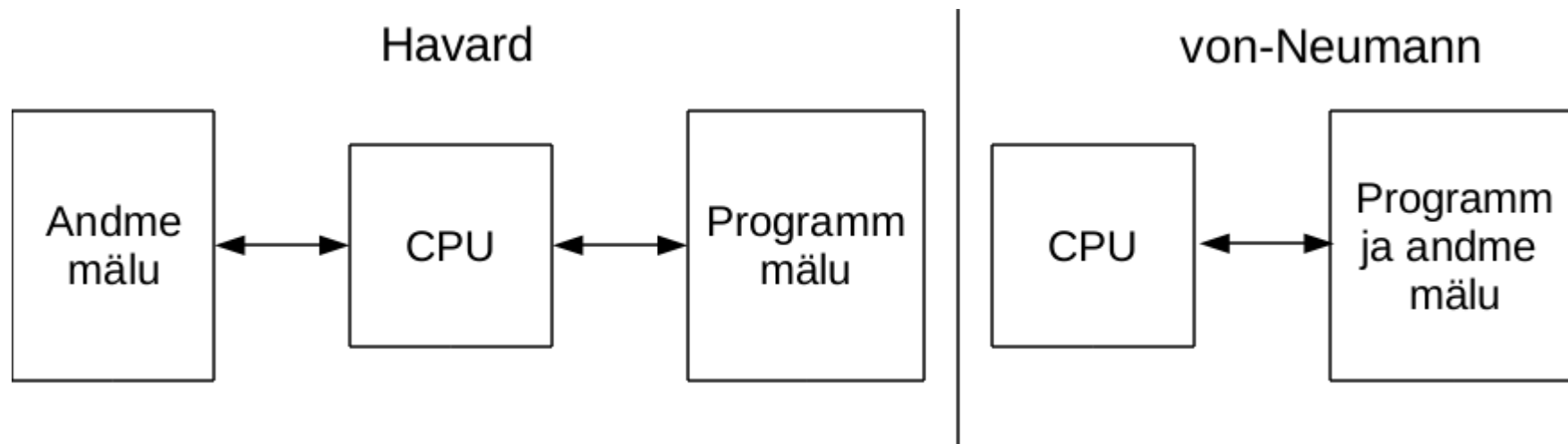


Mikrokontrollerite riistvara

Erkki Moorits

Cybernetica AS, Navigatsioonisüsteemide osakond

Kaks erinevat arhitektuuri



Harvardi ja von-Neumanni erinevused

- ▶ Harvardi arhitektuuriga mikrokontrollerid võivad olla mõningatel juhtudel kiiremad kui von-Neumanni omad
- ▶ GCC ja C teegid on algselt tehtud von-Neumanni arhitektuuri jaoks ja seetõttu käsitletakse mõndasid Harvardi arhitektuuriga seonduvaid asju teisiti
 - Näiteks Harvardi arhitektuuriga mikrokontrolleritel on olemas käsud nagu `printf_P` või programm-mälust lugemise käsud (`pgm_read_byte`)
- ▶ Von-Neumanni arhitektuurile on (natukene) lihtsam programmi kirjutada

Tüüpilised mikrokontrollerite lisad

- ▶ Riistvaralised loendurid ja taimerid
 - Tavaliselt suudavad loendurid ka PWM modulatsiooni genereerimisega toime tulla
- ▶ Seriaal liidesed
 - U(S)ART, SPI, TWI/I2C
- ▶ AD muundid ja/või analoog komparaatorid
 - Enamustel juhtudel on 10 bitine AD muundi
 - Tavaliselt võimaldavad kuni 200 ksp/s (ühe kanali peale)
- ▶ JTAG või mõni muu debugimise liides

Mitte nii tavalised mikrokontrollerite lisad

▶▶ DMA

- Võimaldab tunduvalt lihtsamalt analoog mõõtmisi teha

▶▶ USB

▶▶ CAN ja/või LIN

▶▶ Integreeritud traadita ühenduse võimalus

- Tavaliselt on toetatud ZigBee kuid on näha olnud ka teisi protokolle

▶▶ DA muundi

- Sobilik mootorite juhtimiseks

▶▶ AES ja DES krüpto lisad

Põhilised mikrokontrollerite tüübid

▶▶ 8 bitised

- Atmel'i poolt toodetud 8 ja 8/16 bitised AVR mikrokontrollerid
- Microchip'i poolt toodetud 8 bitised PIC mikrokontrollerid

▶▶ 16 bitised

- Texas Instruments'i MSP430 tootepere

▶▶ 32 bitised

- Atmeli AVR32 tootepere
- Erinevad ARM mikrokontrollerid

AVR mikrokontrollerid I

- ▶ Loodud 1996 Atmeli poolt
- ▶ Harvardi arhitektuuriga
- ▶ Mikrokontrolleri loomisel on silmas peetud C kompilaatorite eripära
 - Sobib päris hästi mikrokontrolleritel kasutatava C keele õppimiseks
- ▶ Sobib väga paljutele rakendustele, kuid:
 - Ei ole just kõige sobilikum esmaseks assembleri õppimiseks
 - Analooz mõõtmistega tegelemine on keerukas

AVR mikrokontrollerid II

- ▶ Kõik vajalik tarkvara on tasuta saadaval internetis:
 - GCC pordist on C, C++ ja Ada kompilaator ning GNU linkur olemas
 - Programmatorite PC tarkvarast on Atmelil mitmeid erinevaid programme AVRStudio'ga kaasas ja vabavarana on Avrdude
 - Debuggeritest ja simulaatoritest on olemas AVRStudioga kaasasolev debugger aga on olemas ka GDB AVR'i port ja AVR'i vabavaraline simulaator
- ▶ Väga palju erinevaid programmatoreid ja katseplaate

AVR mikrokontrollerid III

▶▶ TinyAVR

- 6–32-väljaviiguga korpuses, 0.5 kB – 8 kB flashmälu
- Suhteliselt vähe integreeritud seadmeid

▶▶ MegaAVR

- 28–100-väljaviiguga korpuses, 4 kB – 256 kB flashmälu
- Korrutamise käsud ja suhteliselt palju erinevaid integreeritud lisaseadmeid

▶▶ XMEGA

- 44–100-väljaviiguga korpuses, 16 kB – 384 kB flashmälu
- Korrutamise käsud, DMA, "Event System", krüpto funktsioonid, DAC ja palju muid integreeritud lisaseadmeid

PIC mikrokontrollerid I

- ▶ Harvardi arhitektuuriga
- ▶ Populaarsed hobiprojektides
- ▶ Väga palju tasuta arendustarkvara
 - Kuid puudub GCC kompilaator
 - Peamiselt on arenduse tarkvara Windowsi all
- ▶ Väga palju erinevaid programmeid
- ▶ Peamiseks programmeerimise keeleks on assembler
 - Sobib väga hästi assembleri algõppeks
- ▶ Suhteliselt keerukas kasutada suuremates ja keerukamates projektides
 - Katkestustele vastamine on küllaltki keerukas

PIC mikrokontrollerid II

▶▶ PIC10

- Minimaalselt lisaseadmeid
- Väga väike voolutarve
- 6-8 väljaviiguga korpus

▶▶ PIC12

- Suhteliselt vähe lisaseadmeid
- 8-14 väljaviiguga korpus

▶▶ PIC16

- Piisavalt erinevaid lisaseadmeid
- 14-44 väljaviiguga korpus
- Kasutatakse üpriski palju hobiprojektides

PIC mikrokontrollerid III

▶▶ PIC18

- Palju erinevat lisaseadmeid
- 18-100 väljaviiguga korpus
- Erinevalt teistest 8 bitistest PIC seeria mikrokontrolleritest on peamiseks arenduse keeleks C

MSP430 mikrokontrollerid

- ▶▶ Von-Neumanni arhitektuuriga
- ▶▶ Arendatud 90'ndate alguses Texas Instruments'i poolt
- ▶▶ Väikse energiatarbega
 - Algselt olid mõeldud vee- ja energiamõõtjatele
- ▶▶ Väga lihtsalt kasutatav analoog muundi
- ▶▶ Olemas TI enda poolt pakutav arendus keskkond ja samaväärne GNU arenduskeskkond
- ▶▶ Ei ole kõige parem valik mikrokontrollerite õppimiseks
- ▶▶ Suhteliselt vähe vabavara programmeid

AVR32 ja PIC32 mikrokontrollerid

- ▶ ARM mikrokontrolleri analoogid Atmel'i ja Microchip'i poolt
- ▶ Mõlemad on Harvardi arhitektuuriga
- ▶ Mõlematel on GCC tugi olemas
 - Microchip'il on mitteametlik GCC tugi
- ▶ Väga palju erinevaid integreeritud seadmeid
 - Kuigi neil on integreeritud AD muundi ei pruugi see olla kõige täpsem
- ▶ Suhteliselt keerukad ning algtaseme õppe jaoks mitte eriti sobivad

ARM mikrokontrollerid

- ▶ Algselt loodud x86 asenduseks, kuid lihtsa ehituse ja väikse voolutarbe tõttu pigem sobivad sardsüsteemidesse
- ▶ ARM7 on von-Neumanni arhitektuuriga, ARM9 on Harvardi arhitektuuriga
- ▶ Suuremad kontrollerid ei ole eriti sobivad täpseteks analoogmõõtmisteks
- ▶ Väga lai arendustööriistaade valik
 - On olemas GCC ja GNU linkerid
- ▶ Ei ole kõige sobilikum algajatele

Kernelid

Teemad

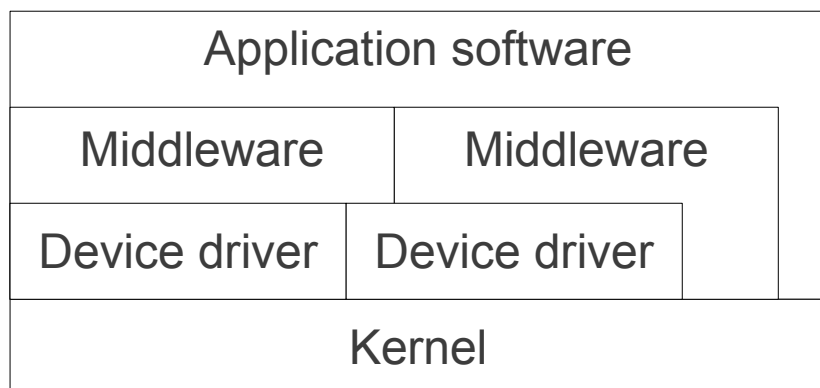
- ▶▶ „Tavalised“ ja real-time kernelid
- ▶▶ Draiverid
- ▶▶ Energia säästmine kontrollerial

Sard OS'i ülesehitus

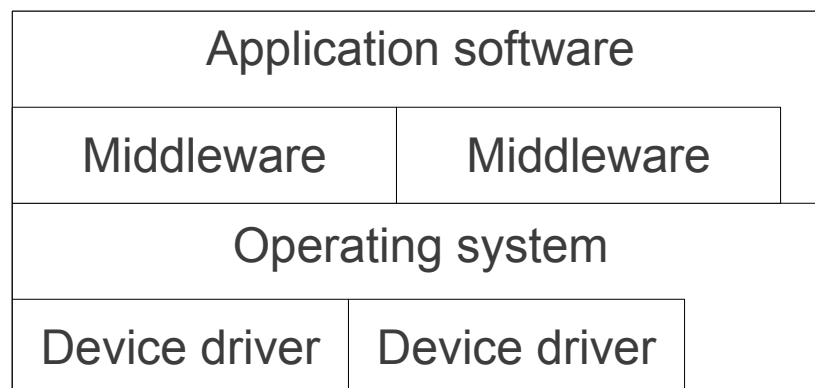
▶▶ Mitu lähenemisviisi

- Kernel koos draiveritega
- Kernel ilma draiveriteta – kasutaja peab ise draiverid kirjutama
- „Tavaline“ kernel RT kerneli peal

Embedded OS

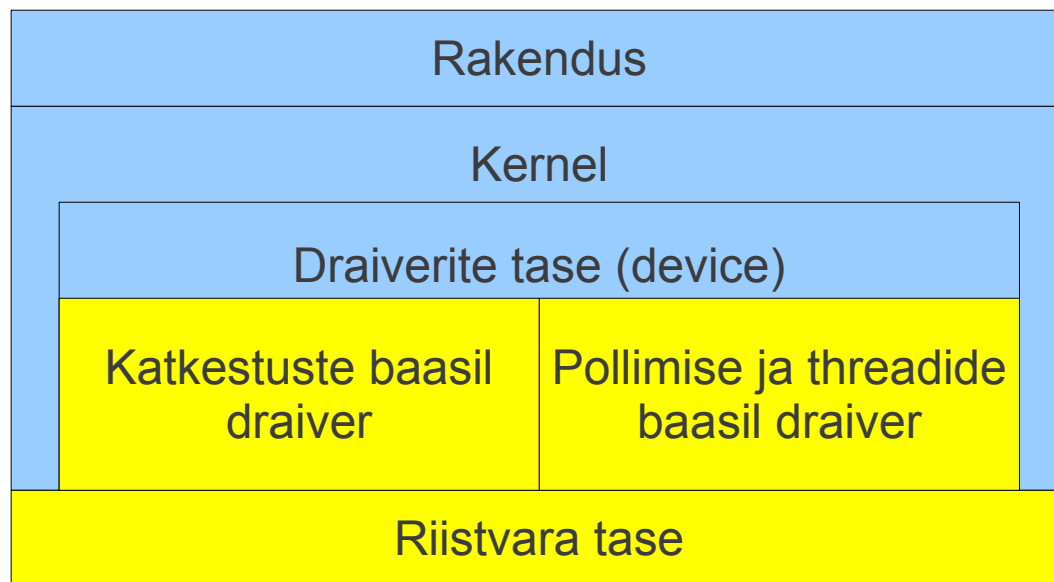


Standard OS



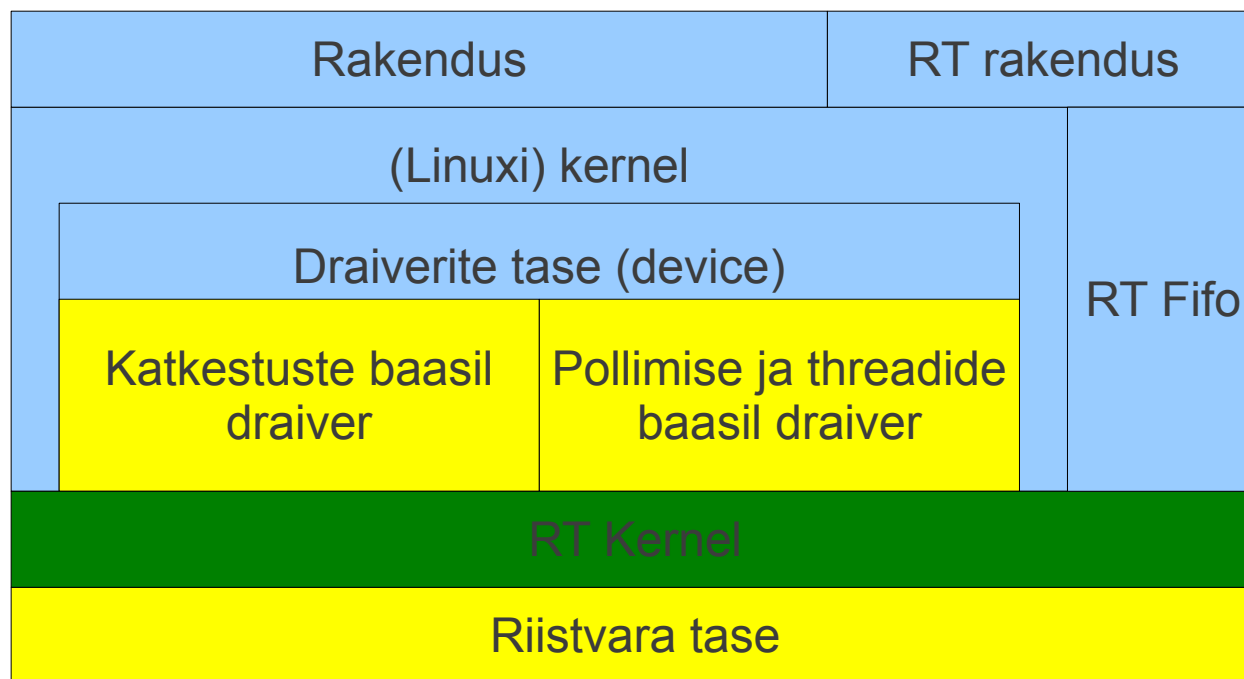
Standardne kernel koos rakendusega

- ▶ Rakendusel ei ole võimalik otse riistvaraga suhelda
 - Riistvaraga suhtlemine on kerneli privileeg
- ▶ Reeglina ei ole mõeldud real-time rakenduste jaoks
 - Võib oma ülesandeid täita erineva kiirusega



Real-time kernel koos rakendusega

- ▶ RT kernel tegeleb otse riistvara juhtimisega ja ühtlasi tagab ka RT rakendustele kiire täitmise
- ▶ RT rakendustel on võimalik suhteliselt lihtsalt riistvaraga suhelda



Real-time kernel vs „tavaline“ kernel

▶ RT kernel (hard real-time)

- Võimaldab kiiresti reageerida välistele signaalidele
- Rakendustel määratletud deadline'd
- Reageerimise ajad mikrosekundi kandis
- Miinuseks on suurem keerukus või kõrge hind

▶ „Tavaline“ kernel (soft real-time)

- Oluliselt suurem arv rakendusi kui RT kernelile
- Palju kasutajaid
- Parem riistvara tugi
- Reageerimise ajad millisekundi kandis (õigemini mõnest millisekundist mõnesajani)
- Tunduvalt odavam, kuid võib tahta rohkem ressursi

... kuid reaalselt

- ▶▶ Soft-realtime kernelid on piisavalt head enamuste toodetele
 - Soft-realtime ei sobi paljudel juhtudel meditsiinis, lennunduses, autodes, robotites
- ▶▶ Kui just pole tegemist mingite väga kiirete sündmustega ei ole RT kerneli väga suuri eeliseid tavalise kerneli ees
- ▶▶ Draiverite tasemel on võimalik teha soft-realtime kerneliga realtime rakendusi
- ▶▶ 8 bitiste kontrolleritega pole eriti keerukas teha hard-realtime süsteemi

Sardsüsteemides levinumad kernelid

- ▶▶ NutOS
- ▶▶ FreeRTOS
- ▶▶ Contiki
- ▶▶ eCos
- ▶▶ Linux

NutOS

- ▶▶ AVR, AVR32, ARM, H8300h ja m68k mikrokontrolleritele/protsessoritele
- ▶▶ Cooperative kernel
- ▶▶ Kerneliga kaasas draiverid
- ▶▶ Determineeritud katkestute latentsused
 - Ei kehti mitme samaaegse katkestuse korral
 - Draiverite tasemel võimaldab väga hästi teha hard real-time rakendust
- ▶▶ Kerneliga kaasas üpris palju lisaprogramme, näidiseid ja teeke
- ▶▶ Arendatakse spetsiaalse riistvara jaoks

FreeRTOS

- ▶▶ 23 erinevale arhitektuurile (AVR, AVR32, MSP430, PIC, ARM.. jne)
- ▶▶ Cooperative või preemptive kernel
- ▶▶ Draiverid näidisprogrammidega kaasas, kuid kernelil endal pole draivereid
- ▶▶ Portimine on üpriski lihtne
- ▶▶ Paralleelselt arendatakse ka OpenRTOS'i (ilma GNU litsentsita) ja SafeRTOS'i (IEC 61508 sertifitseeritud)
- ▶▶ Arendatakse üldise riistvara jaoks, st. arendajad ise ei tooda riistvara

Contiki

- ▶▶ Peamiselt väikestele kontrolleritele (AVR, MSP430)
 - Üks vähesema mälunõudlusega kernel
- ▶▶ Kasutab peamiselt protothreads'e kuid võimalik kasutada preemptive threade
- ▶▶ Palju draivereid on kerneliga kaasas
- ▶▶ Kasutatakse väga palju ülikoolide juures uurimistöode läbiviimisel (peamiselt smart dust)
 - Contiki on see kernel kust on võetud väga paljude teistele väikestele kernelitele võrgu stacki

eCos

- ▶▶ Võimsamatele mikrokontrolleritele/protsessoritele
 - ARM, CalmRISC, FR-V, FR30, H8, IA32, 68K/ColdFire, Matsushita AM3x, MIPS, NEC V8xx, PowerPC, SPARC, SuperH
- ▶▶ Võimalik kasutada mitmeid erinevaid *schedulere*
- ▶▶ Draiverid on kerneliga kaasas
- ▶▶ Hea kasutada seal kus Linuxit ei saa kasutada
- ▶▶ Arendatakse üldise riistvara jaoks

Linux

- ▶▶ Pordid olemas praktiliselt kõikidele võimsamatele protsessoritele
- ▶▶ Standartselt kaasas üpris hea scheduler
- ▶▶ Võimalik kasutada real-time kerneli lisasid
- ▶▶ Draiverid kerneliga kaasas
- ▶▶ Problemaatiline kasutada väikese energiatarbega süsteemides

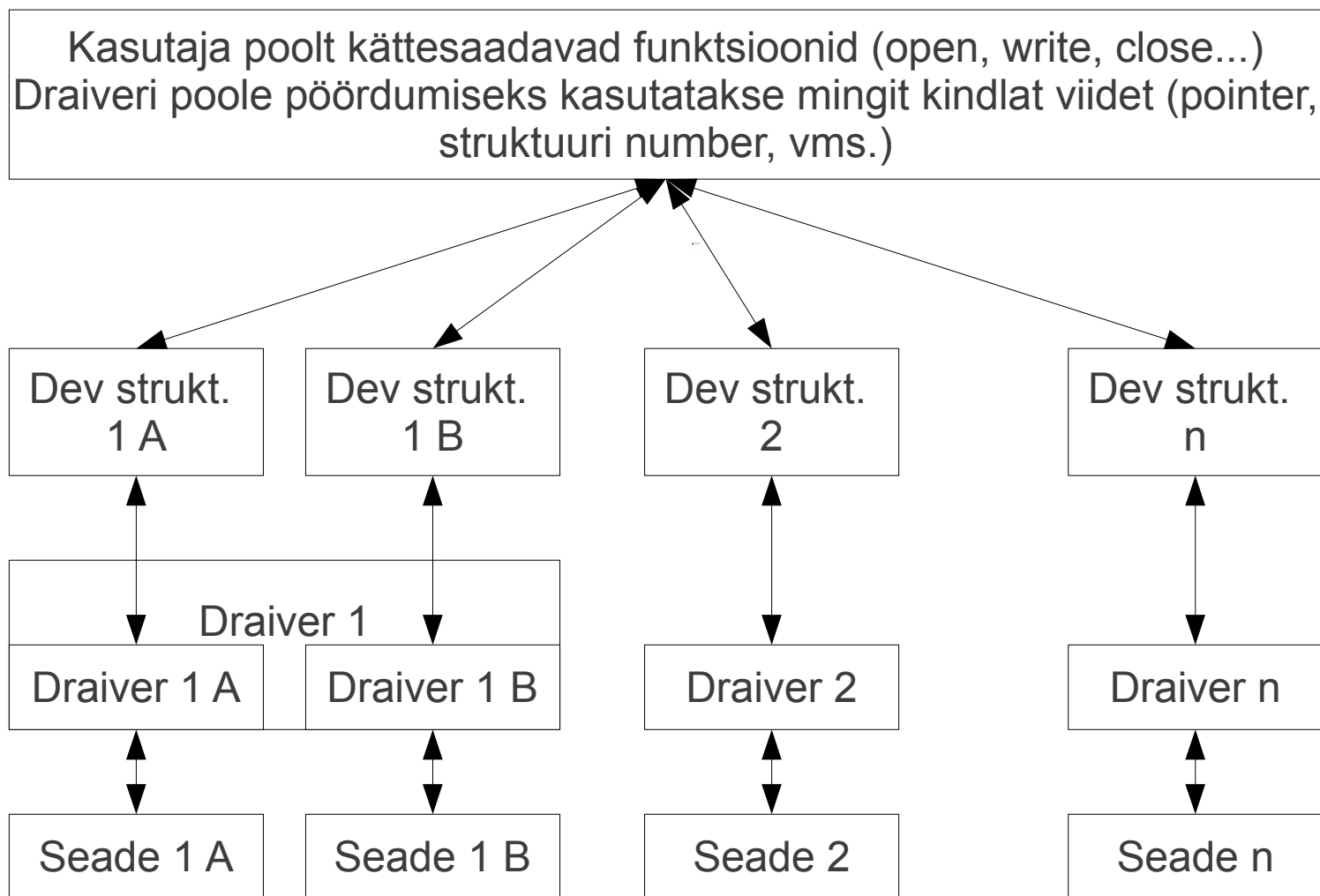
Kernelid üldiselt

- ▶ Kerneleid saab üldiselt jaotada ressurssi nõudluse järgi kolme gruppi:
 - 8/16 bitiste kontrollerite kernelid
 - Väiksemate 32 bitiste kontrollerite kernelid, mis on üldiselt ilma MMU'ta ja vähese mäluga
 - Protsessorite millel on MMU, palju mälu ning ei pea eriti energiat kokku hoidma
- ▶ Arenduses on alati parem kasutada kernelit, millel on rohkem draivereid ja lisaprogramme kaasas
 - Tihtipeale läheb kerneli portimine ühelt arhitektuurilt teisele tunduvalt lihtsamini kui teisele kernelile näiteks võrgu stacki külge panek

Draiverid

- ▶ Draiverid on need kerneli osad mille ülesanneteks on ainult riistvaraga suhtlemine
 - Kuna draiverite funktsioone kutsutakse tihtipeale väga palju välja siis peaksid draiverid olema kirjutatud nii, et nad võtaksid võimalikult vähe ressursi

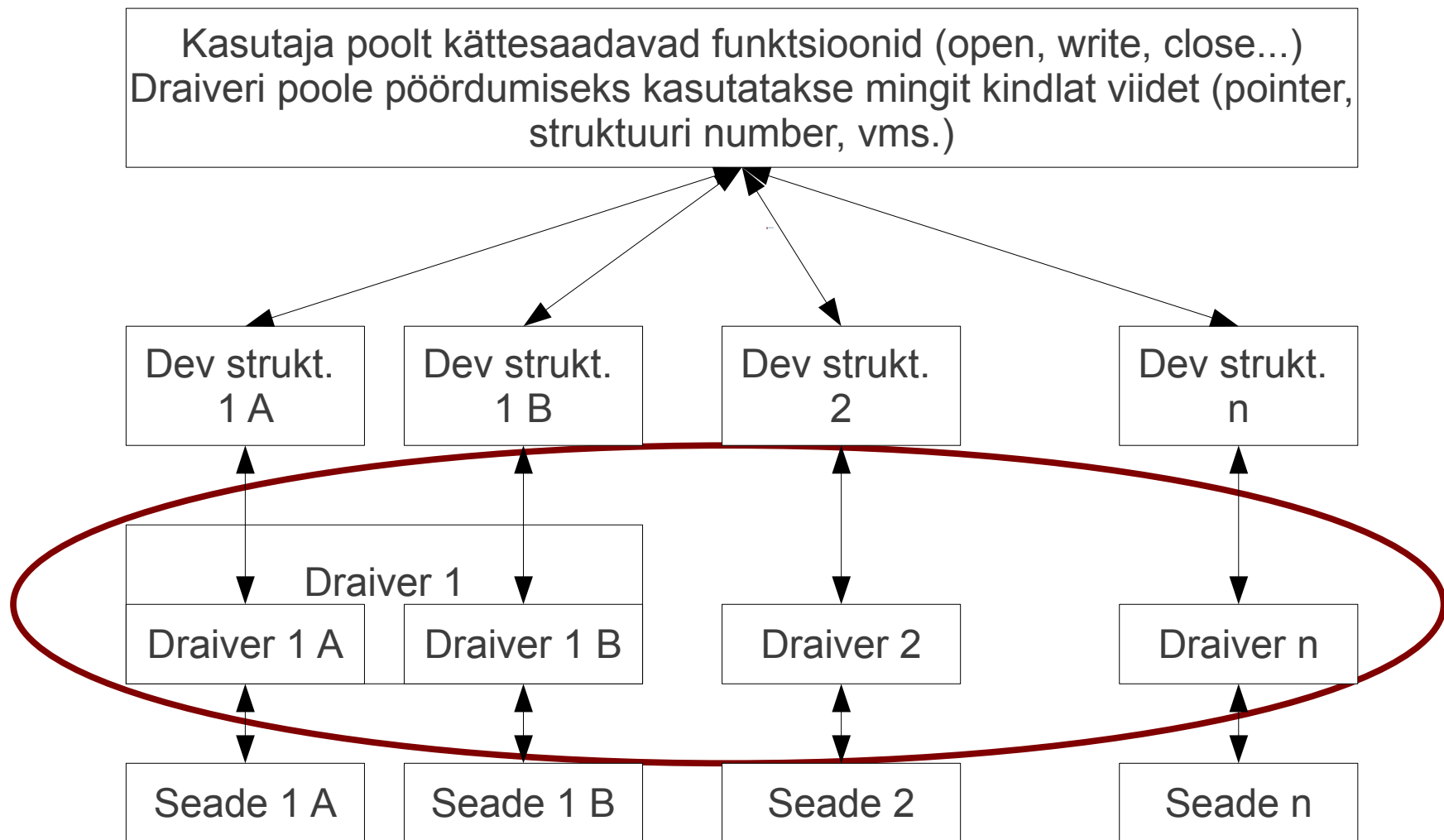
Tüüpiline suhtlus draiveriga



NutOS'i baasil draiveri näide

- ▶ Järgnevatel slaidide on toodud NutOS'i baasil usart draiveri näide. Näites ei ole täielikku koodi välja toodud vaid on toodud olulised koodilõigud

Riistvara lähedased funktsioonid



Riistvara lähedased draiveri funktsioonid

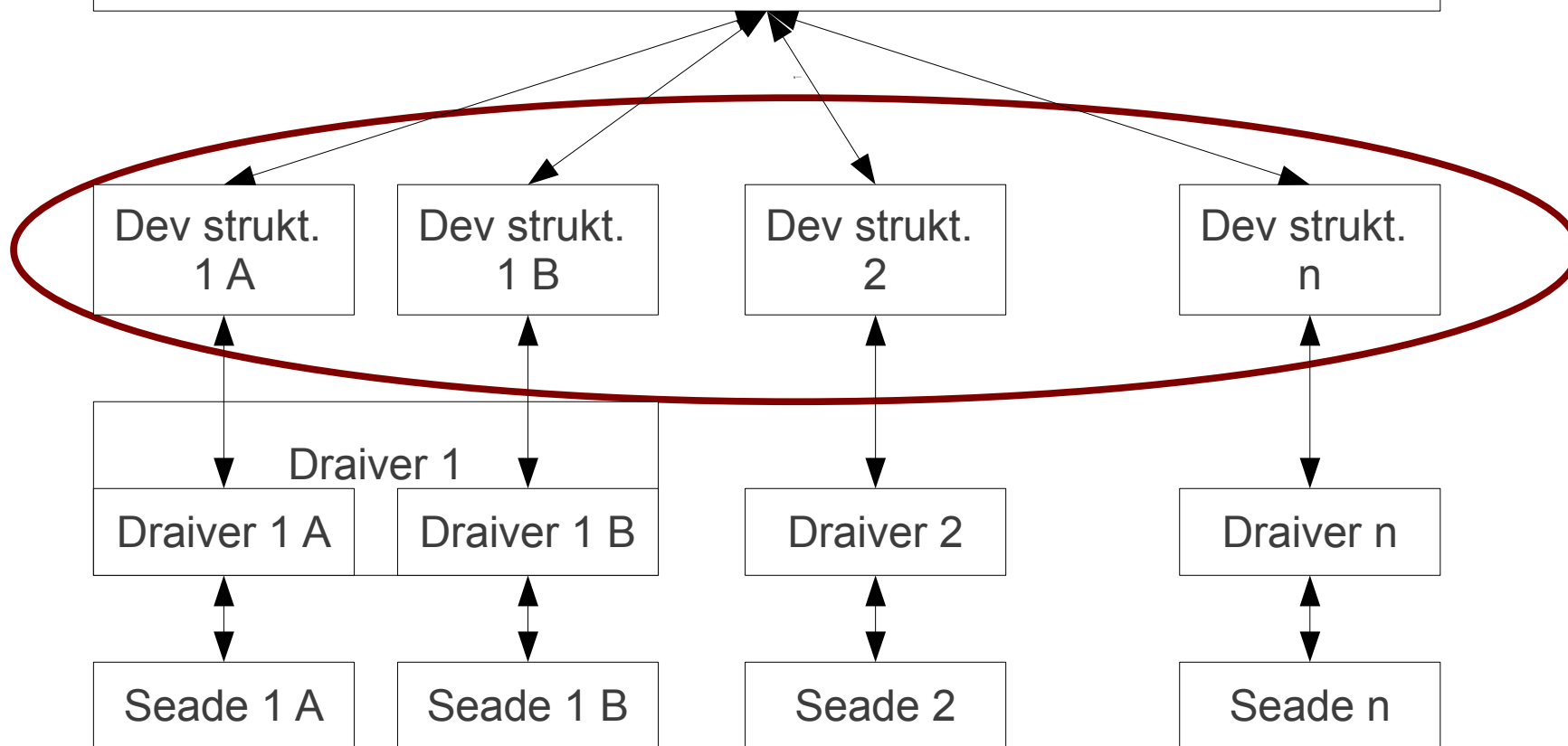
- ▶ Riistvara lähedased funktsioonid paigutatakse kõik DCB struktuuri (*device control block structure*)
 - Siia alla kuuluvad kõiksugused riistvara seadistamised ja portidest lugemised/kirjutamised.
 - DCB struktuuri puhul on tegemist kaudsete funktsioonide struktuuridega

DCB

```
static USARTDCB dcb_usart0 = {
    0, /* dcb_modeflags */
    0, /* dcb_statusflags */
    0, /* dcb_rtimeout */
    0, /* dcb_wtimeout */
    {0, 0, 0, 0, 0, 0, 0, 0}, /* dcb_tx_rbf */
    {0, 0, 0, 0, 0, 0, 0, 0}, /* dcb_rx_rbf */
    0, /* dcb_last_eol */
    AvrUsartInit, /* dcb_init */
    AvrUsartDeinit, /* dcb_deinit */
    AvrUsartTxStart, /* dcb_tx_start */
    AvrUsartRxStart, /* dcb_rx_start */
    AvrUsartSetFlowControl, /* dcb_set_flow_control */
    AvrUsartGetFlowControl, /* dcb_get_flow_control */
    AvrUsartSetSpeed, /* dcb_set_speed */
    AvrUsartGetSpeed, /* dcb_get_speed */
    AvrUsartSetDataBits, /* dcb_set_data_bits */
    AvrUsartGetDataBits, /* dcb_get_data_bits */
    AvrUsartSetParity, /* dcb_set_parity */
    AvrUsartGetParity, /* dcb_get_parity */
    AvrUsartSetStopBits, /* dcb_set_stop_bits */
    AvrUsartGetStopBits, /* dcb_get_stop_bits */
    AvrUsartSetStatus, /* dcb_set_status */
    AvrUsartGetStatus, /* dcb_get_status */
    AvrUsartSetClockMode, /* dcb_set_clock_mode */
    AvrUsartGetClockMode, /* dcb_get_clock_mode */
};
```

Riistvaralähedaste fun. ja kasutaja vahelised funktsioonid

Kasutaja poolt kättesaadavad funktsioonid (open, write, close...)
Draiveri poole pöördumiseks kasutatakse mingit kindlat viidet (pointer, struktuuri number, vms.)



Device struktuur

- ▶▶ Device struktuur on vahelüli kasutaja funktsioonide ja riistvara lähedaste funktsioonide vahel
- ▶▶ Iga draiver hoiab oma kirjutamise, lugemise ja IO juhtimise funktsioone globaalses Device struktuuris. Linuxis all hoitakse sarnaselt *char device*'i funktsioone analoogne struktuuris `file_operations`

Device structuur

```
struct _NUTDEVICE {
    NUTDEVICE *dev_next;
    char dev_name[9];
    uint8_t dev_type;
    uintptr_t dev_base;
    uint8_t dev_irq;
    void *dev_icb;
    void *dev_dcb;
    int (*dev_init) (NUTDEVICE *);
    int (*dev_ioctl) (NUTDEVICE *, int, void *);
    int (*dev_read) (NUTFILE *, void *, int);
    int (*dev_write) (NUTFILE *, CONST void *, int);
#ifdef __HARVARD_ARCH__
    int (*dev_write_P) (NUTFILE *, PGM_P, int);
#endif
    NUTFILE * (*dev_open) (NUTDEVICE *, CONST char *, int, int);
    int (*dev_close) (NUTFILE *);
    long (*dev_size) (NUTFILE *);
};
```

Device structuur

```
NUTDEVICE devUsartAvr0 = {
    0, /* Pointer to next device, dev_next. */
    {'u', 'a', 'r', 't', '0', 0, 0, 0, 0}, /* Unique device name, dev_name. */
    IFTYP_CHAR, /* Type of device, dev_type. */
    0, /* Base address, dev_base (not used). */
    0, /* First interrupt number, dev_irq (not used). */
    0, /* Interface control block, dev_icb (not used). */
    &dcb_usart0, /* Driver control block, dev_dcb. */
    UsartInit, /* Driver initialization routine, dev_init. */
    UsartIOctl, /* Driver specific control function, dev_ioctl. */
    UsartRead, /* Read from device, dev_read. */
    UsartWrite, /* Write to device, dev_write. */
    UsartWrite_P, /* Write data from program space to device, dev_write_P. */
    UsartOpen, /* Open a device or file, dev_open. */
    UsartClose, /* Close a device or file, dev_close. */
    UsartSize /* Request file size, dev_size. */
};
```

Draiveri kõrgematasemelised funktsioonid

```
int UsartInit(NUTDEVICE * dev)
{
    int rc;
    USARTDCB *dcb = dev->dev_dcb;

    /* Initialize the low level hardware
     * driver. */
    if ((rc = (*dcb->dcb_init) ()) == 0) {
        /* Ignore errors on initial
         * configuration. */
        (*dcb->dcb_set_speed)(USART_INITSPEED);
    }
    return rc;
}

/*****/

int UsartRead(NUTFILE * fp, void *buffer,
             int size)
{
    NUTDEVICE *dev = fp->nf_dev;
    USARTDCB *dcb = dev->dev_dcb;
    RINGBUF *rbf = &dcb->dcb_rx_rbf;

    /* seadme lugemise funktsioonid */
};
```

- ▶ Device struktuuri liikmed viitavad vastavate draiveri kõrgematasemeliste funktsioonidele
- ▶ Draiveri kõrgematasemelised funktsioonid võivad olla ühised mitmele madalmatasemelisele draiveri funktsioonile

Draiveri kõrgematasemelised funktsioonid

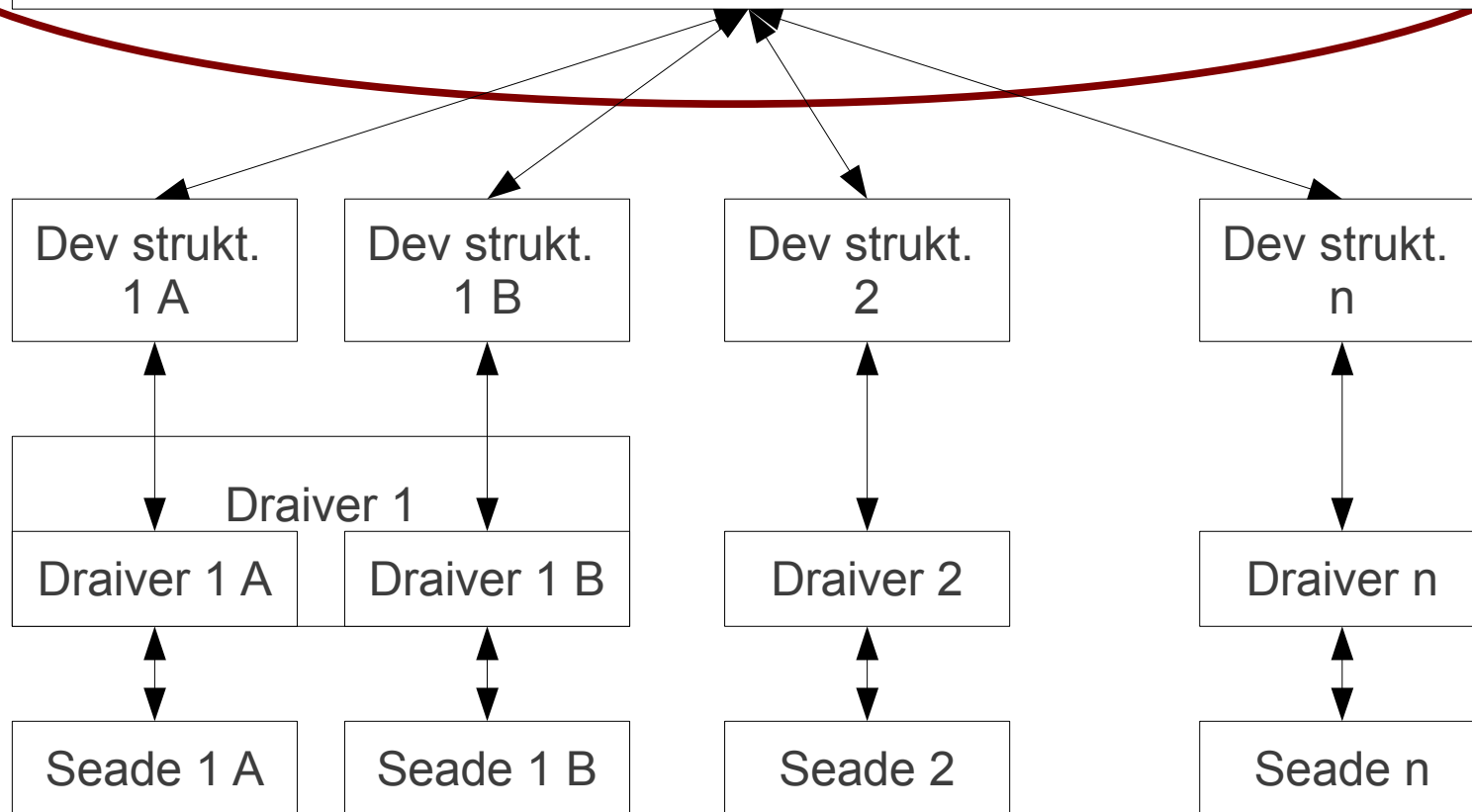
- ▶ Draiveri kõrgematasemeliste funktsioonide kasutamine annab järgmisi eeliseid:
 - Nendega hoiab mälu kokku
 - Kuna draiveri kõrgematasemelised funktsioonid peavad olema porditavad, siis peab vähem riistvaraliste funktsioonide kirjutamistel programmi kirjutama

Linux file_operations structuur

```
struct file_operations {
    struct module *owner;
    loff_t (*llseek) (struct file *, loff_t, int);
    ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
    ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
    ssize_t (*aio_read) (struct kiocb *, const struct iovec *, unsigned long, loff_t);
    ssize_t (*aio_write) (struct kiocb *, const struct iovec *, unsigned long, loff_t);
    int (*readdir) (struct file *, void *, filldir_t);
    unsigned int (*poll) (struct file *, struct poll_table_struct *);
    int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);
    long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
    long (*compat_ioctl) (struct file *, unsigned int, unsigned long);
    int (*mmap) (struct file *, struct vm_area_struct *);
    int (*open) (struct inode *, struct file *);
    int (*flush) (struct file *, fl_owner_t id);
    int (*release) (struct inode *, struct file *);
    int (*fsync) (struct file *, struct dentry *, int datasync);
    int (*aio_fsync) (struct kiocb *, int datasync);
    int (*fasync) (int, struct file *, int);
    int (*lock) (struct file *, int, struct file_lock *);
    ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t *, int);
    unsigned long (*get_unmapped_area)(struct file *, unsigned long, unsigned long,
        unsigned long, unsigned long);
    int (*check_flags)(int);
    int (*flock) (struct file *, int, struct file_lock *);
    ssize_t (*splice_write)(struct pipe_inode_info *, struct file *, loff_t *, size_t,
        unsigned int);
    ssize_t (*splice_read)(struct file *, loff_t *, struct pipe_inode_info *, size_t,
        unsigned int);
    int (*setlease)(struct file *, long, struct file_lock **);
};
```

Kasutajale kättesaadavad funktsioonid

Kasutaja poolt kättesaadavad funktsioonid (open, write, close...)
Draiveri poole pöördumiseks kasutatakse mingit kindlat viidet (pointer, struktuuri number, vms.)



Rakendusele kättesaadavad funktsioonid

```
int _read(int fd, void *buffer,
          unsigned int count)
{
    NUTFILE *fp = (NUTFILE *) ((uintptr_t) fd);
    NUTDEVICE *dev;

    NUTASSERT(fp != NULL);
    dev = fp->nf_dev;
    if (dev == 0) {
        NUTVIRTUALDEVICE *vdev =
            (NUTVIRTUALDEVICE *) fp;
        return (*vdev->vdev_read)
            (fp, buffer, count);
    }
    return (*dev->dev_read) (fp, buffer, count);
}
```

- ▶ Rakendus saab kasutada device struktuuri kaudu vastavaid draiverite funktsioone
- ▶ Rakendusele kättesaadavad funktsioonid võivad olla mitmetele draiveritele ühised

Energia säästmine kontrolleriil

▶▶ Energia säästmine tööajal

- Madala kontrolleri taktiga
- „Kerge“ progarmeerimiskeeltega
- Kontrolleri madala energia tarbega režiimiga
- Operatsioonisüsteemi vastava režiimiga

Kontrolleri takt

- ▶▶ Kontrolleri takt ja energiatarve on omavahel lineaarses sõltuvuses
 - Tavaliselt antakse kas voolutarve/MHz'le või voolutarve/MIPS'le. Näiteks MSP430F551X on 200uA/MIPS, kuid AT32UC3 on 300uA/MHz
- ▶▶ Väikse energiatarbe saavutamiseks võib tuua kontrolleri takti võimalikult madalaks
 - Takti võimalikult madalaks toomine toimib hästi ainult lihtsate programmidega – keerukamad ei pruugi kõikide katkestustega toime tulla

Energia sääst ja programmeerimiskeeled

- ▶▶ Interpreteeritavad keeled (Java) ja ka mõningatel juhtudel „turvalised“ keeled (Ada) nõuavad töötamiseks rohkem energiat
 - Mida pikemalt kontroller on aktiivses režiimis seda rohkem ta energiat kulutab
- ▶▶ Kõrgematasemelised keeled (Java) ei suhtle ise riistvaraga ja seetõttu peab kernel protsessori režiimidega tegelema
 - Java puhul saaks riistvaraga suhtlemise küll JNI'ga teha

Mikrokontrolleri madala energia tarbega režiimid

- ▶▶ Enamustel mikrokontrolleritel on võimalik kasutada madala energiatarbega režiime
 - Näiteks 8 bitisel AVR mikrokontrolleril on kuus erinevat sleep režiimi, 16 bitisel MSP430 mikrokontrolleril on viis kuni seitse erinevat sleep režiimi
- ▶▶ Mikrokontrolleril on võimalik lülitada mittekasutatavad liidesed välja
 - Liideste välja lülitamine ei anna väga suurt energia kokkuhoidu, tavaliselt hoiab nii kokku kuni 20%

Operatsioonisüsteemide valmisolek

- ▶▶ Enamus kerneleid ei tegele pidevalt energia säästu režiimide kontrollimise ja vahetamisega
 - Selliseid lahendusi kernelis on suhteliselt keerukas teha
- ▶▶ On võimalik sundida kernelit/mikrokontrollerit minema sleep režiimi
 - Võib tekkida probleeme riistvara suhtlemisel

Küsimusi?