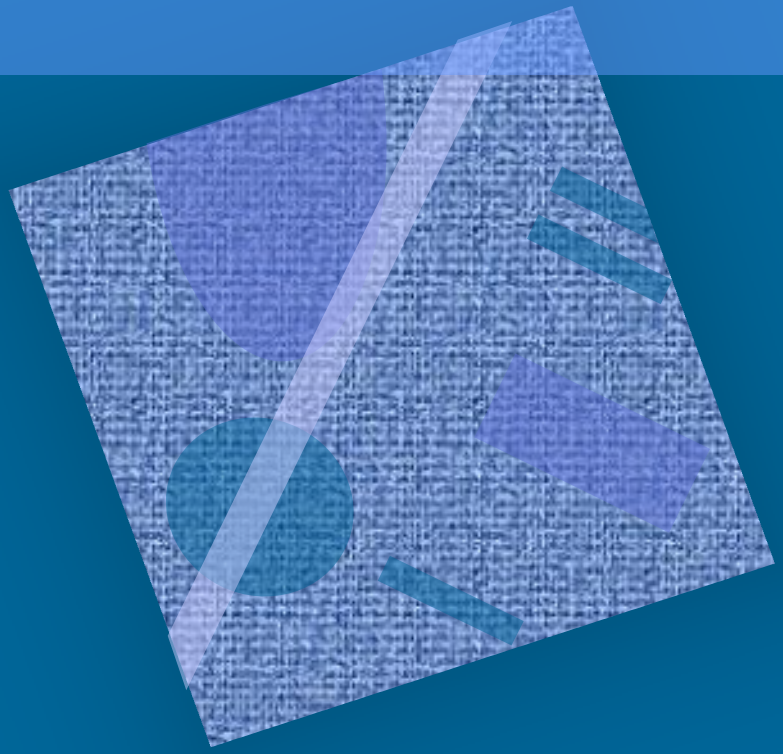
Aliohjelmien toteutus Suoritin ja väylä



Tyypit, Parametrit

Aktivointitietue (AT)

AT-pino, rekursio

Käskyjen suoritussykli

Suorittimen tilat

Poikkeukset ja keskeytykset

Aliohjelmatyypit

- Korkean tason ohjelmointikielen käsitteet
  - aliohjelma, proseduuri
    - parametrit
  - funktio
    - parametrit, paluuarvo
  - metodi
    - parametrit, ehkä paluuarvo
- Konekielitason vastaava käsite
  - aliohjelma
    - parametrit ja paluuarvo(t)

Parametrit ja paluuarvo

- Muodolliset parametrit
  - määritelty aliohjelmassa ohjelmointihetkellä
  - tietty järjestys ja tyyppi
  - paluuarvot
    - käsittely hyvin samalla tavalla kuin parametreillekin

```
Tulosta (int x, y)
void Tulosta (int x, y)

Laske(int x): int
int Laske(int x)
```

- Todelliset parametrit ja paluuarvo
  - todelliset parametrit sijoitetaan muodollisten parametrien paikalle kutsuhetkellä suoritusaikana
  - paluuarvo saadaan paluuhetkellä ja sitä käytetään kuten mitä tahansa arvoa

```
Tulosta (5, apu);
x = Laske( y+234);
```

Parametrityypit

- Arvoparametri
  - välitetään parametrin arvo (eli sen kopio) kutsuhetkellä
  - arvo voidaan lukea
  - alkuperäistä arvoa ei voi muuttaa, mutta arvon kopiota voi muuttaa
- Viiteparametri
  - välitetään parametrin osoite
  - arvo ja osoite voidaan lukea, arvoa voi muuttaa
- Nimiparametri *(ei yleensä käytetä)*
  - välitetään parametrin nimi
  - nimi (merkkijono) kuvataan arvoksi kutsuhetkellä
  - semantiikka määräytyy vasta kutsuhetkellä

```
Swap(i, k);        Swap(i, T[i]);
```

```
tmp = i;
i = T[i];
T[i] = tmp;  # ?
```

Aliohjelmien toteutuksen osat

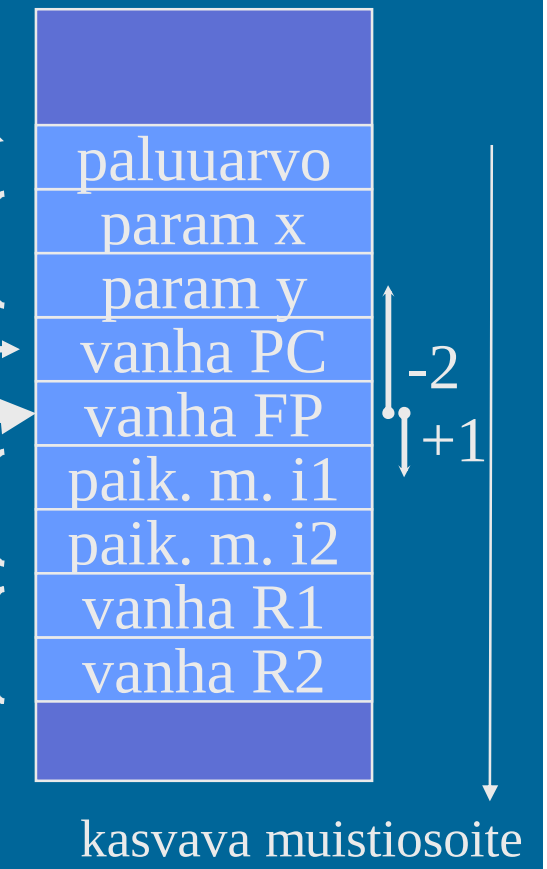
- Paluuosoite
  - kutsukohtaa seuraavan käskyn osoite
- Parametrien välitys
- Paluuarvon välitys
- Paikalliset muuttujat
- Rekisterien varaus (allokointi)
  - Kutsuva ohjelman osa haluaa säilyttää käyttämiensä rekisterien arvot!
    - pääohjelma, toinen aliohjelma, sama aliohjelma, metodi, …
  - Aliohjelman pitää aluksi tallettaa muistiin käytettävien rekisterien arvot ja lopuksi palauttaa ne ennalleen

Aktivointitietue

(activation record, activation frame)

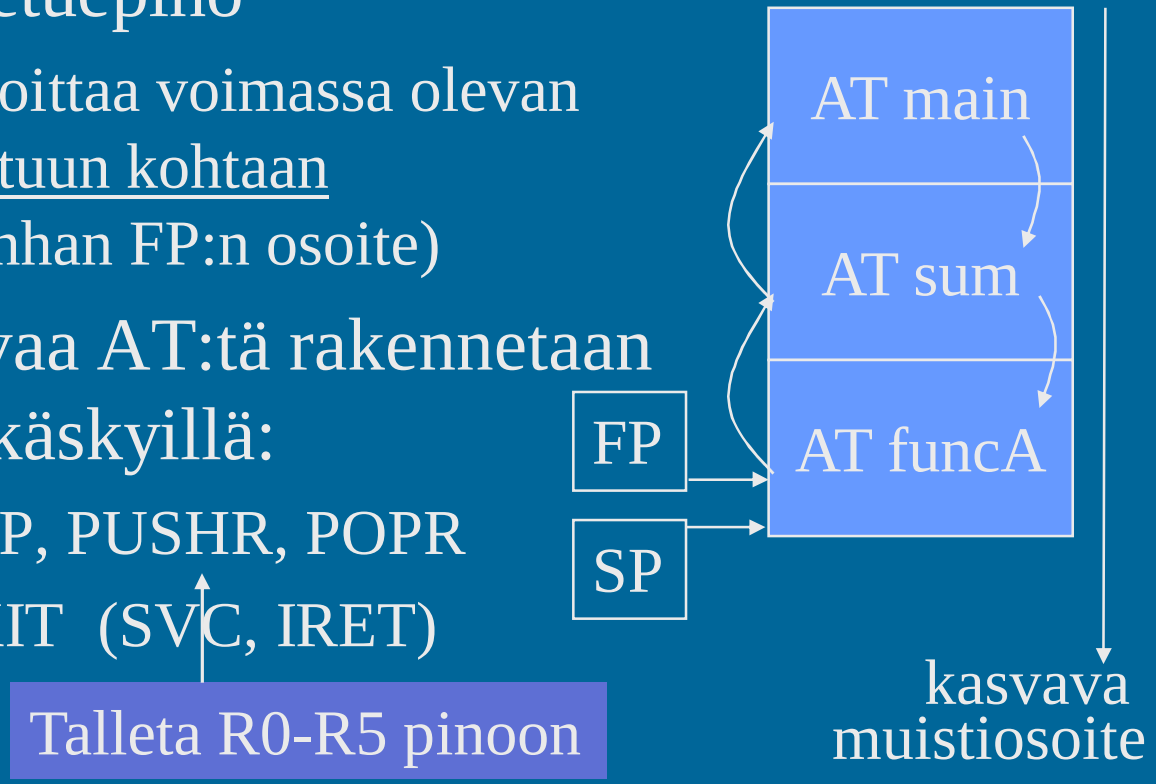
int funcA (int x,y);

- Aliohjelman toteutus (ttk-91)
  - funktion paluuarvo (tai kaikki paluuarvot)
  - kaikkien (sisäänmeno- ja ulostulo-) parametrien arvot
  - paluuosoite
  - kutsukohdan aktivointitietue
  - kaikki paikalliset muuttujat ja tietorakenteet
  - aliohjelman ajaksi talletettujen rekistereiden alkuperäiset arvot



Aktivointitietueiden hallinta

- Aktivointitietueet (AT) varataan ja vapautetaan dynaamisesti (suoritusaikana) pinosta (muistista)
  - SP (=R6) osoittaa pinon pinnalle
- Aktivointitietuepino
  - FP (R7) osoittaa voimassa olevan AT:n sovittuun kohtaan (ttk-91: vanhan FP:n osoite)
- Pinossa olevaa AT:tä rakennetaan ja puretaan käskyillä:
  - PUSH, POP, PUSHR, POPR
  - CALL, EXIT (SVC, IRET)



Aliohjelmakutsun toteutus

- Toteutus jaettu eri yksiköille
  - varaa tilaa paluuarvolle pinosta (Kutsuva rutiini)
  - laita parametrit (arvot tai osoitteet) pinoon (Kutsuva rutiini)
  - talleta vanha PC ja FP, aseta uudet PC ja FP (CALL käsky)
  - varaa tilaa paikallisille muuttujille (Kutsuttu rutiini, prolog)
  - talleta käytettävien rekistereiden vanhat arvot pinoon (Kutsuttu rutiini, prolog)
  - (itse aliohjelman toteutus – varsinainen työ)
  - palauta rekistereiden arvot (Kutsuttu rutiini, epilog)
  - vapauta paikallisten muuttujien tila (Kutsuttu rutiini, epilog)
  - palauta PC ja FP (EXIT käsky)
  - vapauta parametrien tila (Kutsuttu rutiini, epilog)
  - ota paluuarvo pinosta (Kutsuva rutiini)

Aliohjelmaesimerkki

```
int fA (int x, y)
{
        int z = 5;

        z = x * z + y;
        return (z);
}

…

T = fA (200, R);
```

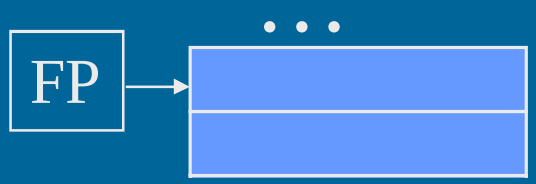
käyttö:

```
R       DC  24
…
PUSH  SP,=0 ; tila paluuarvolle
PUSH SP, =200
PUSH  SP, R

CALL  SP, fA

POP     SP, R1
STORE R1, T
…
```

muistista muistiin!!

talleta PC, FP
aseta PC, kutsu & paluu
palauta FP, PC

2. operandi aina rekisteri



tämän-hetkinen, nykyinen FP

Aliohjelmaesimerkki

(ei animointia)

```
int fA (int x, y)
{
        int z = 5;

        z = x * z + y;
        return (z);
}

…

T = fA (200, R);
```

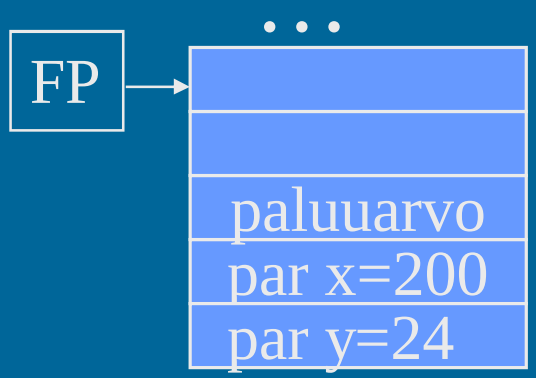
käyttö:

```
R       DC  24
…
PUSH  SP,=0 ; ret. value space
PUSH SP, =200
PUSH  SP, R

CALL  SP, fA

POP     SP, R1
STORE R1, T
…
```

muistista muistiin!!

talleta PC, FP
aseta PC,
kutsu & paluu
palauta FP, PC

2. operandi aina rekisteri



tämän-hetkinen, nykyinen FP

Aliohjelma-esimerkki

```
int fA (int x, y)
{
        int z = 5;

        z = x * z + y;
        return (z);
}

…

T = fA (200, R);
```

aliohjelman toteutus:

```
retfA   EQU  -4  # params
parX    EQU  -3
parY    EQU  -2
locZ    EQU  1   # local vars

fA      PUSH  SP, =0 ; alloc Z
        PUSH  SP, R1 ; save R1

        LOAD  R1,=5; init Z
        STORE R1, locZ (FP)

        LOAD R1, parX (FP)
        MUL   R1, locZ (FP)
        ADD   R1, parY (FP)
        STORE R1, locZ (FP)
        STORE R1, retfA (FP)
        POP    SP, R1; recover R1
        SUB    SP, =1 ; free Z
        EXIT   SP, =2 ; 2 param.
```

prolog

epilog

Kaikki viitteet näihin tehdään suhteessa FP:hen

paluuarvo

Aliohjelma-esimerkki

(ei animointia)

```
int fA (int x, y)
{
        int z = 5;

        z = x * z + y;
        return (z);
}

…

T = fA (200, R);
```

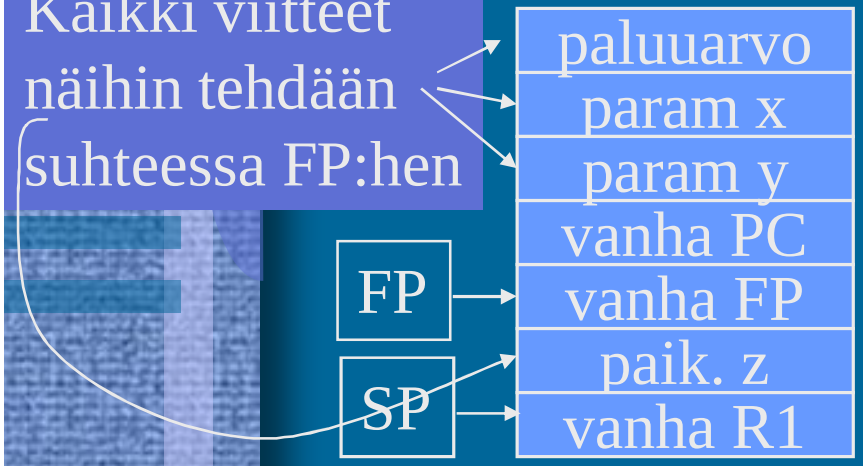
aliohjelman toteutus:

```
retfA   EQU  -4
parX    EQU  -3
parY    EQU  -2
locZ    EQU  1

fA      PUSH  SP, =0 ; alloc Z
        PUSH  SP, R1 ; save R1

        LOAD  R1,=5; init Z
        STORE R1, locZ (FP)

        LOAD R1, parX (FP)
        MUL   R1, locZ (FP)
        ADD   R1, parY (FP)
        STORE R1, locZ (FP)
        STORE R1, retfA (FP)
        POP    SP, R1; recover R1
        SUB    SP, =1 ; free Z
        EXIT   SP, =2 ; 2 param.
```

prolog

epilog

Kaikki viitteet näihin tehdään suhteessa FP:hen



Porinaa

- Mieti ja kirjaa ttk-91 käskyt, joilla voisit
  - Kutsua taulukon alkioiden keskiarvon laskevaa funktiota , esim: tulos = ka(tauluos, koko)
  - Käyttää parametreja funktiossa
  - Välittää tuloksen funktiosta kutsujalle
- Miksi KJ:n kutsurutiinista palataan IRET eikä EXIT käskyllä?
- Aliohjelmakutsun toteutukseen kuluu aika monta konekäskyä. Miten sen voisi tehdä nopeammaksi suorittaa?

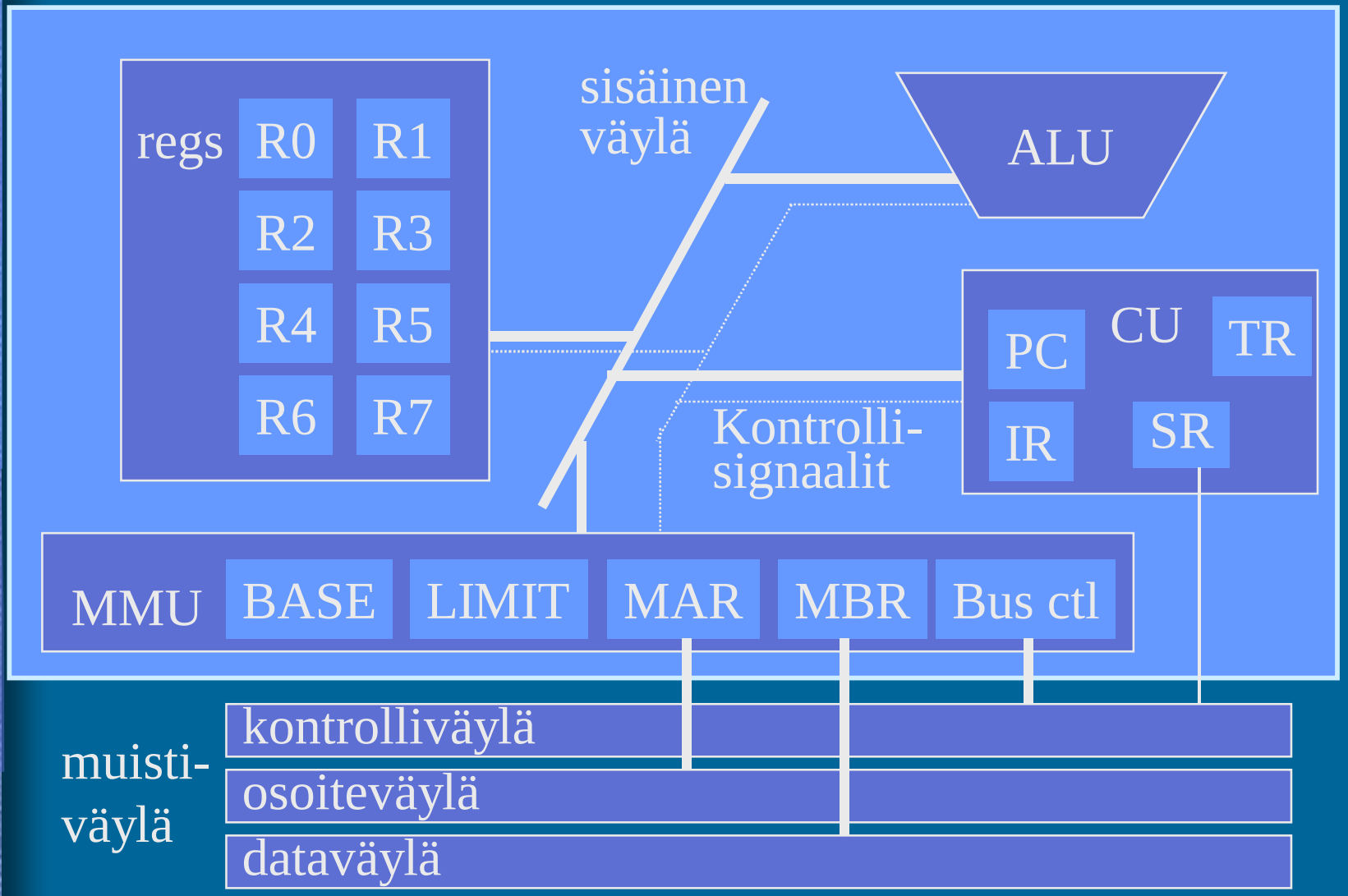
KJ-palvelun kutsu (proseduraalisesti)

- Samalla tavalla kuin aliohjelman kutsu
  - CALL käskyn asemesta SVC
- Tila paluuarvolle?
- Parametrit pinoon vai rekistereissä?
- SVC kutsu
  - Kutsuttavan rutiinin numero operandina
- IRET paluu
- Paluuarvo (OK, virhe) pois pinosta tarkistusta varten

fOK = ReadBlock (fp, 64)

```
...
PUSH  SP, =0 ;paluuarvo
PUSH  SP, =FileBuffer
PUSH  SP, CharCnt
PUSH  SP, FilePtr

SVC    SP, =ReadFile

POP      SP, R1
JNZER  R1, FileTrouble
…
```

TTK-91 suorittimen rakenne



Käskyn suorittaminen

- Titokone näyttää vaiheet simuloimalla
- Store R4, @10(R1) ol. R1 =20, R4=15
  - Käskyn nouto ja ohjelmalaskurin kasvatus:
    - PC->MAR, PC+1 -> PC, mem -> MBR, MBR -> IR
  - Käskyn suoritus:
    - R1+10->MAR, mem->MBR,
      MBR->MAR, R4->MBR, MBR->mem

TTK-91 konekäskyn rakenne

- Käskyn esitys bittitasolla on **aina**:

(pienehkö) muisti-osoite tai vakio

Rj = käskyn ensimmäinen operandi

Ri = indeksirekisteri (0 ≡ ei käytetä)

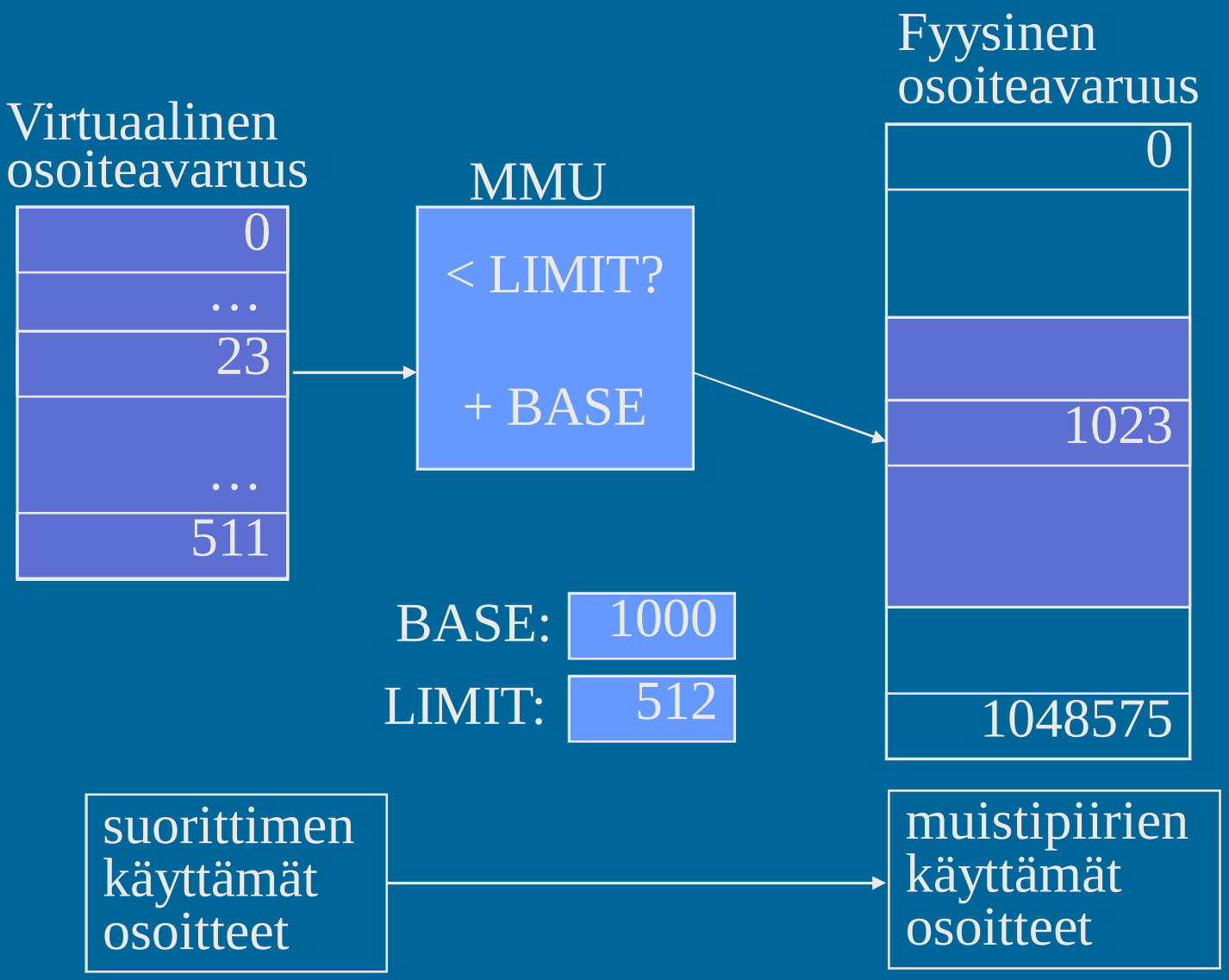
M = muistinoutojen määrä toiseen operandiin (addressing mode)
(ennen mahdollista muistiin talletusta)

00 eli 0 kpl, välitön osoitus (STORE: suora osoitus)
01 eli 1 kpl, suora osoitus (STORE: epäsuora osoit.)
10 eli 2 kpl, epäsuora osoitus (STORE: epäkelpo arvo)
(11 eli 3 kpl, epäkelpo arvo → poikkeustilanne)

Käskyn purku ja tehollisen muisti-osoitteen (EA) laskemisvaihe

- Purku automaattisesti langoitettuna käskyrekisteristä (IR)
- Muistiosoitteen lasku, tulos tilapäisrekisteriin (TR)
  - jos Ri=0, niin TR ← ADDR
  - muutoin TR ← (Ri)+ADDR
    - ALU suorittaa laskutoimituksen
    - jos ADDR = 0, niin TR ← (Ri)
  - Effective Address (EA) on nyt TR:ssä

TTK-91 virtuaalimuisti



Virtuaalimuistin osoitteenmuunnosmenetelmiä

- Kanta- ja rajarekisteriin perustuva
  - base ja limit rekisterit (esim. ttk-91, 8086, ...)
- Sivuttava
  - sivutaulut
  - virtuaaliavaruus jaettu saman kokoisiin sivuihin
- Segmentoiva
  - virtuaaliavaruus jaettu ohjelman mukaan erillisiin eri kokoisiin segmentteihin
    - koodi segmentti, data segmentti, …

Lisää tietoa?  käyttö-järjestelmä kurssi

Lisää tietoa?  käyttö-järjestelmä kurssi

Keskeytysten käsittely

- Jokainen mahdollinen keskeytystyyppi on ennalta tunnettu, eli mitään todella yllättävää ei tapahdu
- Jokaiselle keskeytystyypille on oma käyttöjärjestelmän tuntema keskeytyskäsittelyrutiini

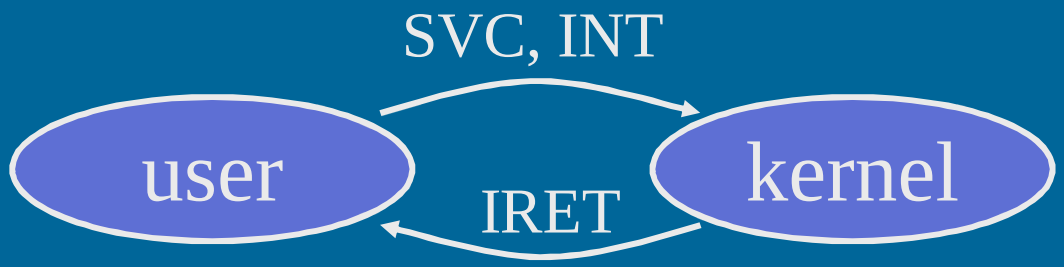
interrupt handler

- Jokaisen käskyn suorituksen jälkeen tarkistetaan keskeytysten olemassaolo SR:stä ja haaraudutaan keskeytyskäsittelijään tarvittaessa
  - joskus keskeytykset on estetty (ttk-91:ssä SR:n bitti D)
  - paluu käsittelijästä ”return-from-interrupt-handler” käskyllä (esim. IRET, tms)
- ”Yllättävä aliohjelmakutsu”

Keskeytyskäsittelijä

- Osa käyttöjärjestelmää
- Ennen keskeytyskäsittelijään hyppäämistä asetetaan suoritin ja MMU etuoikeutettuun käyttöjärjestelmätilaan (supervisor state)
  - SR:n bitti P on päällä => etuoikeutettu tila eli käyttöjärjestelmä tila
    (P = Priviledged)
  - käyttöjärjestelmätilassa saa viitata mihin tahansa kohtaan muistia (MMU: BASE=0, LIMIT="hyvin iso")
  - käyttöjärjestelmätilassa saa käyttää kaikkia konekäskyjä (esim. IRET tai ClearCache)
- Käsittelijästä paluun yhteydessä MMU:n tila ja suorittimen tila asetetaan ennalleen

Suorittimen tilan muuttaminen

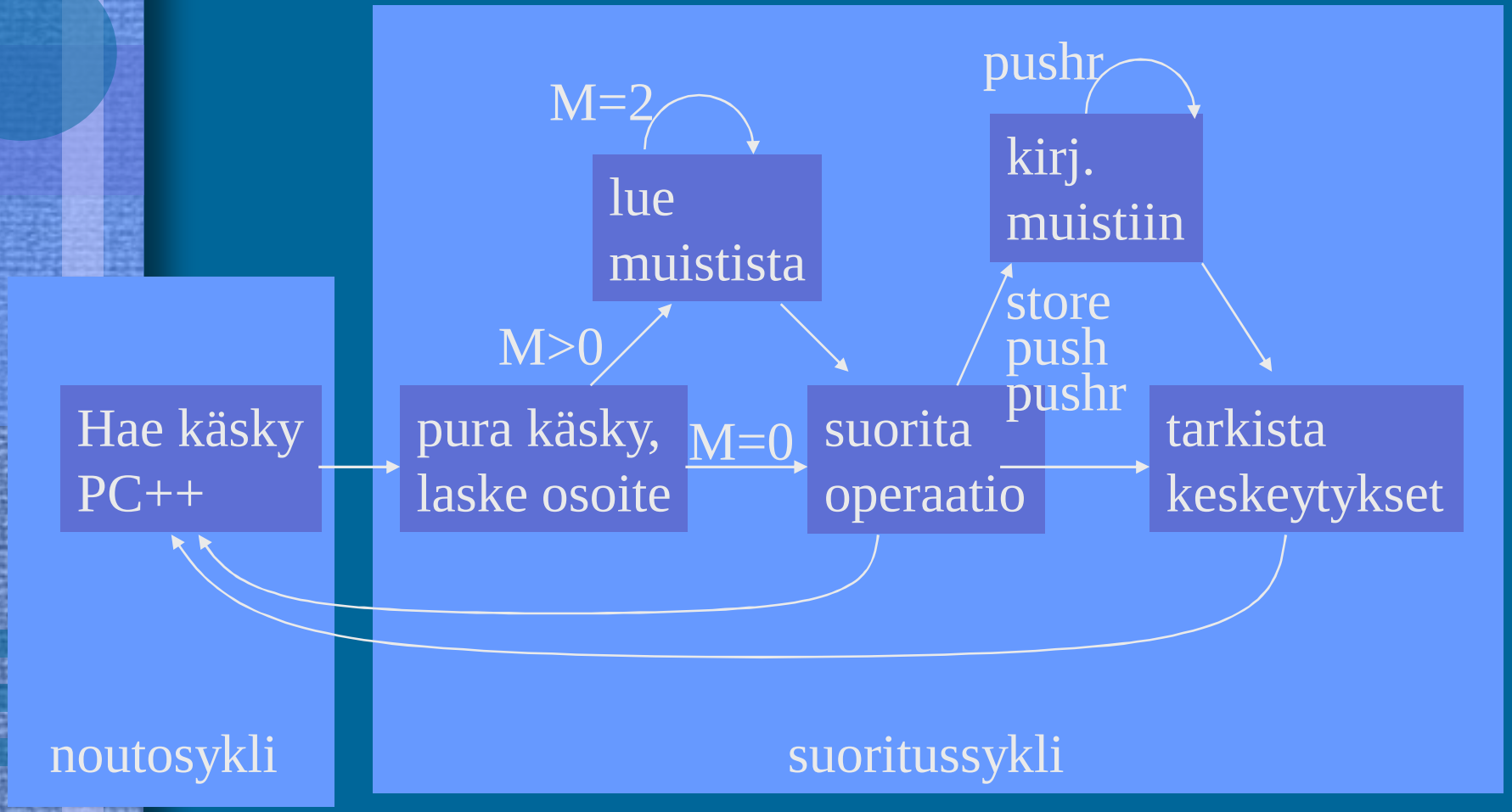


- Käyttäjätila → etuoikeutettu tila
  - keskeytys tai suora KJ:n palvelupyyntö (SVC käsky)
  - keskeytyskäsittelijä tarkistaa onko (oliko) oikeutta tilan vaihtoon

(interrupt handler)

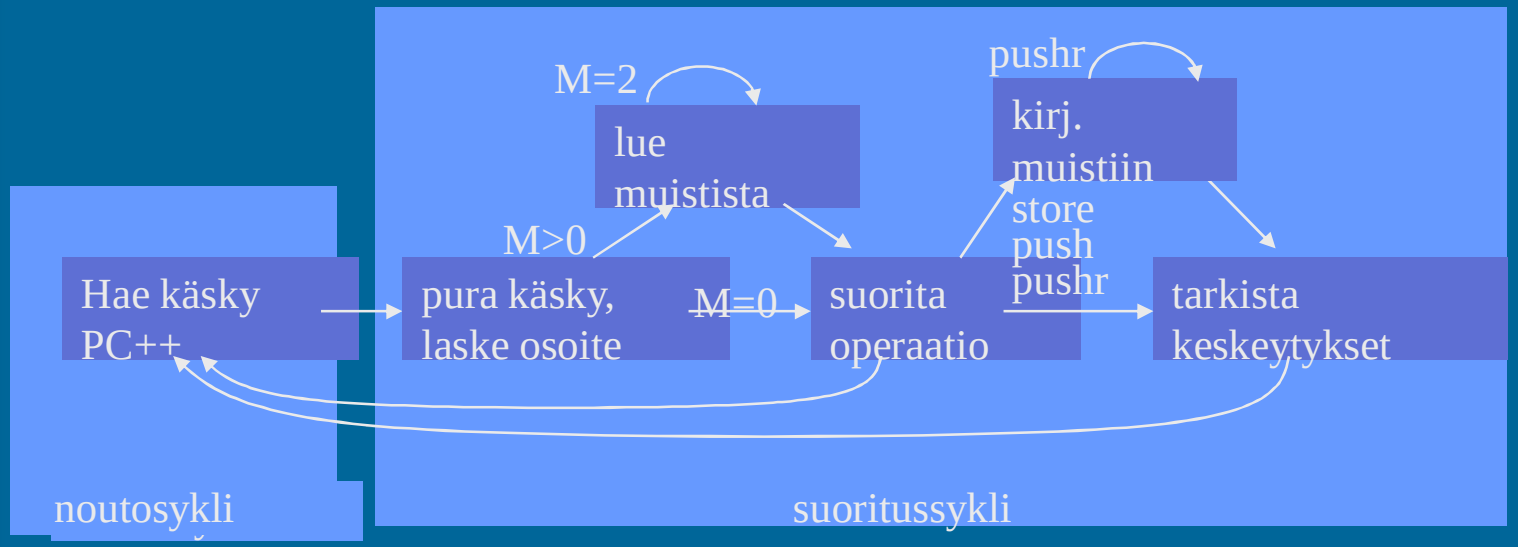
- Etuoikeutettu tila → käyttäjätila
  - etuoikeutettu konekäsky “return from interrupt handler” esim. IRET (Pentium II)
  - palauttaa kontrollin keskeytyneeseen kohtaan ja suorittimen tilan keskeytystä edeltäneeseen tilaan

TTK-91 Nouto- ja suoritussykli vielä vähän tarkemmin



Porinaa

- Miten laitteisto tietää, onko kyseessä suora muistiosoitus tai epäsuora muistiosoitus?
- Voisiko seuraavaksi suoritettavaa käskyä hakea muistista samaan aikaan kun edellistä vasta suoritetaan?
- Mitä kaikkea keskeytyskäsittelijä tekee, jos tulee kokonaislukujen nollalla jako –keskeytys?



TTK-91 koneen simulaattori

- Tavallinen Javalla kirjoitettu ohjelma
- TTK-91 koneen osat tietorakenteina
  - rekisterit, MMU, CU, muisti
- Simuloi käskyjen suoritussykliä käsky kerrallaan
- Toteuttaa myös TTK-91 koneen käyttöjärjestelmän osat osana tavallista ohjelmaa
  - assembler kääntäjä, lataaja, debugger, kesk. käsittelijät
- Graafinen käyttöliittymä

Ks. Processor.java Titokoneen koodissa

titokone.jar\fi\hu\cs\titokone\Processor.java

TTK-91 käskyn suoritussykli

hae käsky simuloidusta muistista

IR = mem[PC]

pura käsky osiin (OPER, Rj, M, Ri, ADDR) ja
laske osoiteosan arvo TR (ADDR tai regs[Ri]+ADDR)

ADDR = IR mod 32768

TR = regs[Ri] +ADDR

tee tarvittava määrä (M) operandin
hakuja muistista rekisteriin TR

TR = mem[TR]

valitse aliohjelma operaatiokoodin (OPER) perusteella

if (opcodeOK[OPER] = FALSE) then SR.U = 1;

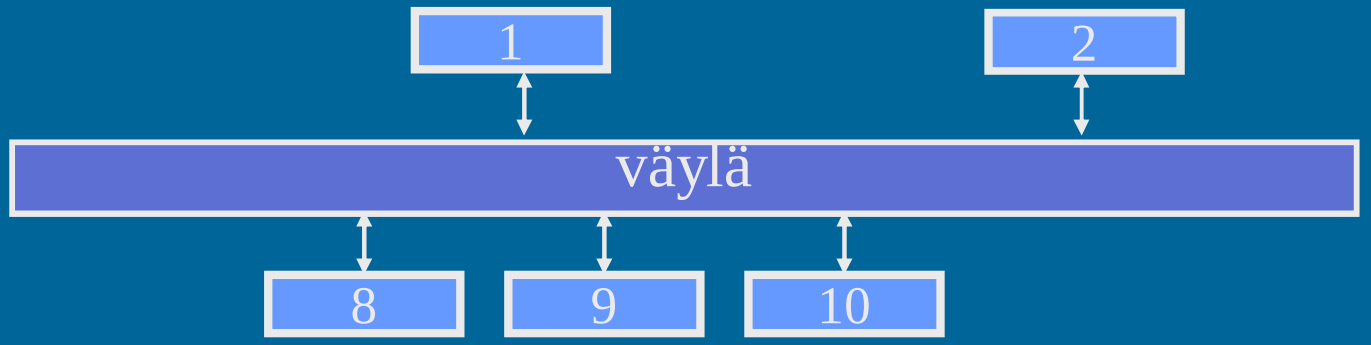
simuloi konekäskyn suorituksen muutokset
rekistereihin (R0…R7, SR, PC, MAR, MBR)

ADD Rj, M ADDR(Ri) ⇒ regs[Rj] += TR;

lopeta suoritus jos SVC tai keskeytys

SR.O = …

Väylät

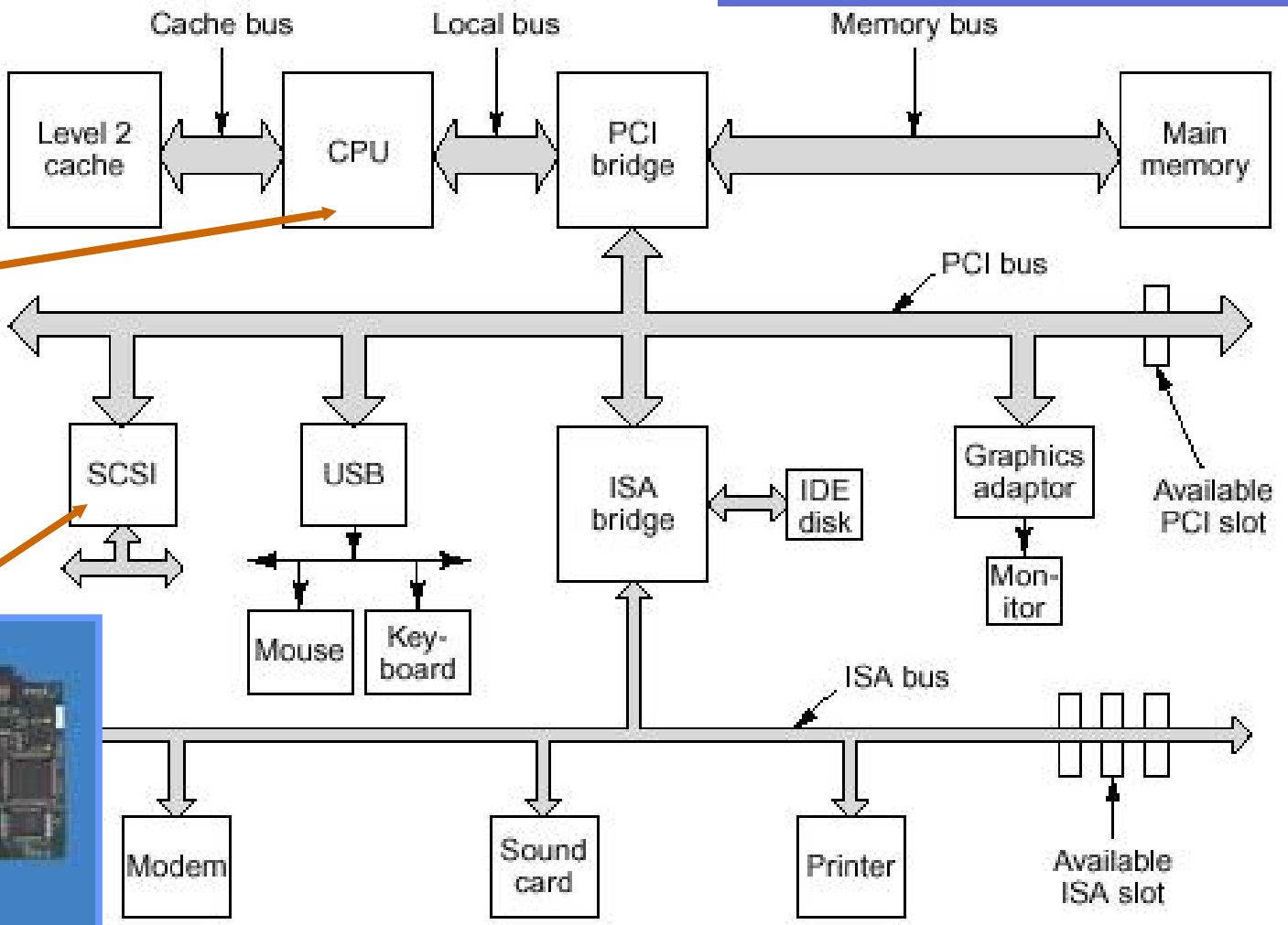


- Kullakin laitteella oma osoite
- Yksi lähettää, kaikki kuulevat, vain ”oikea” laite vastaanottaa
- Paljon erilaisia
- Lähellä suoritinta olevat ovat nopeampia

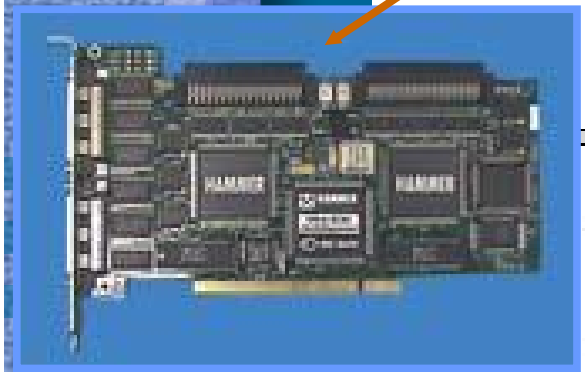
Lisää tietoa? → Tietokoneen rakenne -kurssi

Väylähierarkia

Tyypillinen Pentium II systeemin emolevy



omalla lastulla, tason 1 välimuistin kanssa



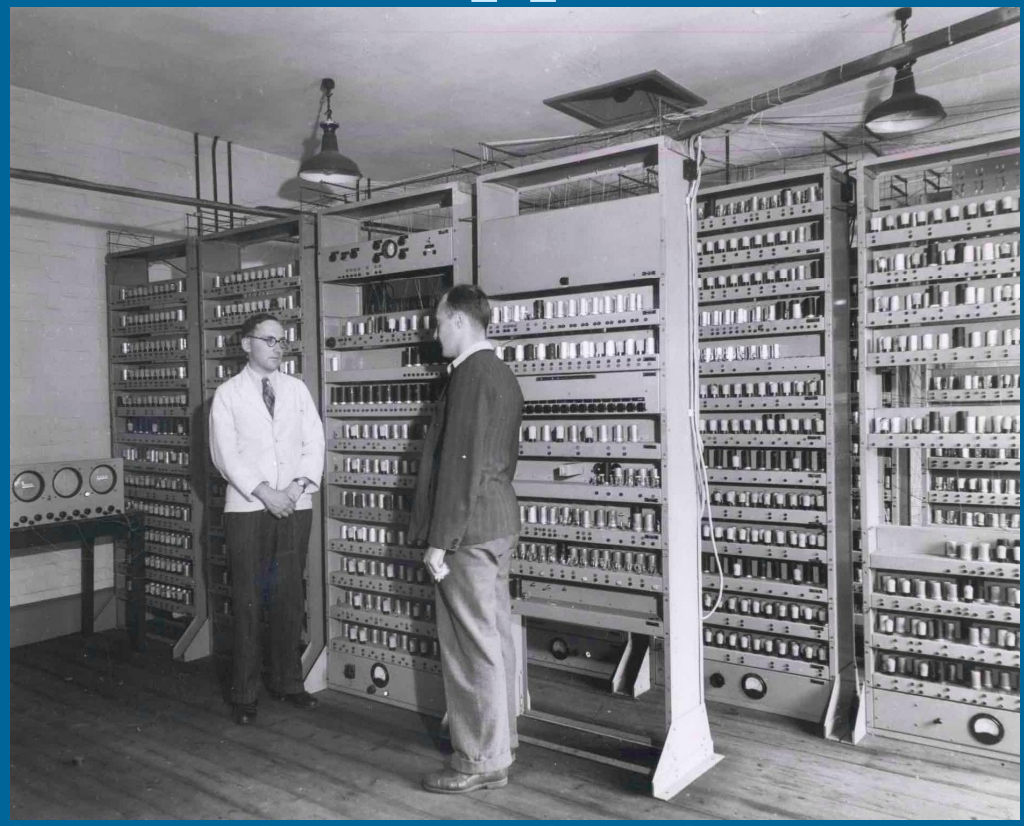
PCI to SCSI bridge

Fig. 3-50 [Tane99]

-- Luennon 3 loppu --

M.Wilkes:
EDSAC I (1949)

- rekisterit (6W), tyhjiöputkilla
- käsky- ja datamuisti, 32 elohopea-viiveputkea, 512W à 36b
- kertolasku 5.4ms, 650 IPS



http://www.cl.cam.ac.uk/Relics/archive_photos.html

- ensimmäinen ”stored program” –tietokone
- 3000 tyhjiöputkea, sähkökulutus 12 kW, tila 5x4m