteknologialla

- sisäinen rakenne Flash-muistia
- pysyväismuistia, ei tarvitse sähkövirtaa tiedon säilymiseen
- hitaampi kuin keskusmuisti
- paketoituna nopeus samaa luokkaa levyjen kanssa

Helpompi käyttää kuten magneettinen tai optinen levy

- paketoitu USB-liittymän (tms) kautta siirrettäväksi kovalevyksi
- käyttö kuten kovalevyllä
- ei voida käyttää kuten keskusmuistia
- ei tarvita omaa käyttöliittymää siirrettävälle keskusmuistille

Kilpailee siirrettävien magneettisten ja optisten levyjen kanssa

- ei ole altis tärinälle
- hyvä hinta/laatu -suhde
- tämän hetken media henkilökohtaisesti siirrettävälle tiedolle

Siirrettävät flash-muistit paketoidaan usein USB-väylän kautta siten, että niiden laiteajurit ovat käyttöliittymältään kovalevyjen laiteajurien kaltaisia. Tiedostojärjestelmä voi nyt käyttää niitä kovalevyjen tapaan, mikä onkin hyvin kätevää. Erikoista kuitenkin on, että keskusmuistin kaltaisella teknologialla toteutettu muistialue näyttää käyttäjälle kovalevyltä! Flash-muisteja on erikseen siirrettävinä muistitikkuina ja myös erilaisille digitaalisille laitteille (esim. digikameroille) suunniteltuja pienemmän ulkomuodon omaavia muistilevyjä, joihin myös on olemassa tietokoneisiin liitettyjä USB-luku/kirjoitusasemia.

Flash-muisti levylaittena

Oikeasti toteutettu Flash-muistiteknologialla

- sisäinen rakenne Flash-muistia
- pysyväismuistia, ei tarvitse sähkövirtaa tiedon säilymiseen
- hitaampi kuin keskusmuisti
- paketoituna nopeus samaa luokkaa levyjen kanssa

Helpompi käyttää kuten magneettinen tai optinen levy

- paketoitu USB-liittymän (tms) kautta siirrettäväksi kovalevyksi
- käyttö kuten kovalevyllä
- ei voida käyttää kuten keskusmuistia
- ei tarvita omaa käyttöliittymää siirrettävälle keskusmuistille

Kilpailee siirrettävien magneettisten ja optisten levyjen kanssa

- ei ole altis tärinälle
- hyvä hinta/laatu -suhde
- tämän hetken media henkilökohtaisesti siirrettävälle tiedolle

Flash-muistit kilpailevat tehokkaasti kaikkien siirrettävien magneettisten ja optisten levyjen kanssa. Niiden etuna on tärähdysten kestäminen ja pieni koko. Flash-muistien heikkoutena optisiin levyihin verrattuna on niiden kallis hinta. Elokuvat ja tietokonepelit on edelleenkin paljon halvempi levittää halvoilla DVD-levyillä muistitikkujen tai -levyjen asemesta.

Ulkoinen muisti ja I/O:n toteutus

Muistihierarkia
Virtuaalimuisti

Kiintolevyt ja muut pyörivät levyt

I/O:n toteutus ja I/O:n tyypit
Laiteajuri ja laiteohjain

Tiedostojärjestelmä

Erilaiset levymuistit

Olemme nyt käyneet läpi tietokonejärjestelmän ulkoisen muistin ja I/O:n perusideat. Aloitimme muistihierarkiasta ja virtuaalimuistista. Kävimme sen jälkeen läpi kovalevyjen rakenteen ja toiminnan tiedon taltioimiseksi levylle. Esittelimme myös, kuinka kovalevyt ja kaikki muutkin I/O-laitteet liitetään käyttöjärjestelmään laiteajurien avulla. Näytimme, kuinka laiteajuri ja laiteohjaimella suorittava laiteohjainprosessi yhdessä toteuttavat I/O:n. Lopuksi esittelimme tiedostojärjestelmän peruspiirteet ja lyhyesti erilaisten magneettisten ja optisten levyjen peruspiirteet.