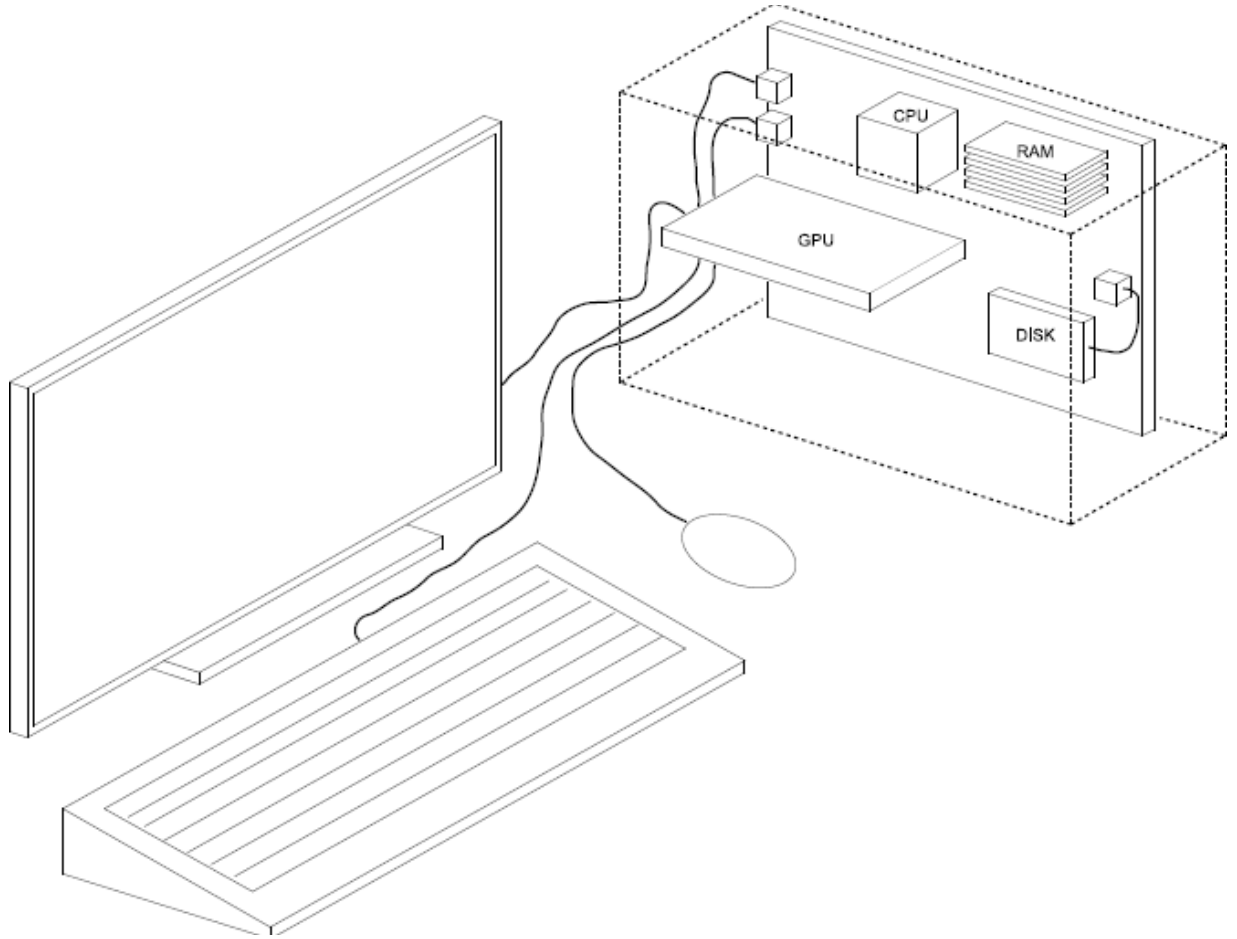
NOS – Naprave

01
Korištenje naprava preko operacijskog sustava

Sučelja OS-a

Što su to naprave?

- računalo se sastoji od:
  - procesora
  - memorije
  - sabirnice
  - sve ostalo su naprave
    - disk
    - grafička kartica (monitor)
    - mrežna kartica
    - tipkovnica, miš (preko USB-a)
    - …



Kako klasificirati naprave? Kategorije, klase, …

- fizičke dimenzije, načine spajanja na/u računalu, brzina rada
- smjer podataka: ulazna, izlazna, ulazno-izlazna
- dohvat podataka: znak po znak ili blokovi podataka (npr. sektor, paket)
- čitanje je slijedno ili je moguć dohvat bilo kojeg podatka
- sinkrono ili asinkrono (čeka se na završetak operacije ili ne)
- istovremeno korištenje od strane više dretvi ili ne
- OS naprave svrstava u zasebne klase
  - npr. na Linuxu su klase: znakovne, blokovske, mrežne

Primjeri naprava i njihovih brzina rada

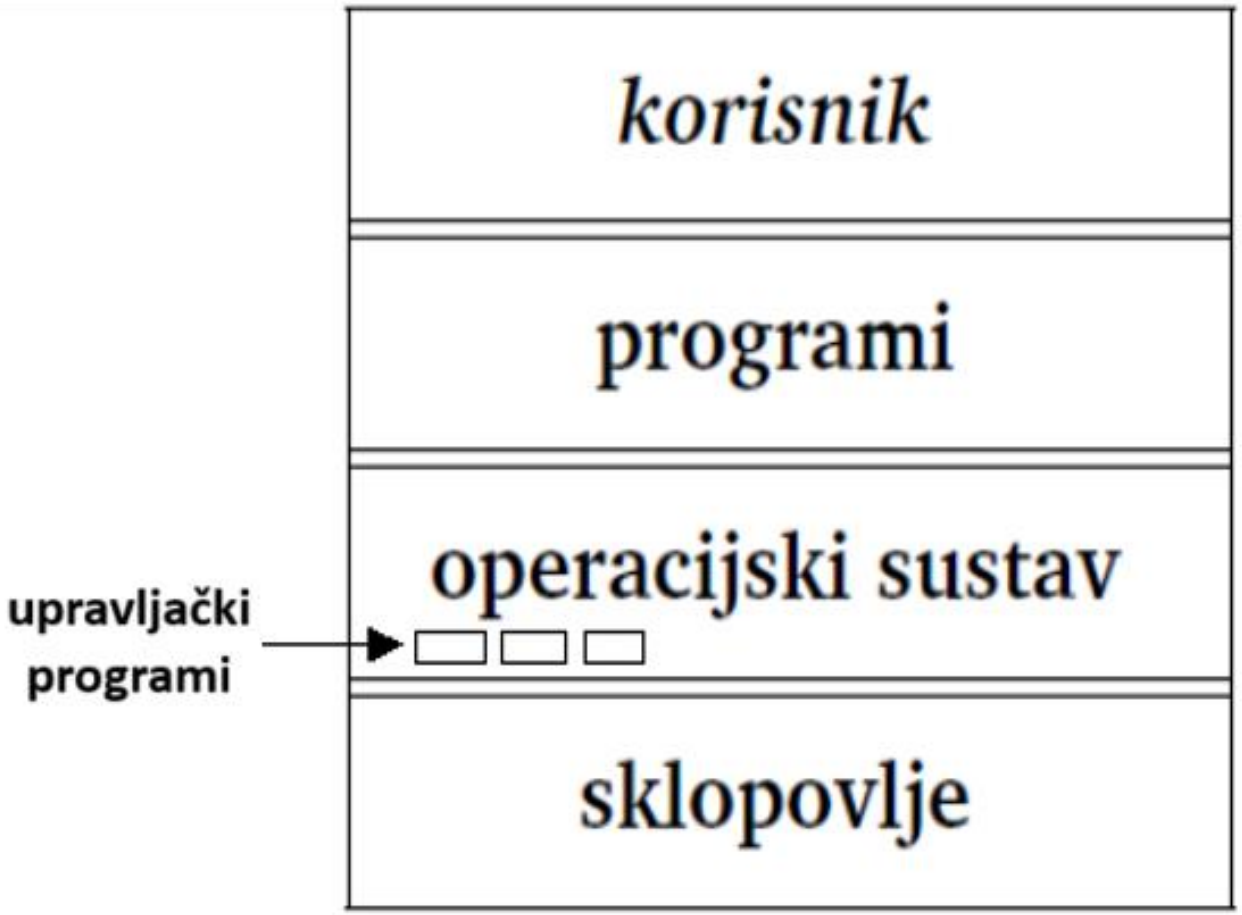
- tipkovnica i miš (USB): 125-1000 Hz
- audio podsustav (analogni/digitalni ulaz/izlaz): 44, 48, 96, ... kHz
- disk (SATA):
  - čitanje jednog podatka ~10 ms za HDD (~0.05 ms za SSD);
  - prijenos kompaktno smještenih podataka ~100 MB/s (i više)
- mrežna kartica (PCIe): npr. 1 Gbit/s ⇒ 125 MB/s; odziv mreže od ~0.1 μs do ~100 ms
- grafička kartica (PCIe): npr. za 1920x1080 sa 60 Hz ⇒ ~124 MHz

Upravljanje napravama radi OS. Kako?

- naprave su spojene na neke upravljačke sklopove (kontrolere)
  - npr. USB kontroler, disk kontroler, PCIe kontroler
- upravljanje napravama ide kroz te kontrolere
  - često je dovoljno kroz sučelje tog kontrolera upravljati napravama
  - ponekad kontroler samo služi kao „niži sloj” za prijenos poruka
- upravljački program za napravu (engl. device driver)
  - sadrži dodatne funkcije za pristup napravi
  - OS nudi „kostur” za naprave, a upravljački program popunjava neke rupe

Programi koriste naprave preko sučelja OS-a

- OS može izvoditi privilegirane instrukcije potrebne za upravljanje
- OS kontrolira pristup napravama
  - paralelni rad programa
  - ograničava pristup
- složenost upravljanja napravama je skrivena u OS-u
  - sučelje prema programima je jednostavno



Sučelje OS-a za korištenje naprava

- naprave su uglavnom dio nekog drugog podsustava
  - iz programa se neizravno koriste, kroz sučelje tih podsustava
    - primjeri: mrežni podsustav, datotečni podsustav, grafički podsustav
- ostale naprave se najčešće mogu koristiti kroz „generičko sučelje”
  - skoro identično sučelju za rad s datotekama (na UNIX-u to i jesu)
  - u nastavku se razmatraju samo takva sučelja

Osnovna sučelja (generička)

- otvori/zatvori, čitaj/piši
  - `int open(const char *pathname, int flags);`
  - `int open(const char *pathname, int flags, mode_t mode);`
  - `int close(int fd);`
  - `ssize_t read(int fd, void *buf, size_t count);`
  - `ssize_t write(int fd, const void *buf, size_t count);`
- ovisno o napravi, moguće i:
  - `off_t lseek(int fd, off_t offset, int whence);`

Primjer korištenja sučelja:

```
int fd;
char ms[11];
char dat[] = "test.txt";
char poruka[] = "1234567890XXXXXABCDEFGHIJXXXXX";

fd = open(dat, O_CREAT | O_WRONLY, 00600);
write(fd, poruka, strlen(poruka));
close(fd);

fd = open(dat, O_RDONLY);
memset(ms, 0, 11);
read(fd, ms, 10);
printf("procitano: %s\n", ms);

lseek(fd, 5, SEEK_CUR);//preskoči idućih pet znakova
memset(ms, 0, 11);
read(fd, ms, 10);
printf("procitano: %s\n", ms);
close(fd);
```

Dodatne operacije

- `int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout);`
- `int poll(struct pollfd *fds, nfds_t nfds, int timeout);`
- `int epoll_wait(int epfd, struct epoll_event *events, int maxevents, int timeout);`
- `int fsync(int fd);`
- `int mknod(const char *pathname, mode_t mode, dev_t dev);`
- `int fcntl(int fd, int cmd, ... /*arg */);`
- `int ioctl(int fd, unsigned long request, ...);`
- `ssize_t readv/writev(int fd, struct iovec *iov, int iovcnt);`

Asinkrone operacije – ideje

- zadaje se naredba ali ne čeka njen kraj
- nekoliko načina da se čeka/dozna za kraj operacije
  - čekanje dodatnim pozivom
  - provjeravanje statusa
  - primitak signala
- u nastavku su prikazane strukture podataka koje se za to koriste te primjeri funkcija
  - ti „detalji” su navedeni radi lakšeg razumijevanja načela rada tih mehanizama
  - ne pitaju se od studenata!

Asinkrone operacije – zahtjev kroz strukturu

```
struct aiocb {
        int aio_fildes;                 /* File descriptor */
        off_t aio_offset;               /* File offset */
        volatile void *aio_buf;         /* Location of buffer */
        size_t aio_nbytes;              /* Length of transfer */
        int aio_reqprio;                /* Request priority */
        struct sigevent aio_sigevent;   /* Notification method */
        int aio_lio_opcode;             /* Operation to be performed;
                                           lio_listio() only */

};
```

Asinkrone operacije – primjeri sučelja (1)

- `int aio_read/aio_write(struct aiocb *aiocbp);`
  - započni asinkronu operaciju čitanja/pisanja
- `int aio_fsync(int op, struct aiocb *aiocbp);`
  - sinkroniziraj operacije ("zapiši iz međuspremnika na naprave")
- `int aio_error(struct aiocb *aiocbp);`
  - dohvati grešku povezanu s zadanom operacijom
- `ssize_t aio_return(struct aiocb *aiocbp);`
  - dohvati status operacije

Asinkrone operacije – primjeri sučelja (2)

- `int aio_suspend(struct aiocb *aiocb_list[], int nitems, struct timespec *timeout);`
  - čekaj kraj operacija
- `int aio_cancel(int fd, struct aiocb *aiocbp);`
  - zaustavi operacije
- `int lio_listio(int mode, struct aiocb *const aiocb_list[], int nitems, struct sigevent *sevp);`
  - zadaj više operacija s jednim pozivom

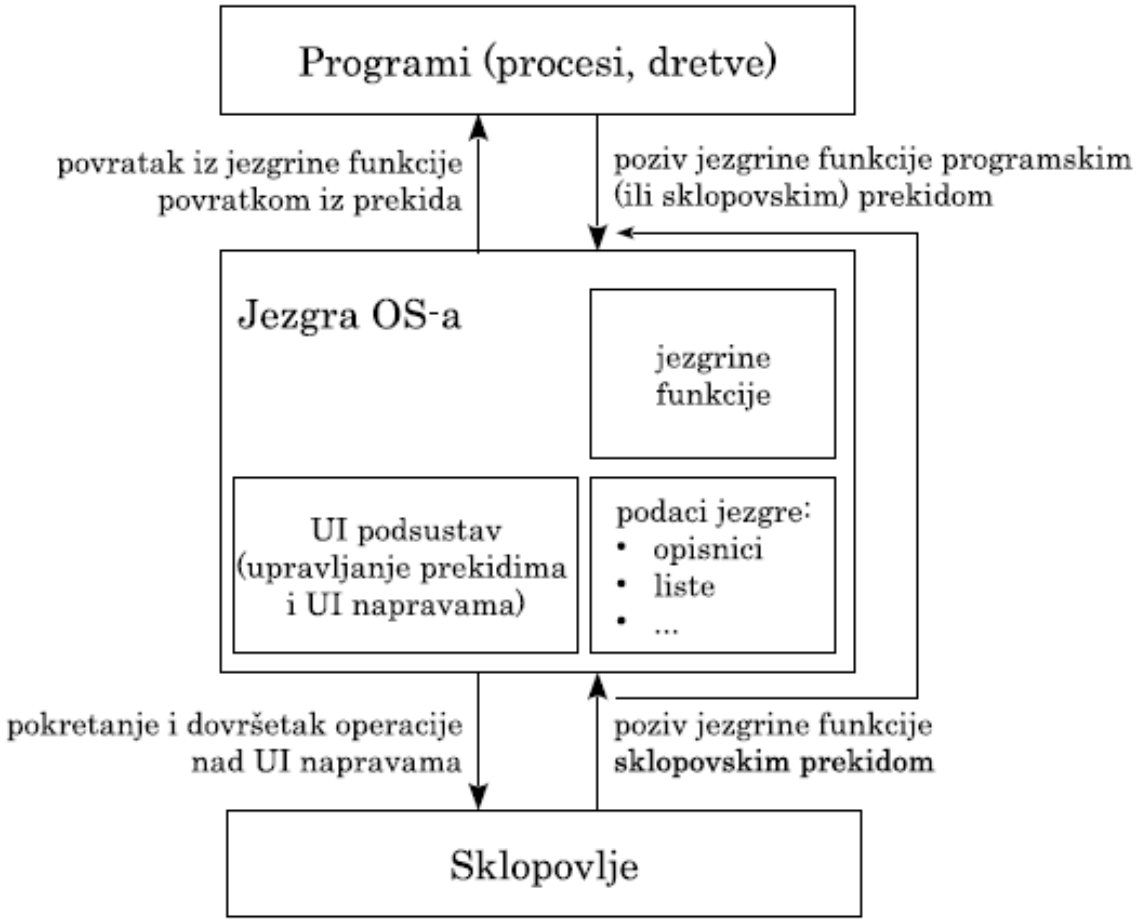
02
Jednostavni model OS-a za upravljanje napravama

Što smo već čuli o napravama u okviru predmeta OS?

Problemi i neka rješenja.

Programi (dretva) ←→ OS ←→ naprave

- dretve upotrebljavaju naprave preko OS-a
- funkcije OS-a == jezgrine funkcije; dretve ih pozivaju mehanizmom programskog prekida
- prekidi koje izazivaju naprave također prekidaju dretve
- kako OS koristi naprave?
  - iz predmeta OS:
    - radno čekanje
    - prekidi
    - izravan pristup spremniku



Radno čekanje

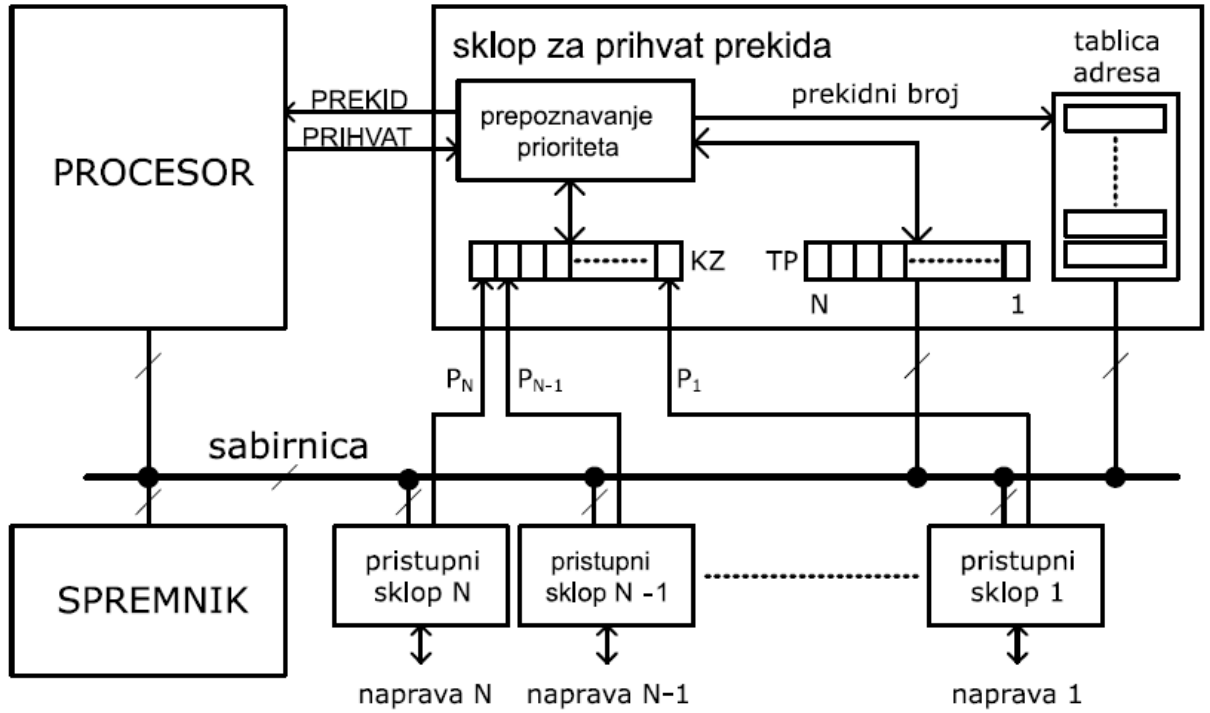
- petlja u kojoj se čeka da naprava bude spremna, tj. petlja u kojoj se neprestano provjerava statusni registar naprave

```
dok je status == NEMA promjene
        ;
dohvati podatak / pošalji novi podatak
```

- problem: neefikasno korištenje računala
- u stvarnim primjenama radi se „prozivanje”, provjeri ima li naprava nešto i ako nema onda se ide dalje na drugu…
  - ne koristi se radno čekanje, već periodički pristupa napravama

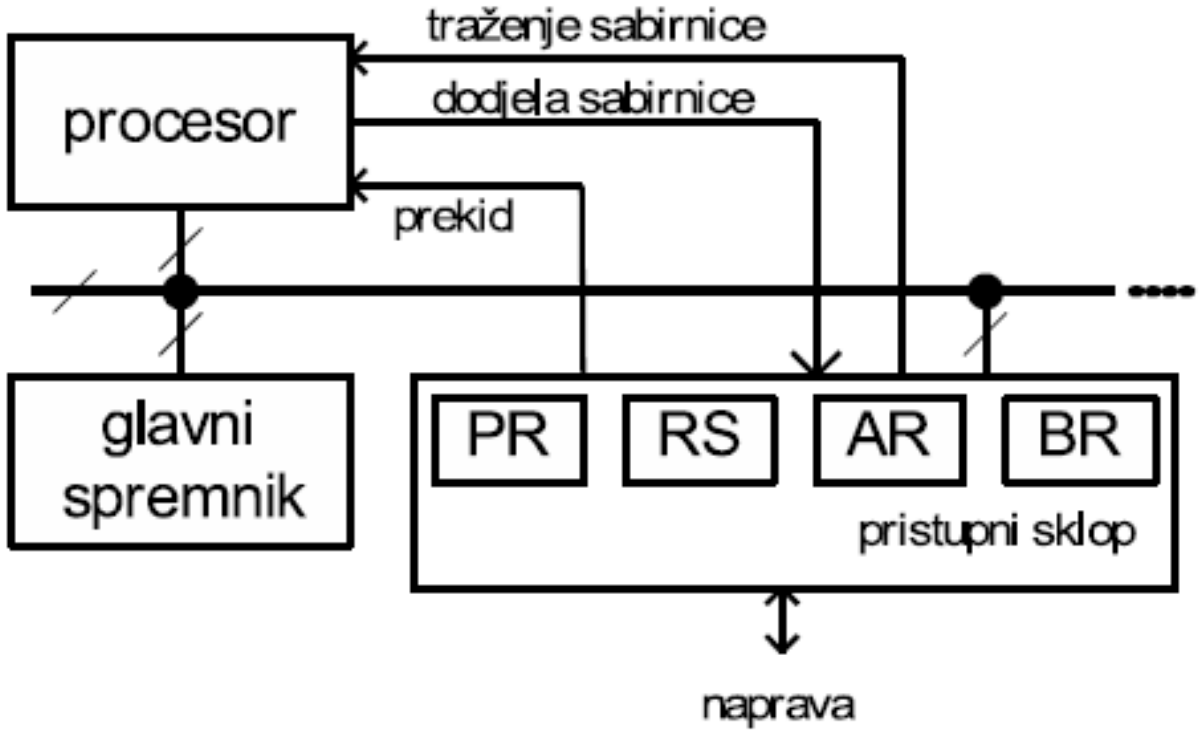
Prekidi naprava

- procesor radi neki korisni posao – izvodi neku dretvu
- kada naprava treba pažnju ona tada izaziva prekid
- procesor privremeno prekida izvođenje dretve, obrađuje prekid naprave te nakon toga nastavlja s prekinutom ili nekom drugom dretvom
- prekid traži dodatne poslove – ima dodatni *overhead*



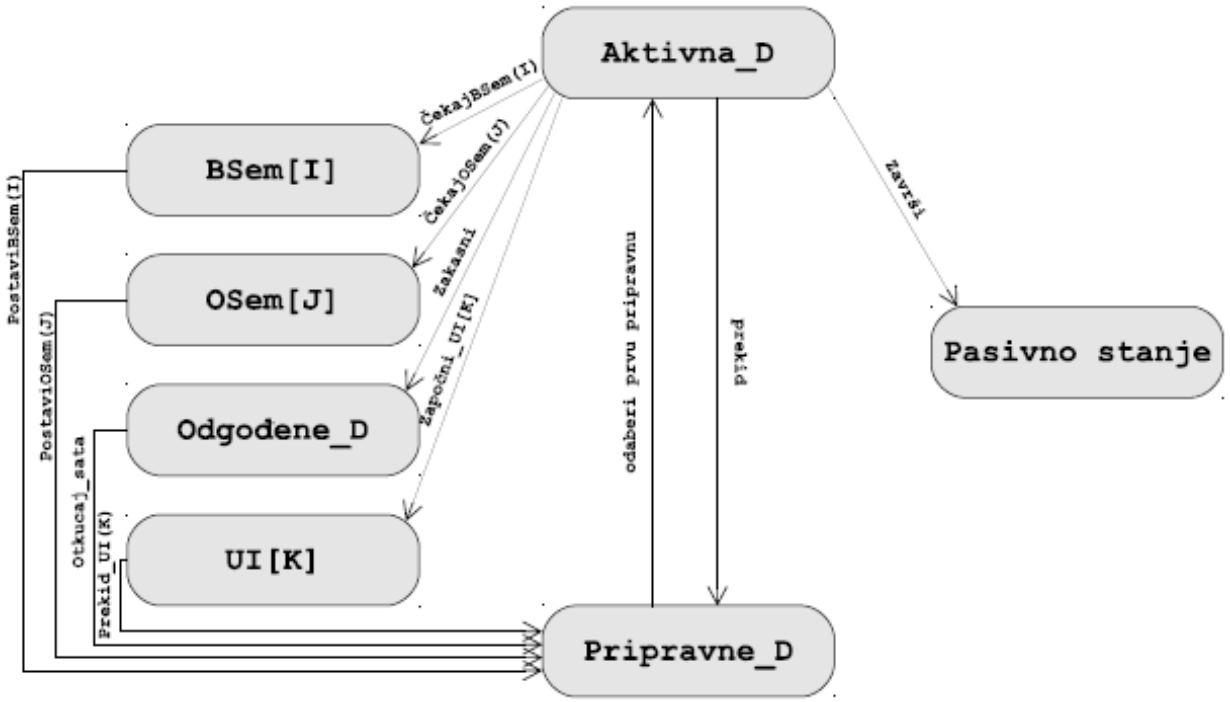
Izravan pristup spremniku

- neke naprave traže prijenos puno podataka u jednom i/ili drugom smjeru
- mehanizam prekida je neučinkovit
- korištenjem pristupnih sklopova s izravnim pristupom spremniku one same traže upravljanje sabirnicom i same prenose podatke



Stanja dretvi, jezgrine funkcije

- dretve rade korisne operacije
- OS upravlja dretvama i nudi dodatne operacije sinkronizacije, upravljanja vremenom te upravljanje napravama
- jezgrine funkcije izvode se unutar obrade prekida



Jednostavni model jezgrinih funkcija za naprave

- dretve traže operaciju od OS-a za naprave
- naprave prekidom javljaju kraj operacije

```
j-funkcija ZAPOČNI_UI(K, parametri) {
   stavi_u_red(makni_prvu_iz_reda(Aktivna_D), UI[K])
   pokreni UI operaciju na napravi K
   stavi_u_red(makni_prvu_iz_reda(Pripravne_D), Aktivna_D)
}
j-funkcija PREKID_UI(K) {
   stavi_u_red(makni_prvu_iz_reda(Aktivna_D), Pripravne_D)
   dovrši UI operaciju na napravi K
   stavi_u_red(makni_prvu_iz_reda(UI[K]), Pripravne_D)
   stavi_u_red(makni_prvu_iz_reda(Pripravne_D), Aktivna_D)
}
```

Problemi jednostavnog model i neka rješenja

- pretpostavlja se da naprava obrađuje zahtjeve po redu prispijeća
  - ali u stvarnim sustavima to često nije tako
  - npr. disk može optimirati zahtjeve; odgovori preko mreže …
- operacija koju dretva traži može biti složena, tražiti više operacija nad napravom ili čak više naprava
  - npr. čitanje iz datoteke može tražiti više zahtjeva prema disku
- Neki mehanizmi stvarnih sustava:
  - redovi zahtjeva za UI operacije
  - jezgrin kontekst dretve

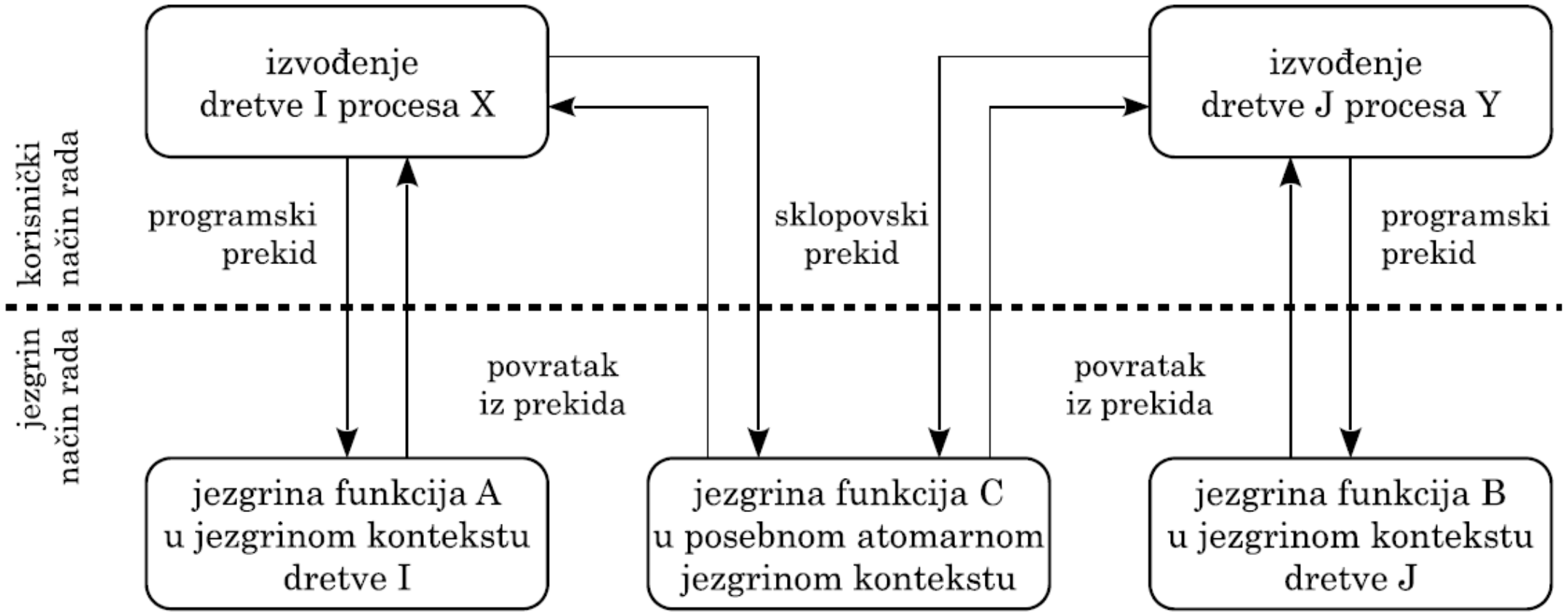
Redovi zahtjeva za UI operacije

- svaki zahtjev se oblikuje kao struktura i ide u listu zahtjeva
- obrada pojedinog zahtjeva može tražiti izvođenje više operacija
- tek kad je zahtjev gotov (odrađen do kraja od strane naprave i jezgre), dretva koja je dala taj zahtjev može nastaviti s radom
- zahtjev tako može duže ostati u sustavu, tražiti složenije operacije nad napravama

Jezgrin kontekst dretve (1)

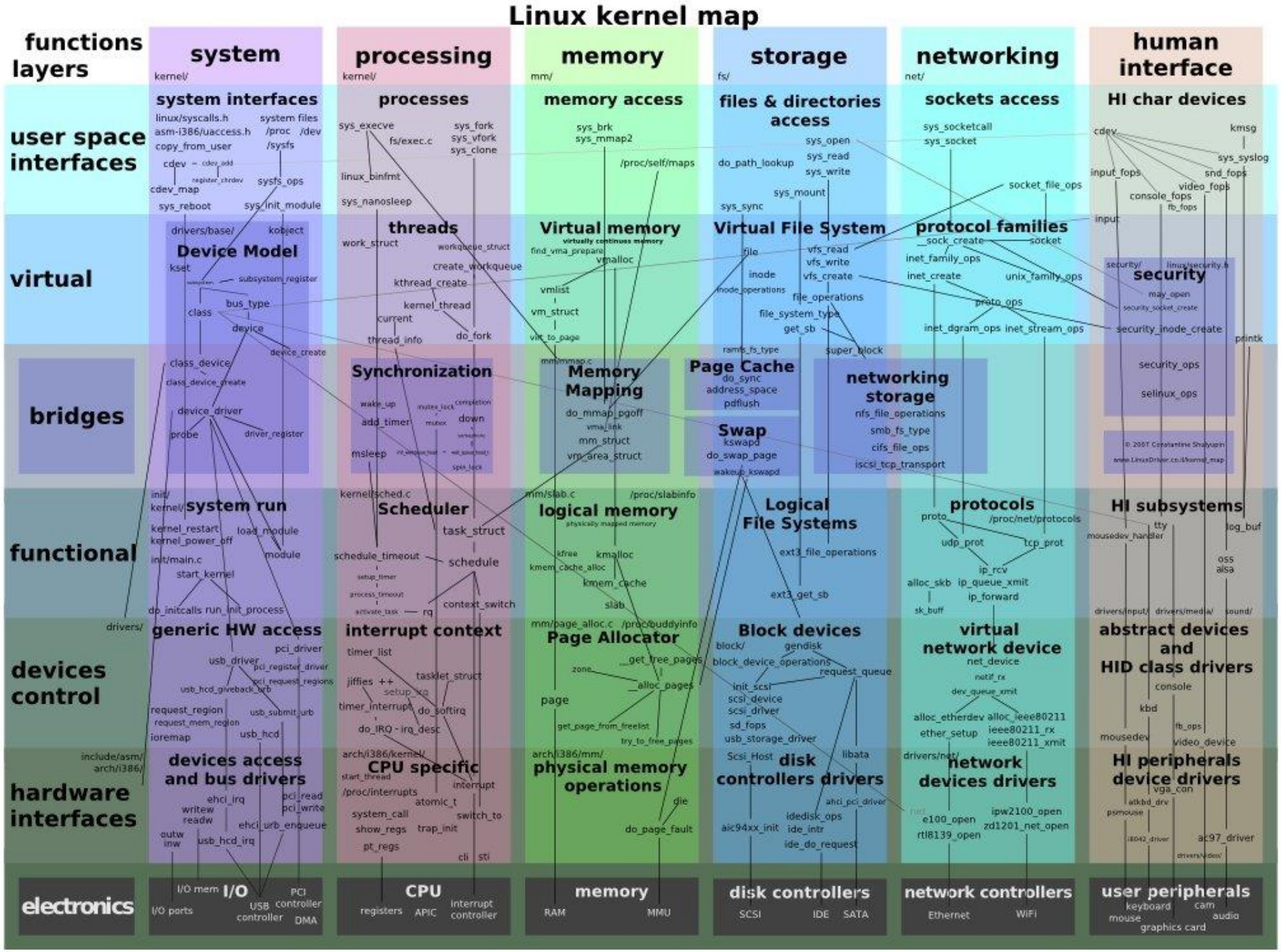
- nakon ulaska u jezgru mehanizmom programskog prekida, aktivira se jezgrin kontekst za zadanu dretvu
- takva jezgrina dretva može biti blokirana u jezgri
- pri blokiranju takve dretve, neka druga dretva nastavlja s radom (jezgrina ili korisnička)
- istovremeno se može izvoditi više jezgrinih funkcija ako im nisu potrebna ista sredstva (zaključavanje unutar jezgre se svodi na minimum)

Jezgrin kontekst dretve (2)



Stvarni sustavi su značajno složeniji – višeslojni

- Stvarni sustavi su značajno složeniji, i u sklopovlju i programskoj potpori
- Složenost se u stvarnim sustavima “rješava” podjelom na podsustave i slojeve
- Iduća prikazuje internu organizaciju Linuxa, podijeljenog na podsustave i slojeve

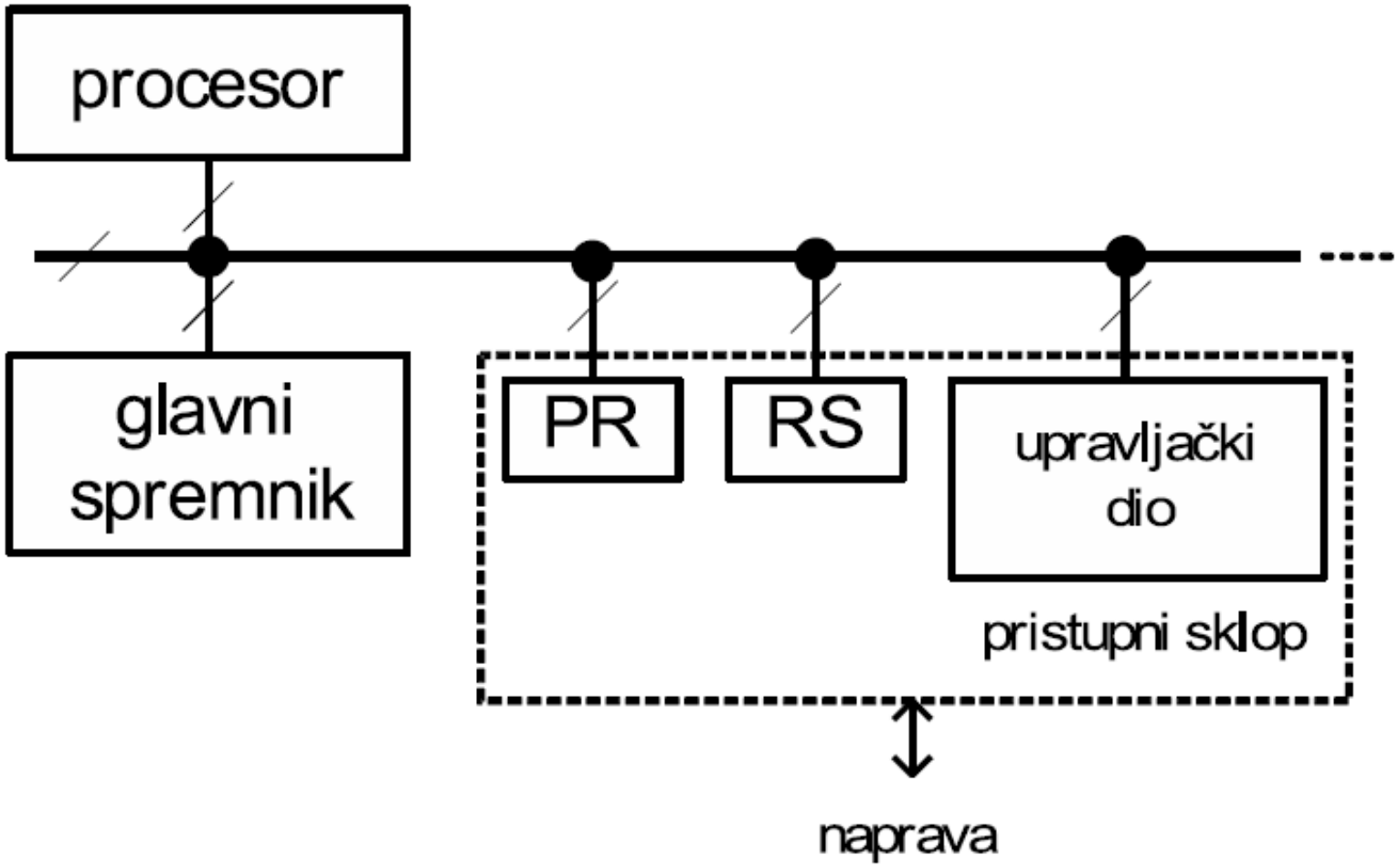

Linux kernel map
functions
layers
system
kernel/
processing
kernel/
memory
mm/
storage
fs/
networking
net/
human
interface
user space
interfaces
system interfaces
processes
memory access
files & directories
access
sockets access
HI char devices
virtual
Device Model
threads
Virtual memory
Virtual File System
protocol families
security
bridges
Synchronization
Memory
Mapping
Page Cache
networking
storage
Swap
functional
system run
Scheduler
logical memory
Logical
File Systems
protocols
HI subsystems
devices
control
generic HW access
interrupt context
Page Allocator
Block devices
virtual
network device
abstract devices
and
HID class drivers
hardware
interfaces
devices access
and bus drivers
CPU specific
physical memory
operations
disk
controllers drivers
network
devices drivers
HI peripherals
device drivers
electronics
I/O
CPU
memory
disk controllers
network controllers
user peripherals

03
Model naprava

Kako se pristupa napravama?

Zašto i kako se koriste međuspremnici?

Model naprave



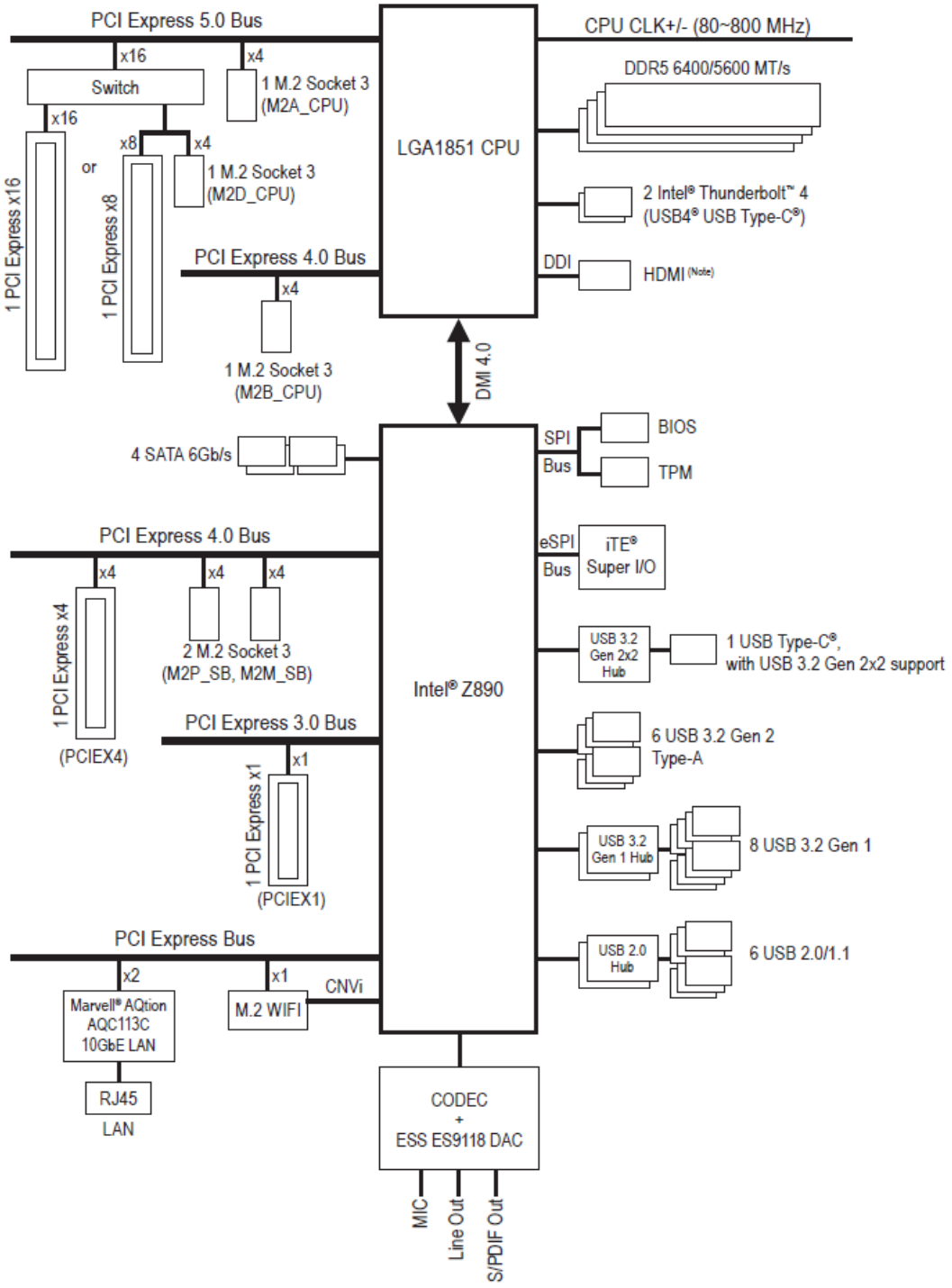
Kako pristupiti registrima naprave?

- običnim instrukcijama – kao i memoriji
  - npr. **`mov r0, [0xE800111C]`**
- posebnim instrukcijama
  - npr. **`in r0, 0xF8`**
- registri: podatkovni, statusni, upravljački
- stvarni sustavi su uglavnom složeniji od prethodne slike, tj. najčešće imaju više registara i memorijskih lokacija
- napravama se pristupa izravno (rijetko) ili preko drugog međusklopa / kontrolera (npr. preko PCI kontrolera, USB međusklopa, ...)

Organizacija naprava u računalu

- razne naprave su različito brze/spore
- da se izbjegne da brze naprave sporije rade zbog sporih naprava one se odvajaju na različite sabirnice, brze i spore
- mostovi povezuju sabirnice
- sve je to sklopovski izvedeno – programski izgledaju kao da su na istoj

Primjer matične ploče



Pristup napravama preko adresa

- najčešći način je korištenje adresa – adrese upravljača (kontrolera) ne same naprave – kontroleri to prosljeđuju napravi (npr. PCIe protokolom)
- adrese mogu biti i virtualne (kad se koristi straničenje)
- poseban oprez pri korištenju adresa jer procesor može koristiti optimizaciju
  - korištenje priručnog spremnika procesora
    - ne šalje odmah napravi već samo sprema u priručni spremnik
  - izmjena redoslijeda izvođenja susjednih operacija
    - out-of-order – kada zaključi da su susjedne instrukcije nepovezane
- nekad navedene optimizacije mogu smetati – izazvati krivi rad s napravom
  - tada to treba spriječiti ubacivanjem/korištenjem posebnih instrukcija

Korištenje međuspremnika

- razlozi korištenja međuspremnika:
  1. povećanje učinkovitosti
     - naprave i programi rade različitim brzinama
       - da se ne čekaju, „višak” podataka se pohranjuje u međuspremnik
       - zapravo sva komunikacija ide preko međuspremnika
         - npr. program → međuspremnik → naprava (i obratno preko drugog međ.)
  2. ponekad tek skup podataka čini cjelinu koju ima smisla prenositi
     - neke operacije se pokreću tek kad je primljena cjelina (blok, paket)

Korištenje međuspremnika

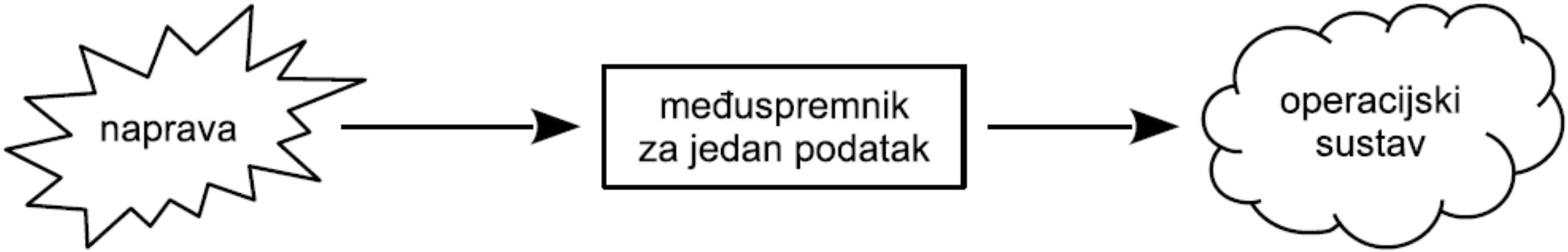
- međuspremnici mogu biti u memoriji (glavnoj) ali i u napravama
- međuspremnici u napravama mogu se koristiti
  - implicitno, nisu „vidljivi” sustavu, ili
  - mogu biti izloženi sustavu – može ih se koristiti kao memoriju, dohvatiti bilo koji dio
- međuspremnici u glavnoj memoriji
  - mogu biti puno većeg kapaciteta
  - zasebni za ulaz i izlaz

Tipovi međuspremnika

- međuspremnik za jedan podatak
- dvostruki međuspremnik (engl. *double buffer*)
- kružno korištenje međuspremnika

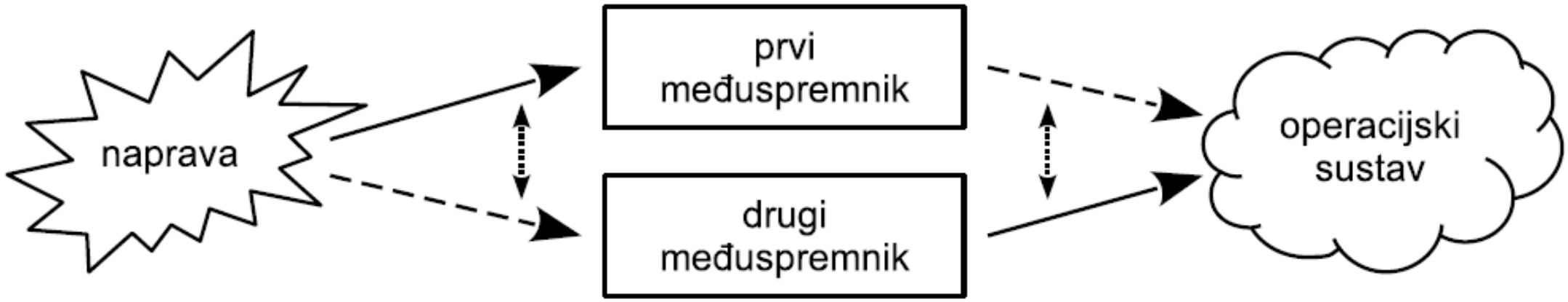
Međuspremnik za jedan podatak

- u njega stane samo jedan podatak (bajt, paket, blok)
- ulazna naprava upiše podatak u međuspremnik
- tek nakon toga OS može čitati taj podatak
- idući podatak naprava može upisivati tek nakon što OS pročita prethodni
- za izlaznu napravu je obrnuto: OS prvi upisuje, naprava čita nakon toga



Dvostruki međuspremnik

- istovremeno se mogu koristiti oba, ali nikada isti istovremeno; npr.
  - naprava upiše podatak u prvi, OS čita iz drugog (ako je tamo naprava prije nešto upisala)
  - kad naprava završi s pisanjem u prvi, može u drugi, ako je OS pročitao podatke koji su tamo; OS može tada pristupiti i čitati iz prvog



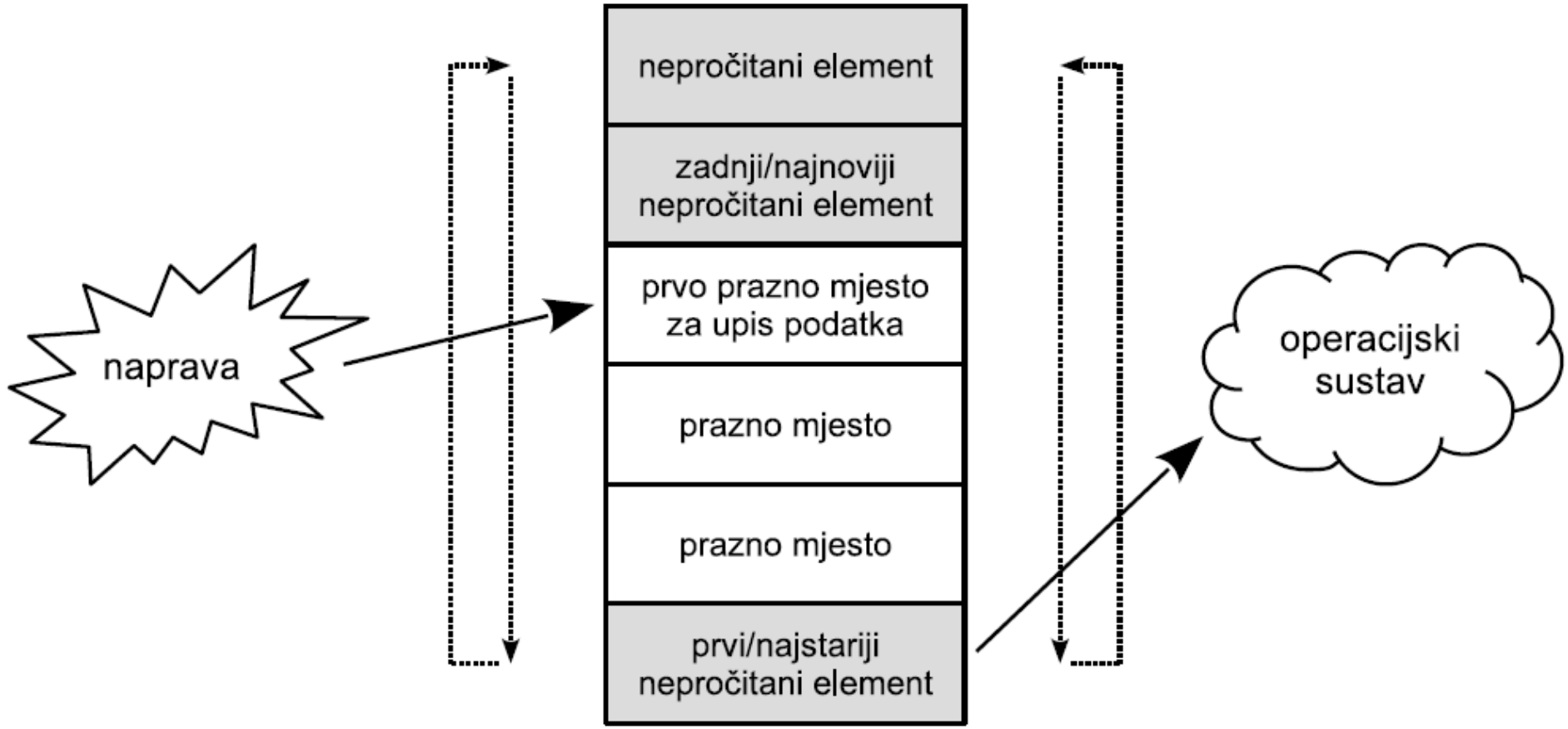
Dvostruki međuspremnik (3)

- primjer iz skripte

vrijeme	*OS*	*naprava*
t = 0 ms	*stavlja u M1*	*čeka OS*
t = 0 ms	*stavlja u M2*	*započinje sa čitanjem iz M1*
t = 0-5 ms	*čeka napravu*	*čita iz M1*
t = 5 ms	*čeka napravu*	*završava s M1*
t = 5 ms	*stavlja u M1*	*započinje sa čitanjem iz M2*
t = 5-10 ms	–	*čita iz M2*
t = 10 ms	–	*završava s M2*
t = 10 ms	–	*započinje sa čitanjem iz M1*
t = 10-15 ms	–	*čita iz M1*
t = 15 ms	–	*završava s M1*
t = 15 ms	–	*čeka OS*

Kružno korištenje međuspremnika (1)

- kružno korištenje je najčešće, nudi najveću fleksibilnost (efikasnost)



Kružno korištenje međuspremnika (2)

- pri pomicanju kazaljki koristi se operacija MOD

$$i = (i + 1) \textbf{ MOD } N$$

  - gdje je N veličina međuspremnika
- MOD (dijeljenje) je složena operacija i želi ju se izbjeći u jezgri
- međuspremnici su zato u jezgri često potencije broja 2: $N = 2^n$
- tada se operacija pojednostavnjuje:

$$i = (i + 1) \ \& \ (N - 1)$$

  - $N\text{-}1 = 11\ldots111_2$ (n jedinica)

  - npr. za N=256 vrijedi: A MOD 256 == A & 255

Kružni međuspremnik s dvostruko mapiranim dijelovima (1)

- problem kopiranja preko granice – trebaju dva kopiranja
  - jedno do granice
  - drugo od početka
  - npr.

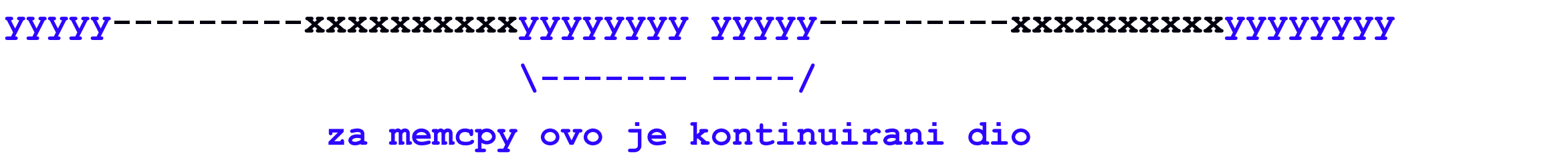
```
----------xxxxxxxxxx--------   (početno stanje)
----------xxxxxxxxxxyyyyyyyy   (prvo kopiranje)
yyyyy-----xxxxxxxxxxyyyyyyyy   (drugo kopiranje)
```

- kad se koristi straničenje može se kopiranje obaviti jednom operacijom

Kružni međuspremnik s dvostruko mapiranim dijelovima (2)

- međuspremnik se mapira i na susjednu lokaciju
  - napravno, međuspremnik mora biti višekratnik veličine stranica
- primjer dvostruko mapiranog međuspremnika:

- kopiranje u međuspremnik



Skica implementacije međuspremnika (1)

```
struct kms { /* kružni međuspremnik */
    void   *ms;       /* kazaljka na dvostruko mapirani međuspremnik */
    size_t  velicina; /* veličina rezervirana za ms */
    size_t  izlaz;    /* prvo nepročitano mjesto */
    size_t  ulaz;     /* prvo slobodno mjesto, ulaz >= izlaz uvijek! */
};
int inicijaliziraj_ms(struct kms *kms, size_t velicina) {
        /* velicina mora biti višekratnik veličine stranice */
        [...]
        x = nađi mjesta u adresnom prostoru procesa za 2 * velicina bajtova
        kms->ms = mmap(x, velicina, ...); /* zauzmi prostor za ms */

        /* mapiraj isti prostor na susjednim adresama */
        mmap(kms->ms + velicina, velicina, ...);
        kms->izlaz = 0;
        kms->ulaz = 0;
        kms->velicina = velicina;
        return 0;
}
```

Skica implementacije međuspremnika (2)

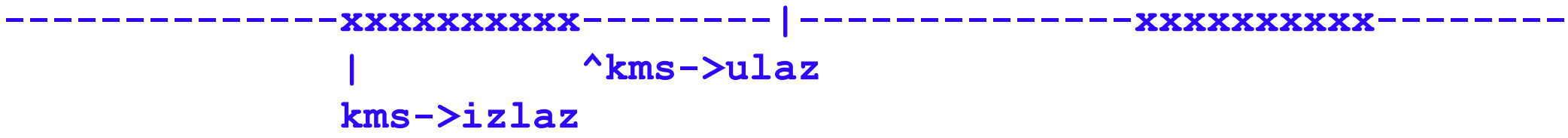
```
size_t stavi_u_ms(struct kms *kms, void *podaci, size_t za_staviti)
{
        size_t slobodno = kms->velicina - (kms->ulaz - kms->izlaz);
        if(slobodno < za_staviti)
                return 0; /* ili: za_staviti = slobodno */

        memcpy(&kms->ms[kms->ulaz], podaci, za_staviti);

        kms->ulaz += za_staviti;

        return za_staviti;
}
```

Skica implementacije međuspremnika (3)

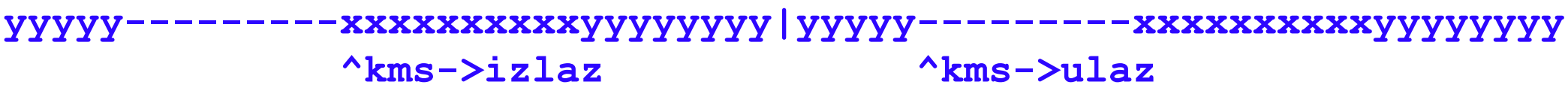
```
size_t uzmi_iz_ms(struct kms *kms, void *podaci, size_t za_uzeti)
{
        size_t ima = kms->ulaz - kms->izlaz;
        if(ima < za_uzeti)
                return 0; /* ili: za_uzeti = ima */

        memcpy(podaci, &kms->ms[kms->izlaz], za_uzeti);

        kms->izlaz += za_uzeti;
        if(kms->izlaz > kms->velicina) { //pomakni kazaljke natrag
                kms->izlaz -= kms->velicina;
                kms->ulaz -= kms->velicina;
        }

        return za_uzeti;
}
```

Primjer dvostruko mapiranog međuspremnika

a) početno stanje:

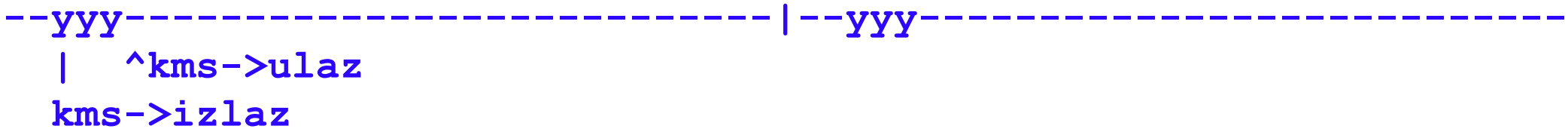


b) nakon dodavanja 13 elemenata:



c) nakon čitanja 20 bajtova
(međukorak)

(nakon ažuriranja kazaljki)



Međuspremnik i priručni spremnik

- koja je razlika?
- često ih poistovjećujemo, ali može se napraviti usporedba prema korištenju

Međuspremnik (engl. *buffer*)

- spremnik koji se koristi za prijenos podataka između dviju strana, npr. naprave i operacijskog sustava.
- pri prijenosu se podaci miču s one strane koja ih je kopirala u međuspremnik
- pri preuzimanju podataka iz međuspremnika, oni se miču iz njega
- podaci su samo na jednom mjestu (nema kopija podataka)

Priručni spremnik (engl. *cache*)

- priručni spremnik se koristi za ubrzanje rada
- kopija podataka (i instrukcija) se dohvaća da bude bliže onome tko ih koristi (procesor, naprava)
- postoji više kopija podataka
- upravljanje priručnim spremnikom je najčešće ostvareno sklopovljem (npr. priručni spremnik procesora)
- u slučaju korištenja naprava, moguće je “ručno” upravljanje – podaci se namjerno kopiraju iz međuspremnika naprave bez da se miču iz nje

Priručni spremnik (2)

- priručni spremnik podataka može ostvariti kôd jezgre, ali i programi, u prostoru procesa
  - pri operaciji **`read`** OS može dohvatiti više nego se tražilo funkcijom (npr. i susjedne blokove datoteke), ali procesu dati samo ono što je tražio
  - pri operaciji **`fread`** sama implementacija funkcije može od OS-a tražiti više te dobiveno držati u priručnom spremniku procesa koji je ona napravila; idući zahtjev možda može biti poslužen s tim podacima, bez da se traži "pomoć" OS-a – bez poziva jezgrine funkcije (`read`)

Usporenje zbog korištenja međuspremnika

- pretjerano kopiranje podataka može usporiti operacije
  - podaci se najprije kopiraju iz naprave u međuspremnik jezgre
  - potom u proces koji obrađuje te podatke
  - dvostruko kopiranje!
- izbjegavanje dvostrukog kopiranja
  - mapiranjem međuspremnika izravno u prostor procesa
  - podaci se izravno s naprave kopiraju u proces

04
Primjeri protokola za komunikaciju s napravama

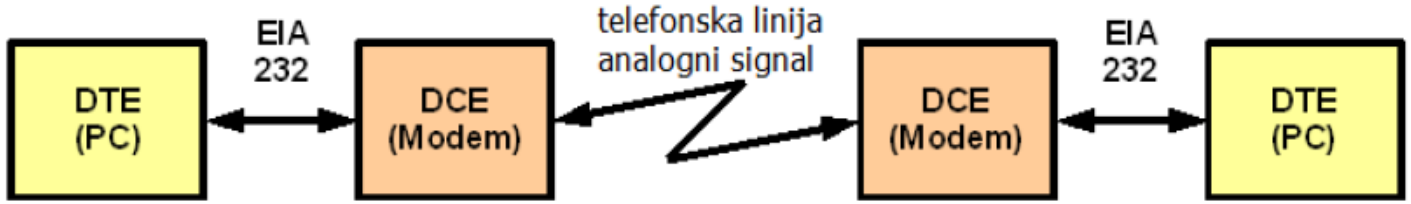
Serijska i paralelna veza, USB, PCI-e, SATA

Serijska i paralelna veza – ideje

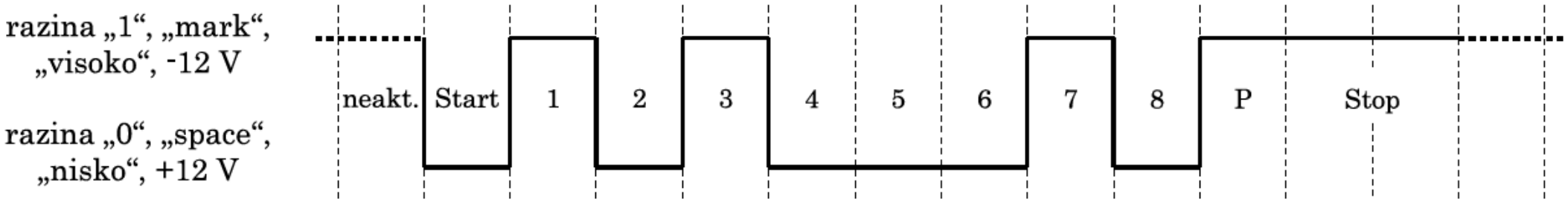
- Sinonimi u nastavku: žica = vodič = linija
- Serijska veza – osnovne ideje
  - prijenos informacije bit-po-bit preko jedne žice u jednom smjeru
  - po jednom ciklusu (taktu signala) prenosi se jedan bit
  - sinkronizacija/provjera ispravnosti nakon svih prenesenih bitova
  - manje žica, veće udaljenosti, manje brzine prijenosa (na prvi pogled)
- Paralelna veza
  - paralelni/istovremeni prijenos bitova preko više žica
  - po jednom ciklusu prenosi se više bitova
  - sinkronizacija nakon svakog ciklusa

Serijska veza – RS-232 (1)

- u prošlosti korištena za komunikaciju s modemom (i starim miševima)



- dvije žice za prijenos signala, po jedna za svaki smjer (TxD, RxD)
- dodatne žice za sinkronizaciju – kada koja strana smije slati (RTS, CTS)



Slika 4.1. Poruka 01000101_2 duljine 8 bita, parnog pariteta uz 2 stop bita

Serijska veza – RS-232 (2)

- podaci se šalju znak po znak (npr. 8 bita, ali može i manje 7-5)
  - u jednom ciklusu sata jedan bit
- kad se šalje više znakova – blok podataka/neka informacija, tada treba posebnim znakovima označiti početak i kraj (XON/XOFF/DLE)
- frekvencija sata, paritet, broj STOP bitova se moraju dogovoriti unaprijed
- npr. 9600 bauda/s (znakova u sekundi), 115200 bauda/s
  uobičajeni parametri: 8N1 – znak=8 bita, bez pariteta, 1 stop bit

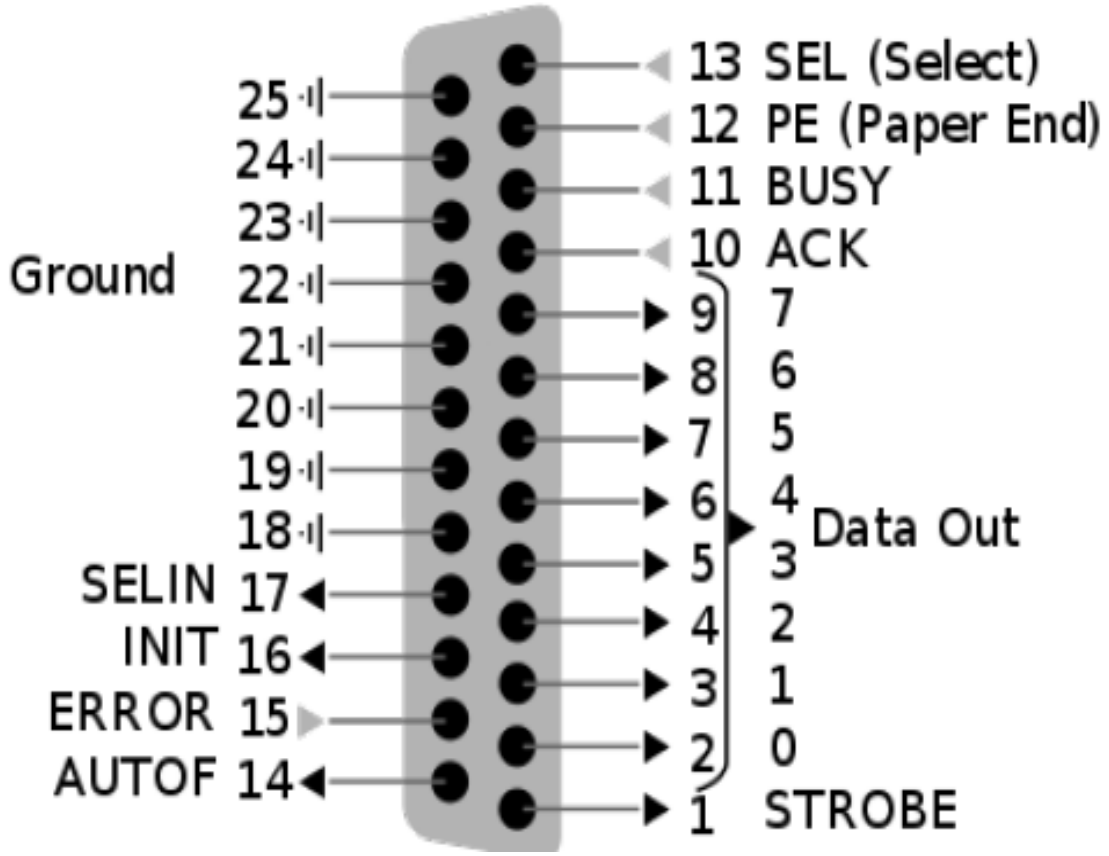
Serijska veza – RS-232 (3)

- svojstva serijske veze
  - za dvosmjernu komunikaciju su dovoljne četiri žice:
    TxD, RxD, CTS, RTS
  - robusna, ali spora

- modernije veze koje koriste serijski prijenos bitova (ali na napredniji način):
  - USB, PCIe, SATA

Paralelna veza (paralelni port, DB-25)

- osam bita ide paralelno, svaki preko svoje žice
  - u jednom ciklusu sata 8 bitova
- sinkronizacija nakon svakog ciklusa sata



Paralelna veza (paralelni port, DB-25) – svojstva

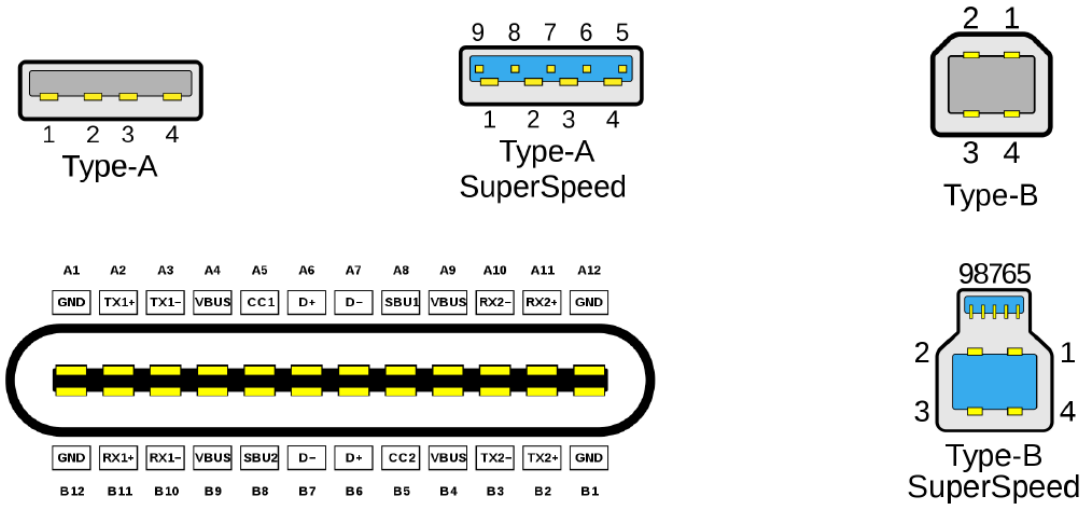
- iako idejno bi trebao biti brži prijenos, problem sinkronizacije nakon svakog ciklusa smanjuje performanse
- veliki broj žica stvara puno smetni – nije za veće udaljenosti

- osim paralelnog porta, paralelne veze su uključivale protokole
  - PCI – koji je danas zamijenjen PCIe (serijskim protokolom)
  - PATA – komunikacija s diskovima – danas se koristi SATA i sl. (M.2)

- paralelna veza danas praktički ostaje samo za procesor – memorija

USB – Universal Serial Bus

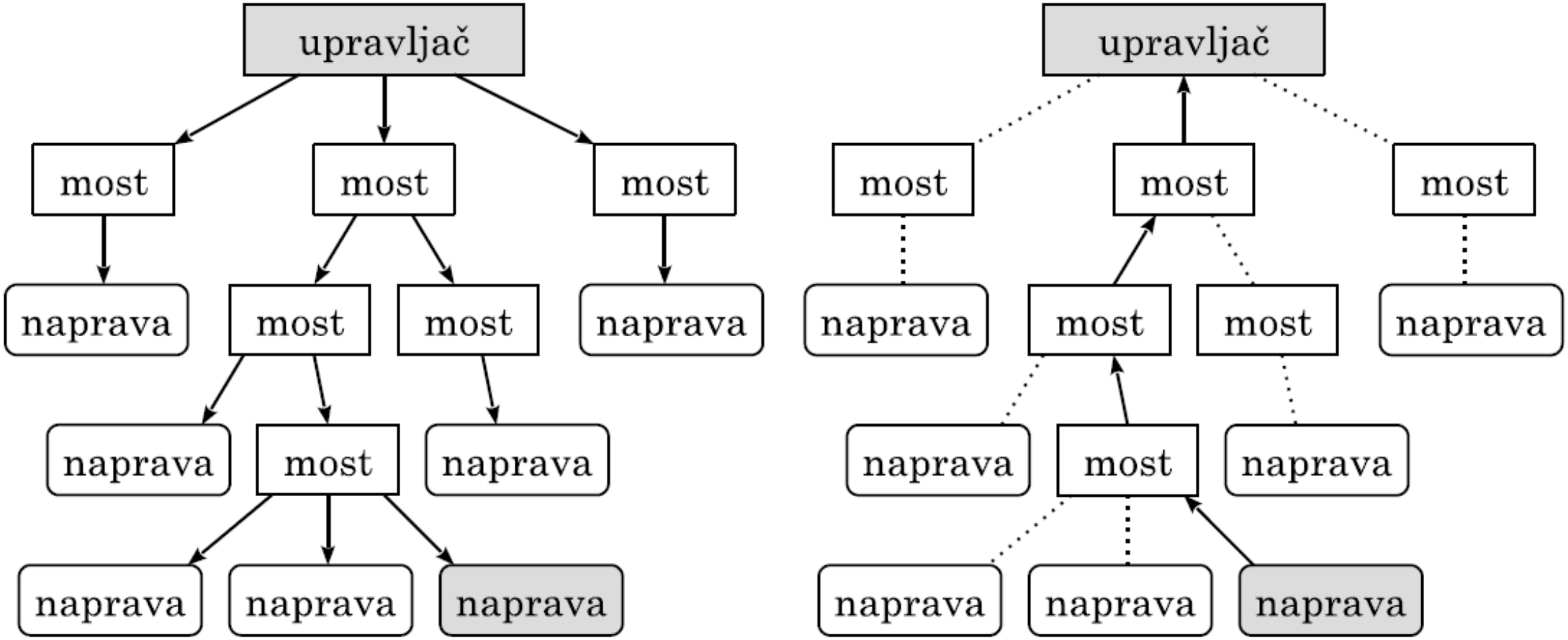
- osmišljen da zamijeni starije protokole (RS232 i paralelni port)
- osmišljen da bude jednostavan za ostvarenje u periferiji računala
  - tipkovnica, miš, pisač, memorijske kartice
- USB 1, 2, 3, 4; više različitih priključaka: Type-A/B, USB-C, ...



USB – „sabirnica”

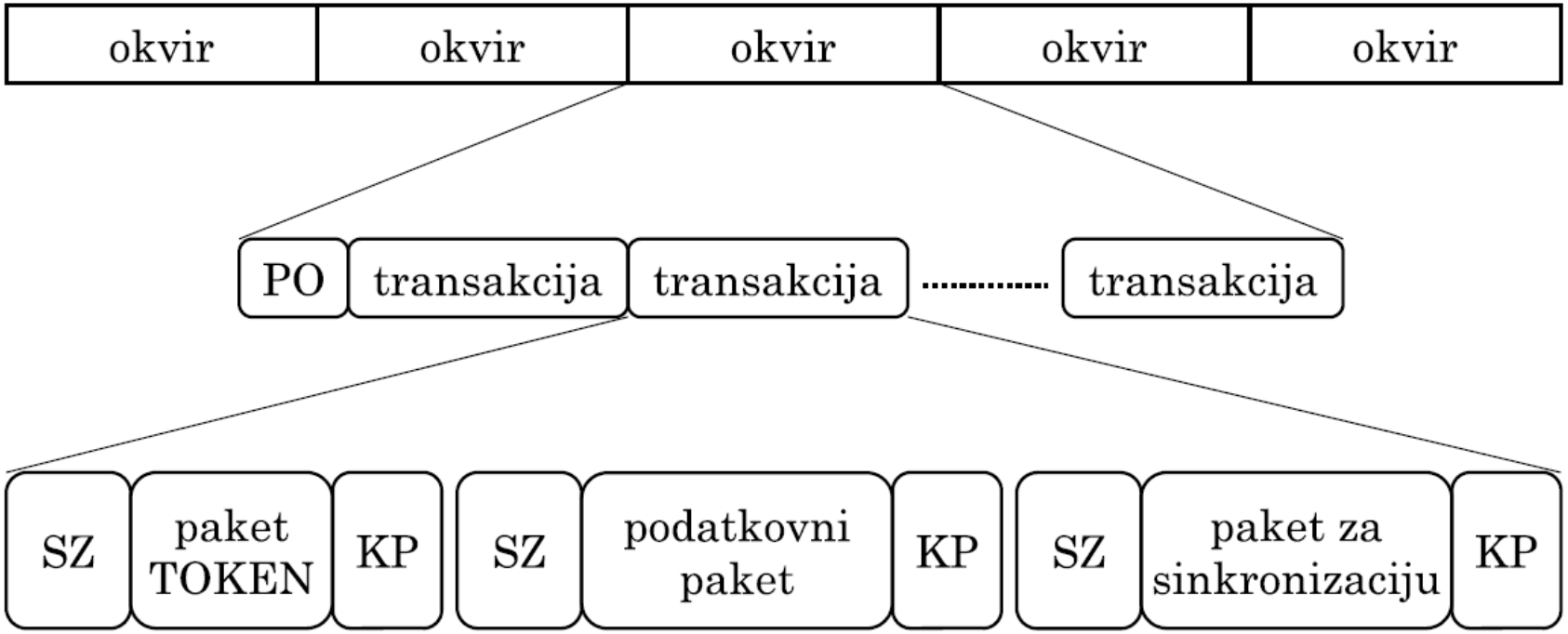
- logički je sabirnica, fizički stablo
- elementi stabla: jedan upravljač (host/root hub), mostovi (hub) te naprave
- upravljač inicira sve komunikacije – logički se sva komunikacija obavlja između upravljača i naprave (nema izravne komunikacije naprava-naprava)
- pri slanju upravljač → naprava
  - svi mostovi prosljeđuju primljeno dalje na sve svoje izlaze
- pri čitanju naprava → upravljač
  - podaci idu samo jednim putem prema upravljaču

USB – „sabirnica”



USB komunikacija (1)

- komunikacija je podijeljena u okvire → transakcije → pakete



- svaka naprava dobiva adresu: 7-bitovni broj (127 naprava max po uprav.)
- unutar svakog okvira upravljač „proziva naprave” (svake 1ms ili češće)

USB - paket

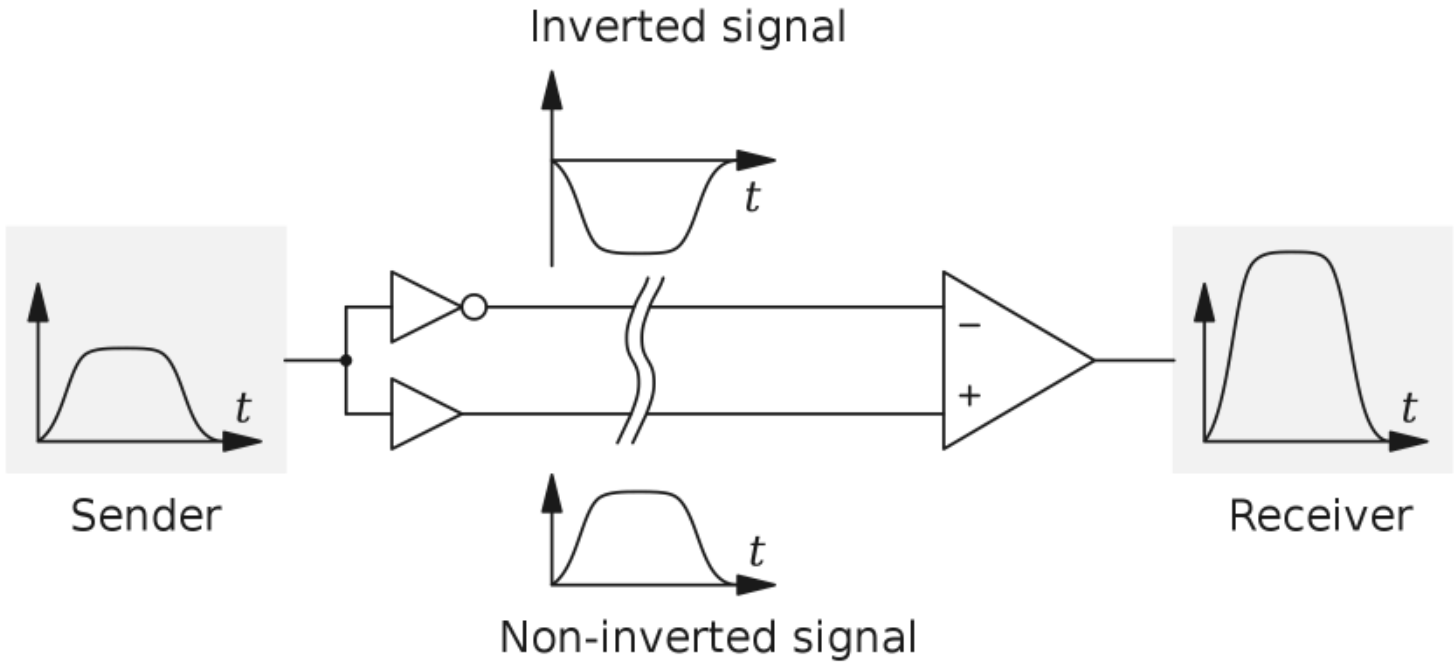
- SZ – sinkronizacijsko zaglavlje (fizička razina, ne nosi informacije)
- KP – kraj paketa, slično kao i SZ (vidi kasnije)
- uobičajena transakcija se sastoji od tri paketa:
  1. TOKEN
  2. podaci
  3. sinkronizacija (ACK)
- paket TOKEN (obično prvi u transakciji) definira
  - tip transakcije (IN, OUT, SETUP, PING, Start-of-frame)
  - adresu naprave s kojom želi komunicirati (7-bita adrese + funkcija)

USB – funkcije

- naprava može imati više „funkcija” (endpoint), svaka sa svojim brojem
  - do 32 po napravi, 16 ulaznih i 16 izlaznih
- svaka komunikacija ide prema nekoj funkciji (to je adresa u paketima)
- na višoj razini koristi se princip cjevovoda za prijenos podataka; tipovi
  - cjevovod za razmjenu poruka (dvosmjerni)
  - cjevovod za protok podataka (jednosmjerni); tipovi
    - izokroni prijenosi – garantira se propusnost, ali s mogućim gubicima podataka
    - prekidni prijenosi – za brze odgovore na događaje
    - veliki prijenosi – za veće količine podataka; ne garantira se propusnost ni kašnjenja; koriste se slobodni ciklusi na sabirnici

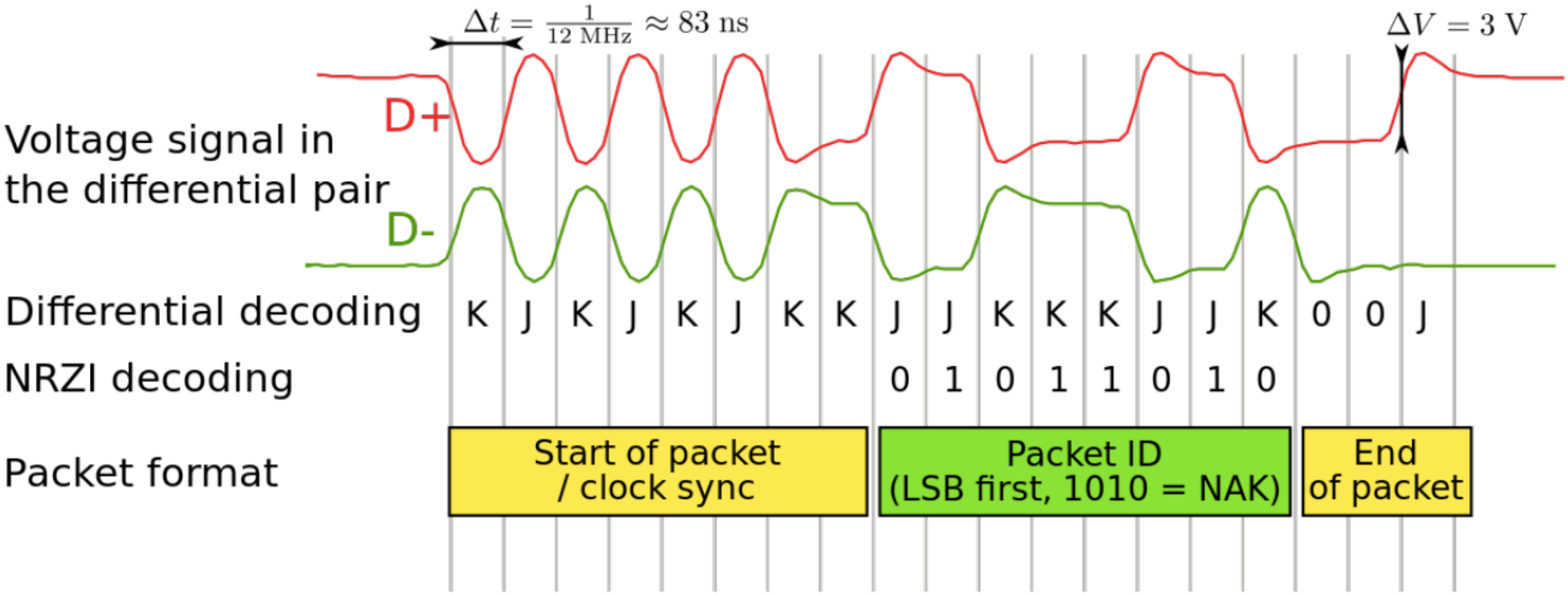
USB – fizička razina

- USB 1 i 2 koriste 4 žice: Vcc=5V, GND te D+/D- (USB 3, 4 više takvih)
  - podaci idu preko D+/D- ali samo u jednom smjeru istovremeno (half duplex)



- naponske razine ~0-3 V za USB1/2, ~0-400 mV za naprednije

USB – fizička razina – kodiranje signala



USB – fizička razina – kodiranje signala

- koristi se NRZI – non-return-to-zero-inverted
  - promjena signala u ciklusu: bit 0
  - bez promjene: bit 1
- svaki prijenos započinje sa sinkronizacijskim zaglavljem SZ
  - bitovi 00000001 – svaka nula je promjena – služi za sinkronizaciju primatelja
- ostatak prema već opisanome
- kraj paketa opet poseban
  - dva ciklusa: nisko-nisko oba vodiča
  - zadnji ciklus: visoko jedan, nisko drugi
- kodiranje: 8b/10b (USB2 128/130, USB4 64/66)
  - previše uzastopnih jedinica bi moglo desinkronizirati napravu

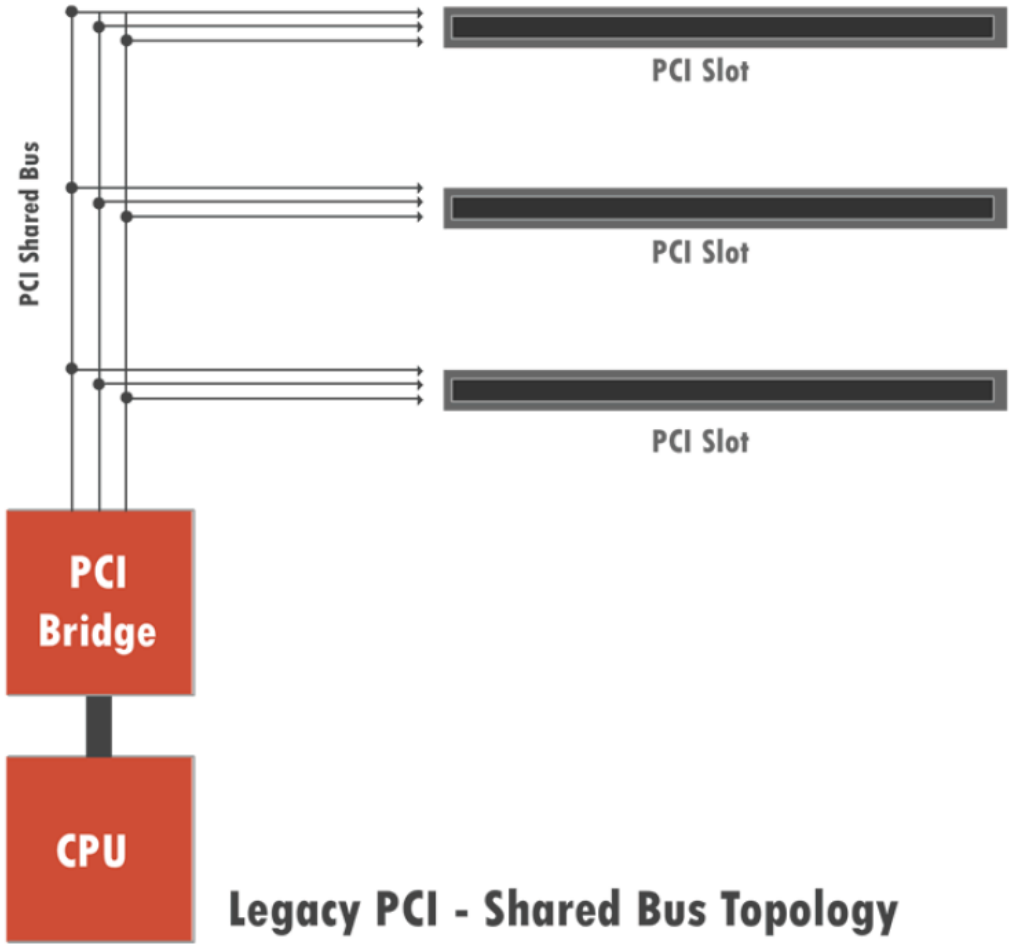
USB – tipovi naprava – podrška OS-a

- klase naprava omogućuju da nisu potrebni dodatni upravljački programi
  - za svaku klasu su definirani protokoli (poruke i sl.)
  - OS već zna raditi s raznim klasama naprava
- primjeri klasa
  - audio naprava, naprave za umrežavanje, naprava sučelja prema korisniku (miš, tipkovnica), pisač, podatkovna naprava, video, . . .
- brzine komunikacije:
  - niska (low): 1,5 Mbit/s
  - puna (full): 12 Mbit/s
  - velika (high): 480 Mbit/s
  - super (superspeed): 5 Gbit/s; super plus: 10 Gbita/s

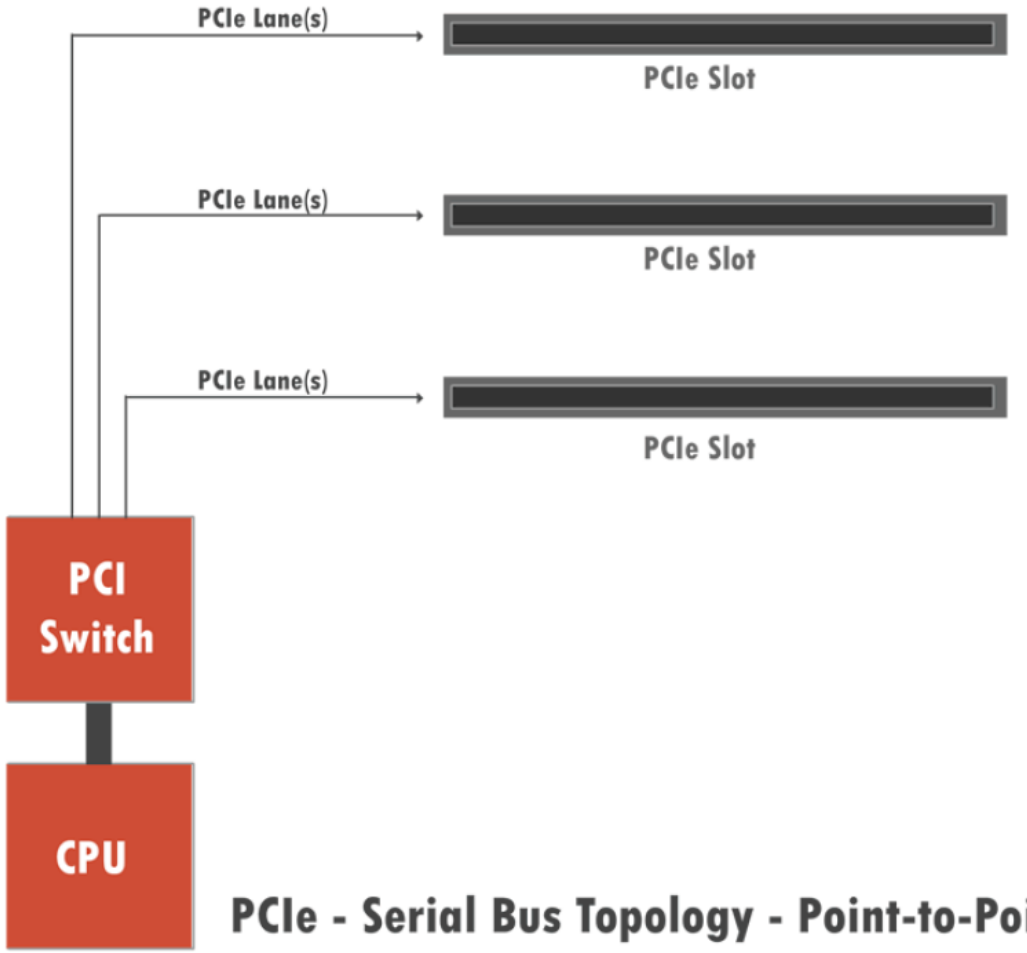
PCI Express (PCIe)

- PCI = Peripheral Component Interconnect
- zamjena za stariju PCI koja ima problema
  - zajednička sabirnica (usko grlo), potreban i arbiter
  - komunikacija samo u jednom smjeru istovremeno
  - previše signala, loše za sinkronizaciju, ograničava brzinu
  - zbog sinkronizacije takta, najsporija naprava diktira tempo
- PCIe:
  - izravna komunikacija između dviju strana (npr. naprave i PCIe kontrolera)
  - istovremena dvosmjerna komunikacija (engl. full duplex)
  - serijska komunikacija po pojedinim stazama (engl. lane)

PCI vs PCIe

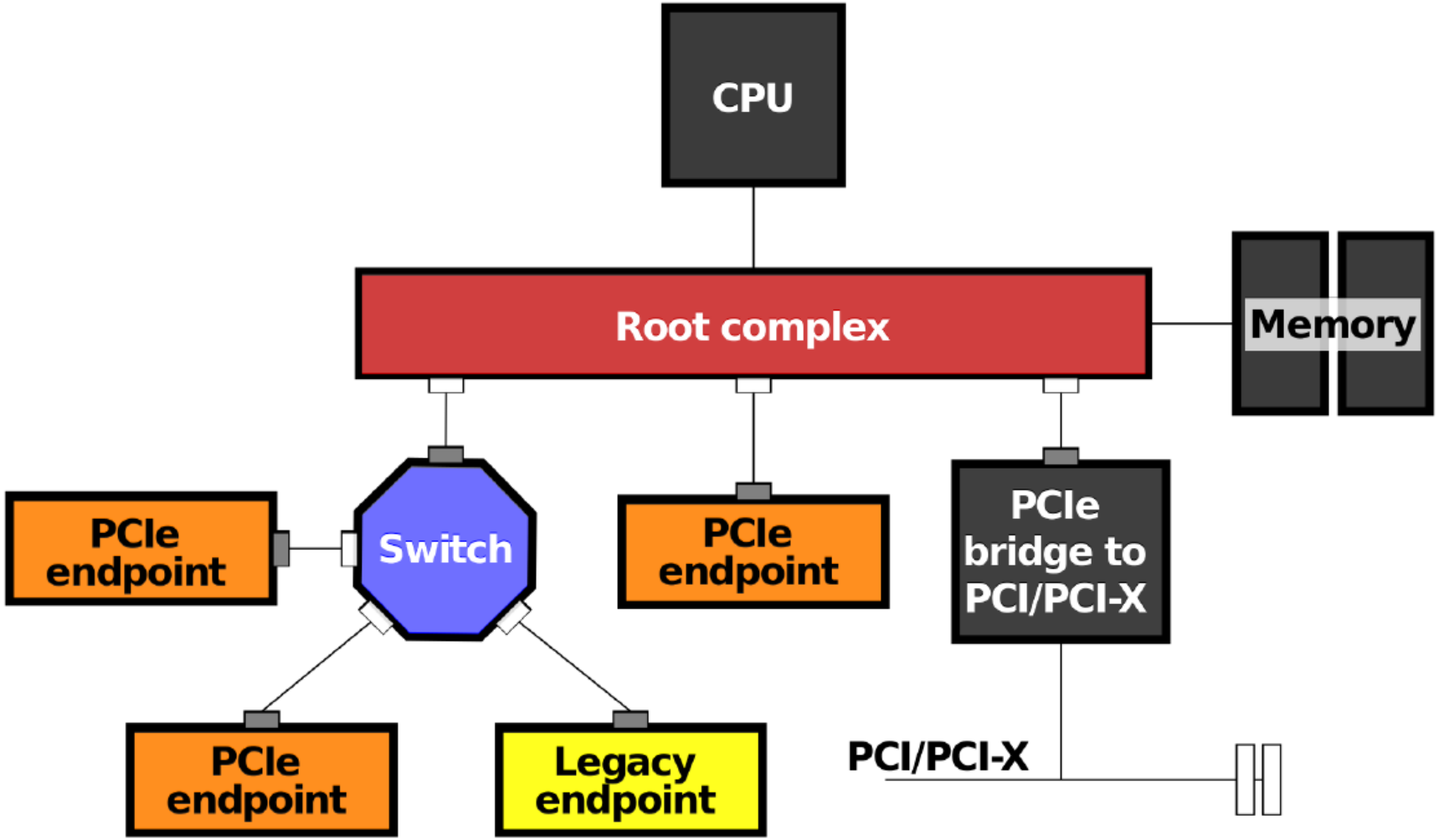


Legacy PCI - Shared Bus Topology

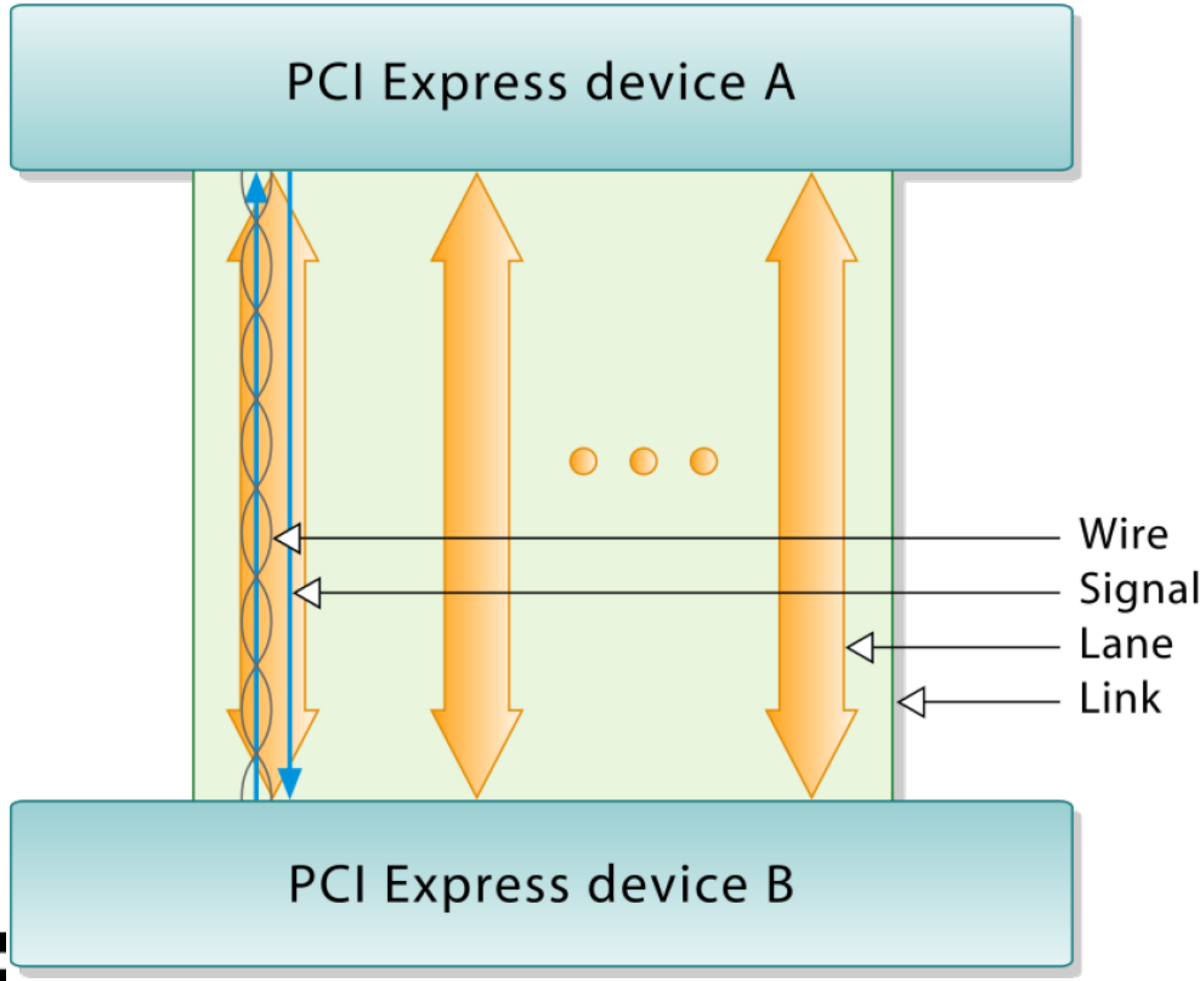


PCIe - Serial Bus Topology - Point-to-Point

PCIe složenije povezivanje



PCIe: veza (link), staza (lane)



- 4 žice za svaku stazu
  - dvije za svaki smjer
- broj staza 1, 2, ... 16 => oznake: x1, x2, ...
- podaci (paketi) koji se šalju se najprije podijele u dijelove koji se onda nezavisno (paralelno) šalju preko dostupnih staza

PCIe paketi

- razine:
  - PHY – fizička razina
  - DLLP – podatkovna
  - TLP – transakcijska
- svaka razina dodaje svoje dijelove/zaglavlja/podnožja
- svaki paket se potvrđuje ACK/NAK
- „kredit” – koliko može primiti podataka; da se izbjegne zagušenje

PCIe brzine, sadržaj

- za PCIe 3.0:
  - 8 GHz → 8 GT/s ~1 GB/s u jednom smjeru (2 GB/s dvosmjerno)
  - za 16 staza (x16), dvosmjerno ~32 GB/s

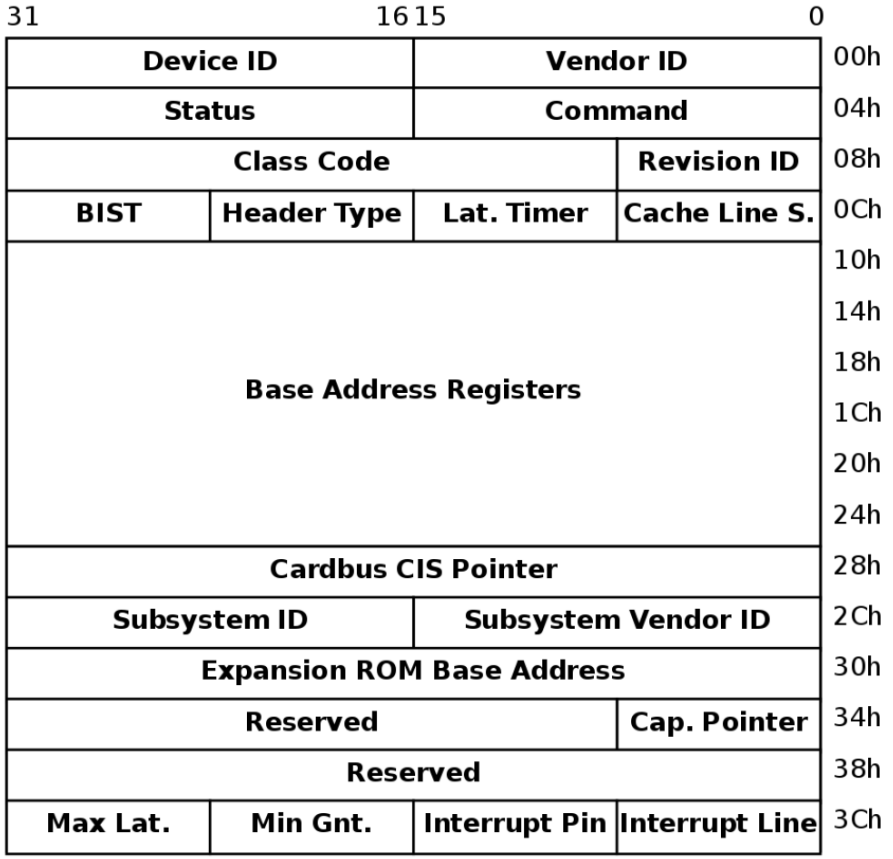
- preko PCIe se šalju PCI zahtjevi – taj dio se nije promijenio

PCI adrese, naredbe, inicijalizacija

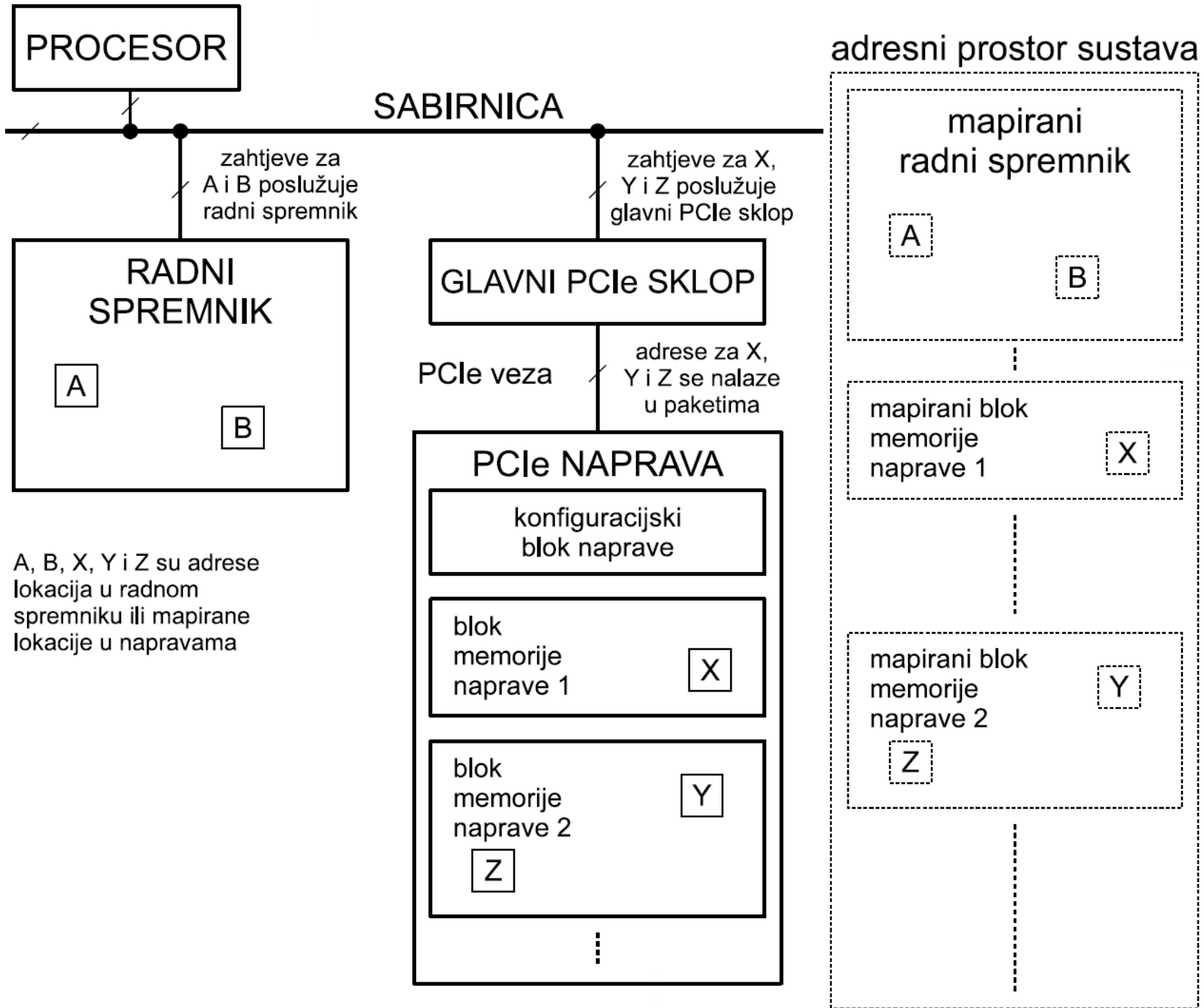
- adrese:
  - segment (PCIe samo) – 16 bita
  - PCI sabirnica – 8 bita
  - adresa naprave – 5 bitova
  - adresa funkcije – 3 bita
- inicijalizacija naprave – prozivanjem
  - na kojim će adresama biti vidljiva – to treba i napravi postaviti
- korištenje naprava – preko adresa
- naredbe: čitaj, piši, čitaj/piši u postavke (4 najbitnije)

PCI – konfiguracijski blok

- Polja zaglavlja konfiguracijskog bloka
  - Device ID, Vendor ID, Subsystem ID, Subsystem Vendor ID
    - da OS (ili BIOS) dozna o kakvoj se napravi radi
  - Status – za odgovor na podržane mogućnosti i dojavu greške
  - Base Address Registers (BAR)
    - adrese preko kojih je naprava dostupna

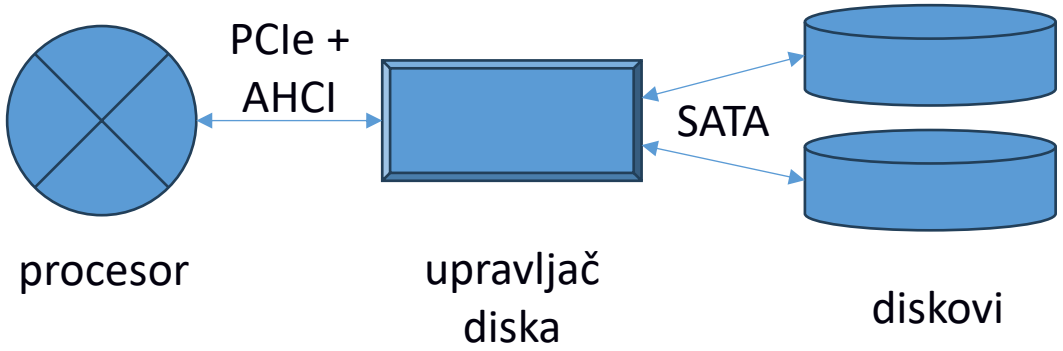


Korištenje preko adresa



Serial ATA - SATA

- Serial AT Attachment (AT=Advanced Technology)
- Slično kao i za PCIe/USB: serijska veza, dvije staze, za svaki smjer jedna
- SATA se koristi za komunikaciju prema disku od strane upravljača diska (disk controller)
  - s upravljačem diska komunicira se najčešće preko PCIe uz protokole
    - AHCI
    - NVMe
    - stariji ATA, PATA (IDE), ATAPI



AHCI

- Advanced Host Controller Interface
- half duplex – komunikacija samo u jednom smjeru istovremeno
- optimiran za rad s tvrdim diskovima
- definira način zadavanja operacija čitanja/pisanja, upravljanja, ...

NVM Express (NVMe)

- Non-Volatile Memory Host Controller Interface Specification (NVMHCIS)
- optimiran za SSD (taj tip memorije)
- mogućnosti za više redove zahtjeva s duljim nizom
- omogućuje full duplex

M.2

- SATA Express => PCIe, veće brzine
- umjesto osmišljavanja protokola za još veće brzine → iskoristiti PCIe koji to već nudi
- dodatna proširenja sučelja (USB 3) ...

SATA, M.2, svojstva

- jedinica podataka je blok (jedan ili više uzastopnih sektora)
- cijeli disk je polje blokova
  - geometrija skrivena od OS-a (glave, staze, sektori)
- zahtjevi: čitanje/pisanje skupine blokova
- DMA za prijenos – manje opterećenje procesora; on može nešto drugo raditi dok prijenos nije gotov (kad dobije prekidni signal DMA sklopa)
- brzine prijenosa
  - do 600 MB/s preko SATA priključka
  - nekoliko GB/s preko M.2 (PCIe)

Podrška za ostvarenje naprava u Linuxu

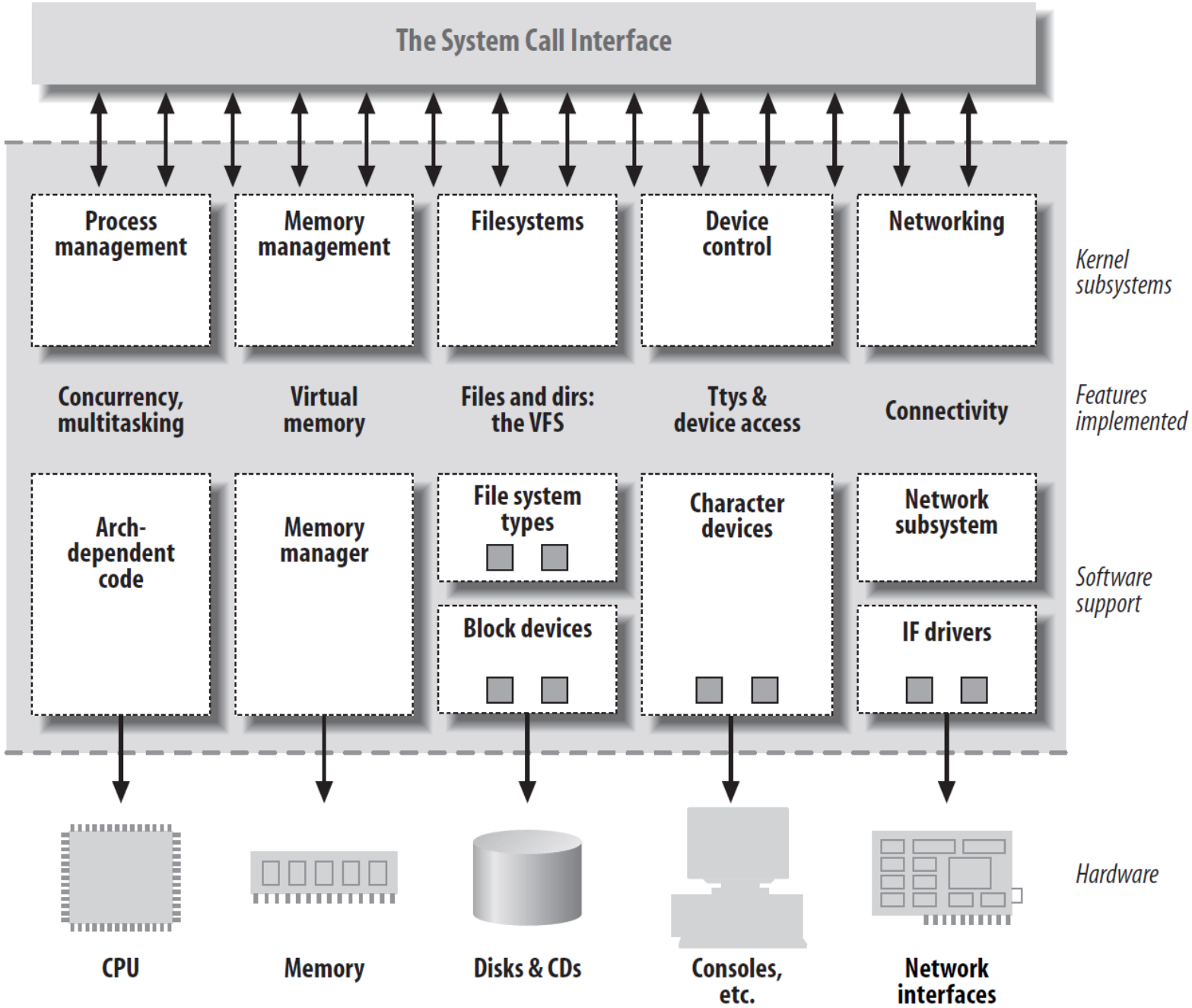
Konteksti izvođenja jezgrina koda

(Ne)dozvoljene operacije u jezgri

Moduli, upravljački programi

Složenost

- Linux, kao i svaki drugi netrivijalni OS je složen sustav
- Stoga je podijeljen na podsustave i slojeve
- Ovdje nije cilj svladati tu složenost, već samo pokazati neka svojstva jezgre potrebna za izgradnju upravljačkog programa
  - kako se izvodi jezgrin kod
    - ovisno kako je pozvan
  - što se smije i ne smije koristiti u kodu (ovisno o kontekstu)
  - prekidi
  - moduli, upravljački programi


The System Call Interface
Process management
Memory management
Filesystems
Device control
Networking
Kernel subsystems
Concurrency, multitasking
Virtual memory
Files and dirs: the VFS
Ttys & device access
Connectivity
Features implemented
Arch-dependent code
Memory manager
File system types
Block devices
Character devices
Network subsystem
IF drivers
Software support
CPU
Memory
Disks & CDs
Consoles, etc.
Network interfaces
Hardware
features implemented as modules

Izvođenje jezgrina koda

1. u jezgrinom kontekstu dretve nekog korisničkog procesa
   - dretva poziva jezgrinu funkciju

2. u kontekstu jezgrine dretve
   - jezgrin proces – posao jezgre, ne nužno vezan za neki proces

3. bez konteksta dretve – “atomarno”
   - obrada prekida, alarmi, taskler, softirq

Poziv jezgrine funkcije iz programa

- pri ulasku u jezgru (prekidom) prelazi se u **jezgrin kontekst procesa**
  - aktivira se **jezgrina dretva** koja izvodi jezgrinu funkciju
    - jezgrina dretva ima ista svojstva kao i obična (npr. isti prioritet, id, ...)
  - ona je “produžetak” korisničke, ali izvodi se kod jezgre, ne programa
  - u tom kontekstu dohvatljiva je jezgra i njene strukture podataka
  - podaci o dretvi i procesu su također jednostavno dohvatljivi,
    - npr. preko varijable `current` koja opisuje dretvu (`struct task_struct`)
- adresni prostor jezgre odvojen je od adresnog prostora procesa
  - za dohvat korisničkih podataka koristiti posebne funkcije
    - npr. `copy_to_user`, `copy_from_user`
- **jezgrina dretva može se blokirati** posebnim “unutarnjim” mehanizmima, sličnima “vanjskim” (semafori, monitori, ...)

Tablica 5.1. Primjer izvođenja programa u korisničkom i jezgrinom načinu rada

korisnički način (program)	jezgrin način (jezgrine funkcije)

`int fd, a, b;`
`char buf[BUFFER_SIZE];`
`ssize_t x, y;`
`fd = open("ime-datoteke", O_RDONLY);`	

	- pronalazak datoteke - provjera prava pristupa - stvaranje opisnika datoteke
`x = BUFFER_SIZE;` `y = read(fd, buf, x);`	
	- pronalazak opisnika u procesu - obavljanje operacije (može zahtijevati i čekanje) - kopiranje podataka u buf
`sscanf(buf, "%d %d", &a, &b);` `printf("a+b=%d\n", a + b);`	
	ispis niza znakova

Poziv jezgrine funkcije preko prekida naprava

- naprave izazivaju prekide – kako ih prihvatiti i obraditi?
- obrada se izvodi u jezgrinom načinu i nije povezana s prekinutom dretvom
- obrada se izvodi u atomarnom kontekstu – ne smije se blokirati
- poželjno je da takav atomaran način rada traje što kraće, da bi se mogli prihvaćati novi zahtjevi
- ako je potrebno više vremena za obradu, onda se posao obrade dijeli na dva dijela
  - neophodni dio – “top half” (Windowsi: Interrupt Service Routine)
    - izvodi se u atomarnom kontekstu – kod koji kreće s prihvatom prekida
  - dodatni dio – “bottom half” (Windowsi: Interrupt Service Thread)
    - poželjno izvoditi u ne-atomarnom kontekstu (jezgrina dretva)

Mogućnosti za dodatni dio obrade prekida u Linuxu

1. red poslova – “workqueue”
2. višedretvena obrada prekida – “threaded IRQs”
3. lagani prekid – “softirq”
4. zadačić – “tasklet”

- prva dva više/manje namijenjena za duže poslove obrade
  - izvode se u kontekstu jezgrinih dretvi – mogu biti blokirani
- 3. i 4. za kraće poslove – izvode se u atomarnom kontekstu
  - mogu biti prekidani drugim prekidima
  - ne mogu se sami blokirati u izvođenju – izvode posao do kraja
  - najčešće obavljaju poslove iz nekog reda, jedan za drugim

Dozvoljene i nedozvoljene operacije u jezgri

- atomaran kod (obrada prekida=top half, tasklet, softirq, alarmi)
  - ne smiju se koristiti funkcije koje mogu blokirati (npr. ne mutex_lock)
  - mogu funkcije s radnim čekanjem: **spinlock**
  - nije povezan s procesom pa se ne može (jednostavno) pristupiti nekom procesu
- kod s kontekstom procesa
  - sve interne funkcije, blokirajuće i neblokirajuće
  - može se pristupiti procesu i njegovom adresnom prostoru
- ne koristiti realne brojeve u jezgri
  - to bi zahtijevalo prošireni kontekst dretve

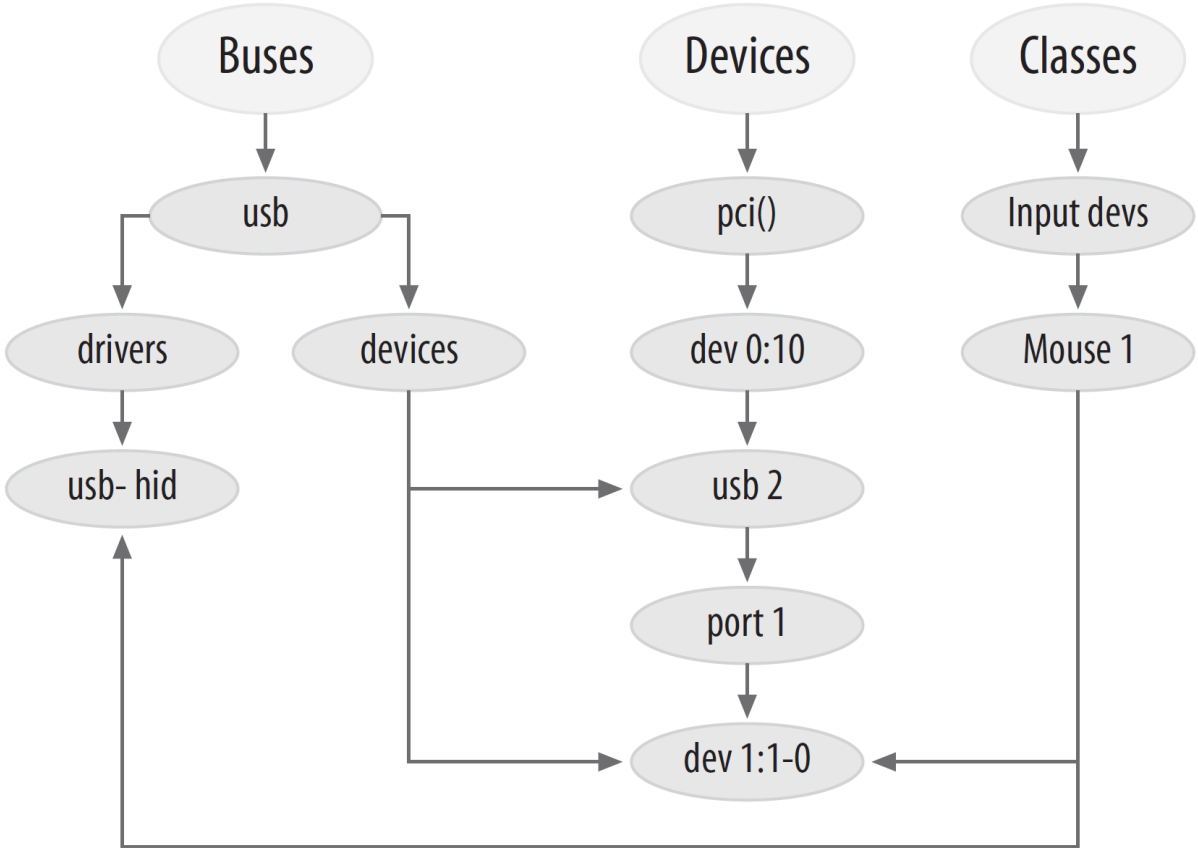
Moduli

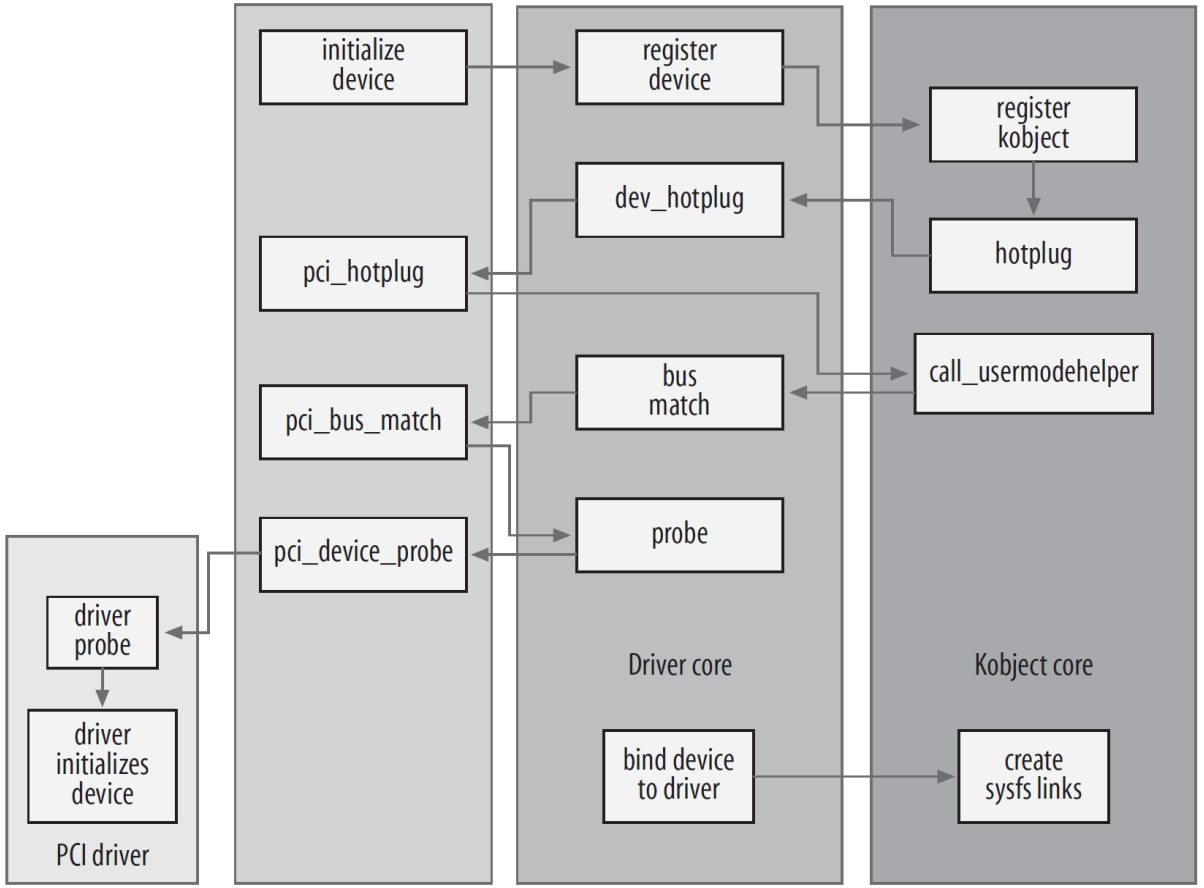
- upravljački programi za naprave
  - pripremaju se kao moduli
  - izvode se (u Linuxu) u jezgrinom način rada
  - mogu biti učitani
    - s jezgrom i cijelo vrijeme biti u sustavu (statički moduli)
    - na zahtjev (kao moduli); mogu se kasnije i maknuti iz sustava
- za pripremu modula treba koristiti i izvorni kod jezgre (samo zaglavlja)
- moduli se posebnim instrukcijama učitavaju i miču iz sustava
- više detalja o modulima u Linuxu u idućem poglavlju

Upravljački programi

- upravljaju stvarnim ili virtualnim napravama
- sastoje se od funkcija koje se pozivaju iz drugih funkcija jezgre
  - to su tzv. callback funkcije
- upravljački program treba registrirati svoje funkcije prilikom učitavanja
- postupak registracije ovisi o tipu naprave – Linux razlikuje:
  - znakovne naprave – te ćemo detaljnije u idućem poglavlju
  - blok naprave – za rad s diskom i sličnim spremištima datoteka
  - mrežna sučelja
- interna organizacija naprava u Linuxu je vrlo složena – ali se njome nećemo baviti

Primjeri interne organizacije i registracije naprave





Upravljački programi unutar Linuxa

- upravljački program „samo” popunjava neke dijelove potrebne za rad; ostalo je ostvareno kroz infrastrukturu Linuxa
- primjer lanca poziva od programa do funkcije upravljačkog programa
  - do_syscall
  - sys_write
  - ksys_write
  - vfs_write
  - shofer_write ← ovo je od upr. pr.