



# LTAT.06.001

## Operatsioonisüsteemid

Mäluhaldus II jätk.  
Võrk I

Artjom Lind

[artjom.lind@ut.ee](mailto:artjom.lind@ut.ee)

7.11.2025

# Plaan

- Mäluhaldus
  - Paging (leheküljetabelid)
    - Erinevad viisid
  - Segmentation
  - Kokkuvõtte
- Võrk
  - ...

# Objektifailide sidumine

Sidumine on objektifailide ja teekide ühendamise protsess käivitatava- või teegifaili loomiseks. See samm toimub pärast kompileerimist, mis muudab lähtekoodi objektifailideks. Linkimist on kahte peamist tüüpi: staatiline ja dünaamiline.

Staatiline sidumine: objektifailid ja teegid ühendatakse üheks käivitatavaks failiks. Tulemus on iseseisev, kuid võib olla suur.

Dünaamiline sidumine: käivitatav fail sisaldab viiteid välistele teekidele. Need lahendatakse käitusajal, vähendades faili suurust, kuid lisades sõltuvusi.

# Aadresside sidumine

Keelekonstruksioonide sidumine mäluaadressiga võib toimuda erinevatel etappidel:

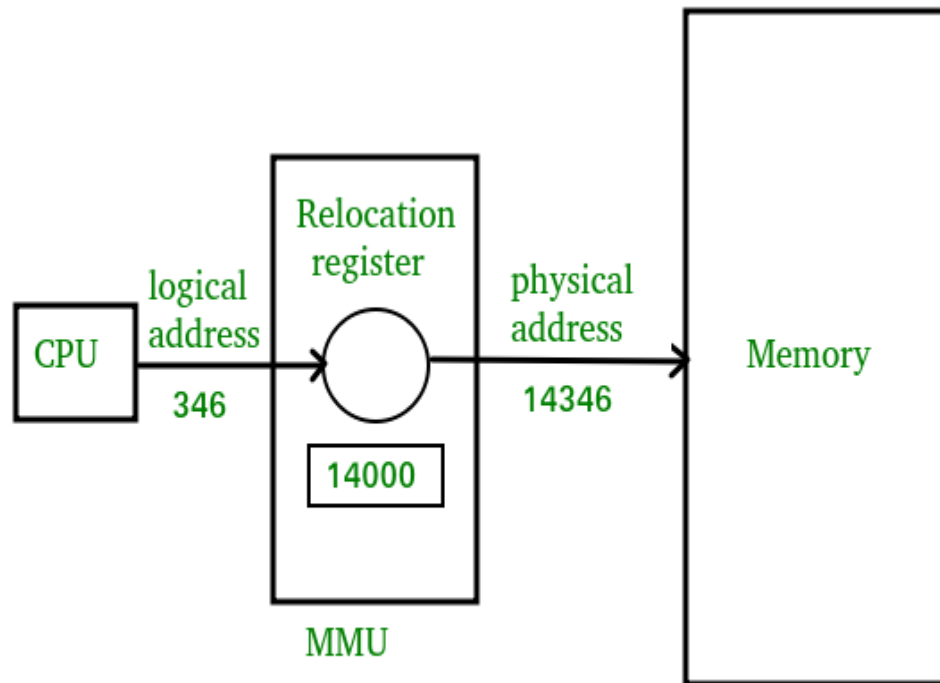
- Staatilise kompileerimise ajal — mälupaigutus on ette teada, genereeritakse kood absoluutaadressidega, paigutuse muutumisel on vaja ringi kompileerida
- Laadimise ajal, dünaamiliselt lingitud — genereeritakse mälus ümberpaigutatav kood, reaalsed mäluaadressid asendatakse programmi laadimisel
- Täitmise ajal, dünaamiliselt laetud — aadresside sidumine lükatakse edasi konkreetse aadressi poole pöördumiseni. Programmi saab täitmise ajal mälus ringi liigutada. Vajab riistvaralist tuge

# Loogilised ja füüsilised aadressid

- Loogilise ja füüsilise aadressi eraldamine on mäluhalduse üks olulisemaid kontseptsioone
  - Loogiline aadress — programmide poolt kasutatav aadressiruum, nimetatakse ka virtuaalseks aadressi ruumiks
  - Füüsiline aadress — aadress, mida mäluseade tegelikult näeb ja kasutab • Kompileerimise ja laadimise ajal seotavate aadresside puhul on loogiline ja füüsiline aadress samad, täitmise ajal seotavatel erinevad
- MMU (Memory Management Unit) — riistvaraline seade loogiliste aadresside füüsiliseks teisendamiseks
- Kasutajaprogramm tegeleb oma loogiliste aadressidega (0. . . max) ega näe otseselt füüsilisi aadresse

# Füüsiline aadressiruum

Viitab tegelikule süsteemi installitud RAM-ile. Kui protsessor pääseb juurde andmetele või juhiste, tõlgitakse see lõpuks asukohta füüsilises mälus. Füüsilist aadressiruumi haldab operatsioonisüsteem koos riistvarafunktsioonidega, nagu mäluhaldusüksus (MMU).



# Loogiline aadressiruum

Tuntud ka kui virtuaalne aadressiruum, see on abstraktsioonikiht programmi ja füüsilise mälu vahel. Iga süsteemis töötav protsess näeb oma külgnevat aadressiplokki, mis tavaliselt algab nullist. Operatsioonisüsteem ja MMU on selle loogilise aadressiruumi toovad vastavusse füüsiliste aadressidega.

Tõlge: MMU, sageli tõlkepuhvri (TLB) abiga, teisendab loogilised aadressid füüsilisteks aadressideks. Tavaliselt on see rakenduse jaoks läbipaistev.

# Dünaamiline linkimine

- Linkimine lükatakse edasi täitmise ajaks
- Tegelikult alamprogrammi asemel on proksi (stub), mis esimesel tema poole pöördumisel laadib päris alamprogrammi ja asendab ennast selle alamprogrammiga (enamasti viida vahetusega)
- Operatsioonisüsteemi tuge on vaja juhul, kui tahame neid alamprogramme jagada mitme protsessi vahel ja kasutame seejuures mälukaitset
- Eriti kasulik (süsteemsete) teekide kasutamisel — kettaruumi ja mälu kokkuhoid, mugav teekide uuendamine

# Dünaamiline linkimine

Laadimisaeg: teegid laaditakse ja lingitakse programmi käivitamise ajal.

OS-i hallatav: linkimisega tegeleb opsüsteemi laadija.

Aadressi sidumine: sümbolid lahendatakse ja aadressid seotakse laadimis- või käitusajal.

Paindlikkus: madalam kui dünaamiline laadimine, kuna programmi käivitamiseks peavad teegid olema kohal.

Kasutusjuhtumid: levinud üldotstarbelistes rakendustes koodi jagamiseks ja mälu säästmiseks.

# Dünaamiline laadimine

- Alamprogramme ei laadita enne, kui neid kord vaja läheb
- Mälukasutus on efektiivsem
- Kasulik näiteks siis, kui harva esinevate erijuhtude töötlemiseks on kokku palju koodi
- Ei vaja operatsioonisüsteemi tuge, realiseeritav täiesti programmi enda tasemel:
  - POSIX (dlfcn.h) - dlopen()
  - WinAPI (loaderapi.h) - LoadLibrary()

# Dünaamiline laadimine

Käitusaeg: teegid laaditakse töötavasse programmi nõudmisel.

Programmi hallatav: programm ise haldab laadimist, kasutades süsteemikutseid, nagu `dlopen()` või `LoadLibrary()`.

Aadressi sidumine: programm otsib selgelt sümboleid, tavaliselt nime järgi, ja aadressid seotakse käitusajal.

Paindlikkus: suurem kui dünaamiline linkimine, kuna mooduleid saab lennult laadida ja maha laadida.

Kasutusjuhtumid: kasulik pistikprogrammides, laiendustes ja olukordades, mis nõuavad suurt modulaarsust.

# Ülekate (overlay)

- Rakendusprogramm hoiab mälus ainult osa programmist ja andmetest, mida hetkel on vaja
- Kasutatakse, kui programm kokku on suurem kui talle antud mälu
- Opsüsteemilt tuge pole vaja, realiseeritav kasutaja tasemel
- Näide: DOS ja suured programmid
  - mälupiirangud, täpsemalt 640 KB tavamälu limiit
  - võimaldab programmidel olla olemasolevast mälust suuremad, jagades need väiksemateks osadeks, mida nimetatakse ülekateteks
  - vajalikud ülekatted laaditakse käitamise ajal mällu, vahetades vastavalt vajadusele sisse ja välja.

# Saalimine (swapping)

- Protsessi võib ajutiselt mälust välja salvestusruumi kirjutada ning hiljem tagasi mällu laadida
- Salvestusruum — kiire plokkeade, mis mahutab kõigi kasutajaprogrammide mälu kujutised; peab olema otsejuurdepääsuga suvalise mälu kujutise juurde
- Roll out, roll in — saalimise variant prioriteedil põhinevate planeerimisalgoritmide juurde: madalama prioriteediga protsess saalitakse välja kõrgema prioriteediga protsessi töö ajaks
- Enamuse saalimise ajast võtab andmete kopeerimine välisseadmele; aeg on lineaarselt sõltuv mälu hulgast
- Muudetud kujul on saalimine kasutusel ka tänapäeva süsteemides; näiteks Unix, Linux, Windows

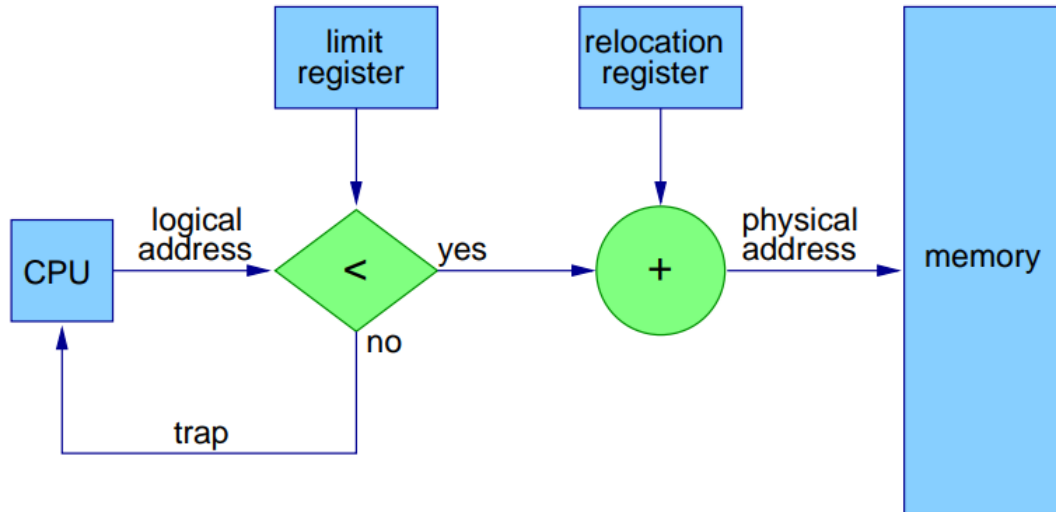
# Pidevate mälualade hõivamine (1)

- Põhimälu on enamasti jagatud kahte ossa:
  - Residentne opsüsteem (tihti madalamatel mäluaadressidel katkestusvektori asukoha tõttu)
  - Kasutajaprotsessid
- Kasutajaprotsessidele mõeldud mälu on omakorda jaotatud väiksemateks tükki
- Igale täidetavale protsessile eraldatakse üks sobiva suurusega terviklik mälutükk
- Auk — vaba mälutükk (uued protsessid saavad mälu nendest)

# Pidevate mälu alade hõivamine (2)

Iga protsessiga on seotud teisaldus- ja piiriregister

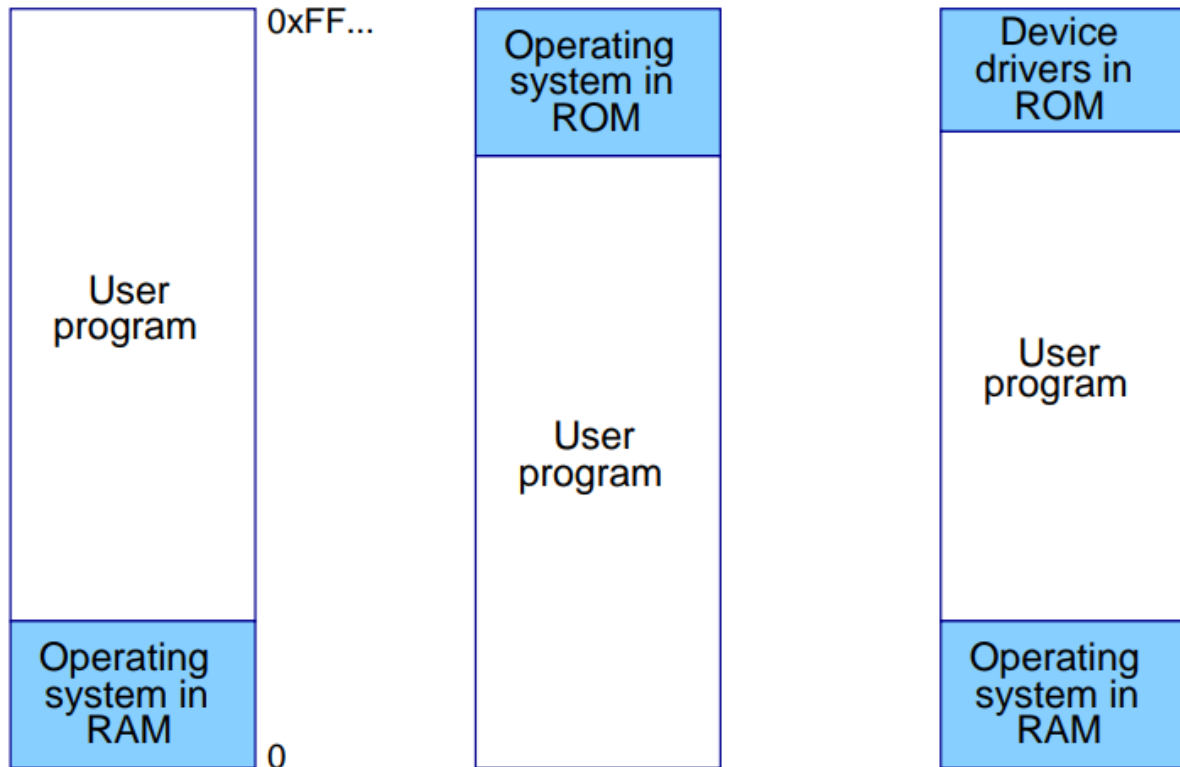
- Kasutatakse loogiliste aadresside teisendamisel
- Kaitseb protsesse ja opsüsteemi vigaste mälupöördumiste ees



## Pidevate mäluualade hõivamine (3)

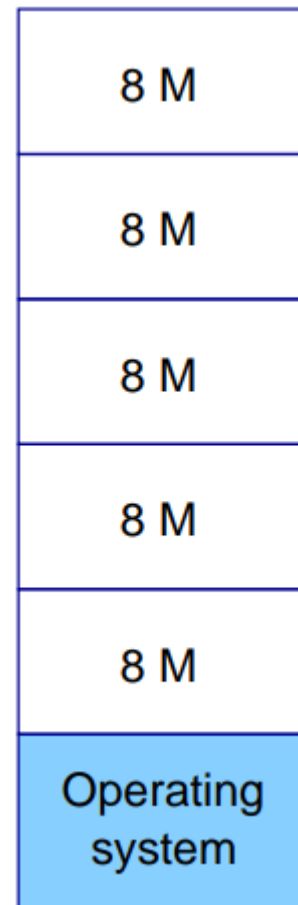
- Kasutajaprotsessidele mõeldud mälu tükkeks jagamine võib toimuda staatiliselt või dünaamiliselt
- Staatilise tükelduse korral on tükide arv ja suurus eelnevalt fikseeritud:
  - kõik tükid võrdse suurusega;
  - erineva suurusega tükid
- Erijuht — monoprogrammeerimine, kus korraga täidetak ainult üks protsess, millele antakse kogu opsüsteemist üle jääv mälu

# Mälujaotus monoprogrammeerimise süsteemides



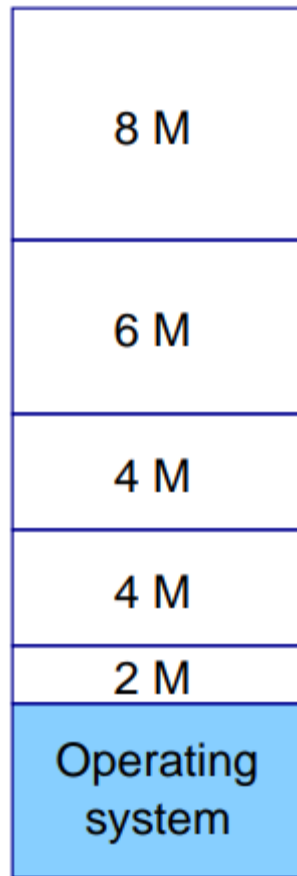
# Fikseeritud tükeldusega mälujaotus (1)

- Võrdse suurusega tükid:
  - Protsess, mille suurus ei ületa tüki suurst, laetakse suvalisse vabasse tükki
  - Kui kõik tükid on hõivatud, võib opsüsteem mõne protsessi välja saalida
  - Kui programm ei mahu tükki ära, peab programmeerija kasutama ülekatmist
- Lihtne realiseerida, kuid mälukasutus on ebaefektiivne
  - Sisemine fragmenteerumine — ükskõik kui väike ka protsess ei oleks, võtab ta enda alla terve tüki



# Fikseeritud tükeldusega mälujaotus (2)

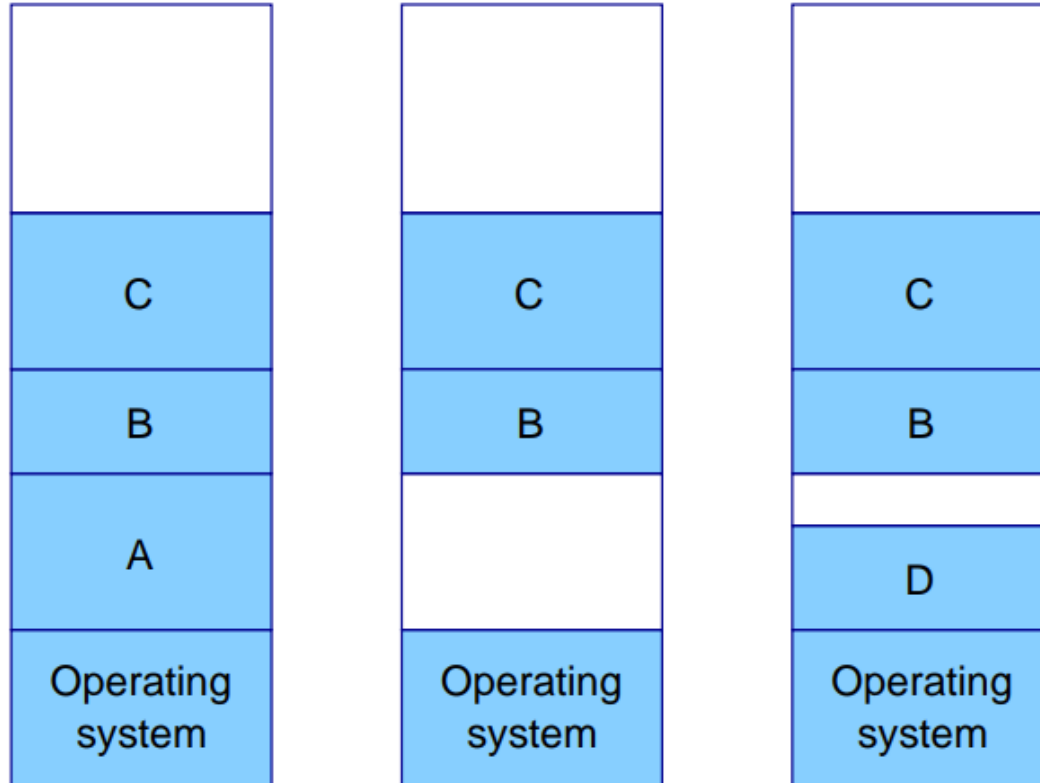
- Erineva suurusega tükid:
  - Protsessile antakse vähima suurusega tükk, millesse ta ära mahub
- Igal tükil (või sama suurusega tükkidel) üks eraldi protsesside järjekord:
  - Vähendab sisemist fragmenteerumist
  - Ebaefektiivne mitmetegumilisus
- Üks globaalne sisend järjekord:
  - Valida esimene protsess, mis auku mahub
  - Valida suurim protsess, mis auku ära mahub
  - Teisel juhul näljutamise oht



# Dünaamilise tükeldusega mälujaotus (1)

- Tükkide arv ja suurus ei ole eelnevalt fikseeritud, vaid muutub vastavalt täidetavate protsesside arvule ja suurusele
- Protsessile antakse täpselt nii suur tükk, kui ta vajab
- Protsessi lõpetamisel või välja saalimisel vabanev tükk ühendatakse võimalusel ümbritsevate aukudega suuremaks auguks
- Kalduvus tekitada kasutuses olevate tükkide vahele väikseid auke
- Väline fragmenteerumine — vajalik vaba mälu leidub, aga ta ei ole sidus

## Dünaamilise tükeldusega mälujaotus (2)



# Dünaamilise tükeldusega mälujaotus (3)

- Probleem: kuidas rahuldada päringut mäloplokile pikkusega  $n$  etteantud aukude nimekirjast?
  - Esimene sobiv (first-fit) — kasutatakse esimest piisava suurusega auku. Esimest võib lugeda algusest või eelmisest leitud august alates
  - Parim sobiv (best-fit) — kasutatakse vähimat piisava suurusega auku. Tuleb läbi otsida kogu nimekiri, enamasti on see suuruse järgi järjestatud \* Vähimad üle jäävad tükid
  - Halvim sobiv (worst-fit) — kasutatakse suurimat vaba auku, samuti vaja kogu nimekiri läbi vaadata \* Suurimad üle jäävad tükid
- Esimene sobiv ja parim sobiv annavad parema kiiruse ja mälu kasutuse kui halvim sobiv

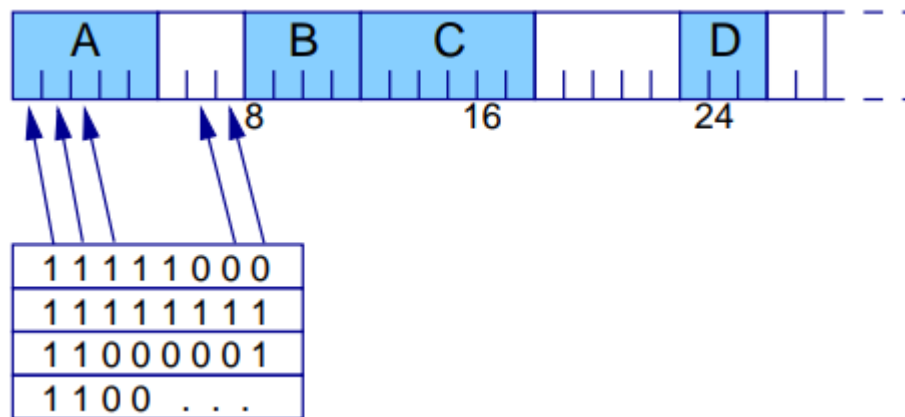
# Fragmenteerumine ja tihendamine

Välist fragmenteerumist saab vähendada mälu tihendamise (compaction) abil

- Tõstame hõivatud mälualasid ümber, näiteks kõik ühte otsa kokku
  - Näiteks liigutame protsessid kõik mälu algusse üksteise järel
- Võimalik ainult dünaamilise ümberpaigutamise korral
- Pooleli olev I/O segab (kopeerime OS puhvritesse või jätame protsessi paigale)
- Suhteliselt kulukas

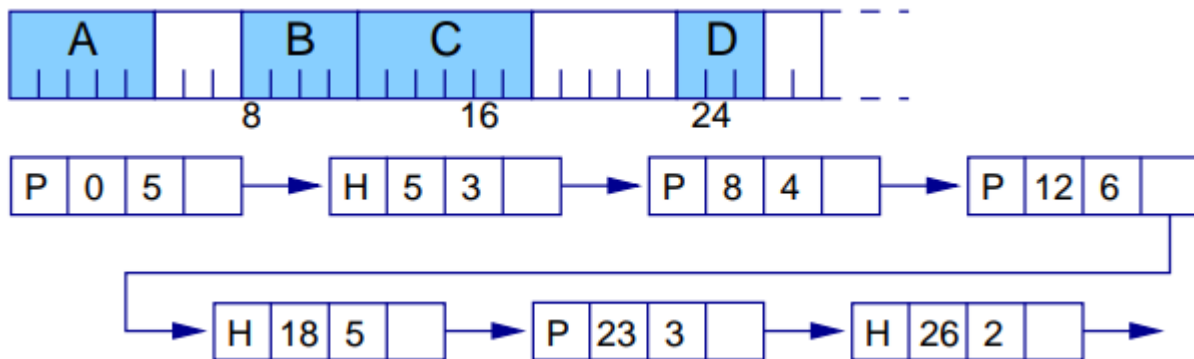
# Mäluhaldus bititabelitega

- Süsteem peab pidama arvet hõivatud ja vabade mälupiirkondade üle
- Lihtsaim meetod on kasutada bititabelit
- Kõrvuti olevate aukude ühendamise on automaatne
- Piisava suurusega vaba piirkonna leidmine on aeglane



# Mäluhaldus listidega (1)

- Iga tüki jaoks peame meeles tema staatuse, alguse ja pikkuse
- Vastavat infot hoiame dünaamilises listis (P — process, H — hole)
- Kõrvuti olevate aukude ühendamise on lihtne
- Piisava suurusega vaba piirkonna leidmine on kiirem kui bititabeli korral, kuid siiski lineaarse keerukusega



# Mäluhaldus listidega (2)

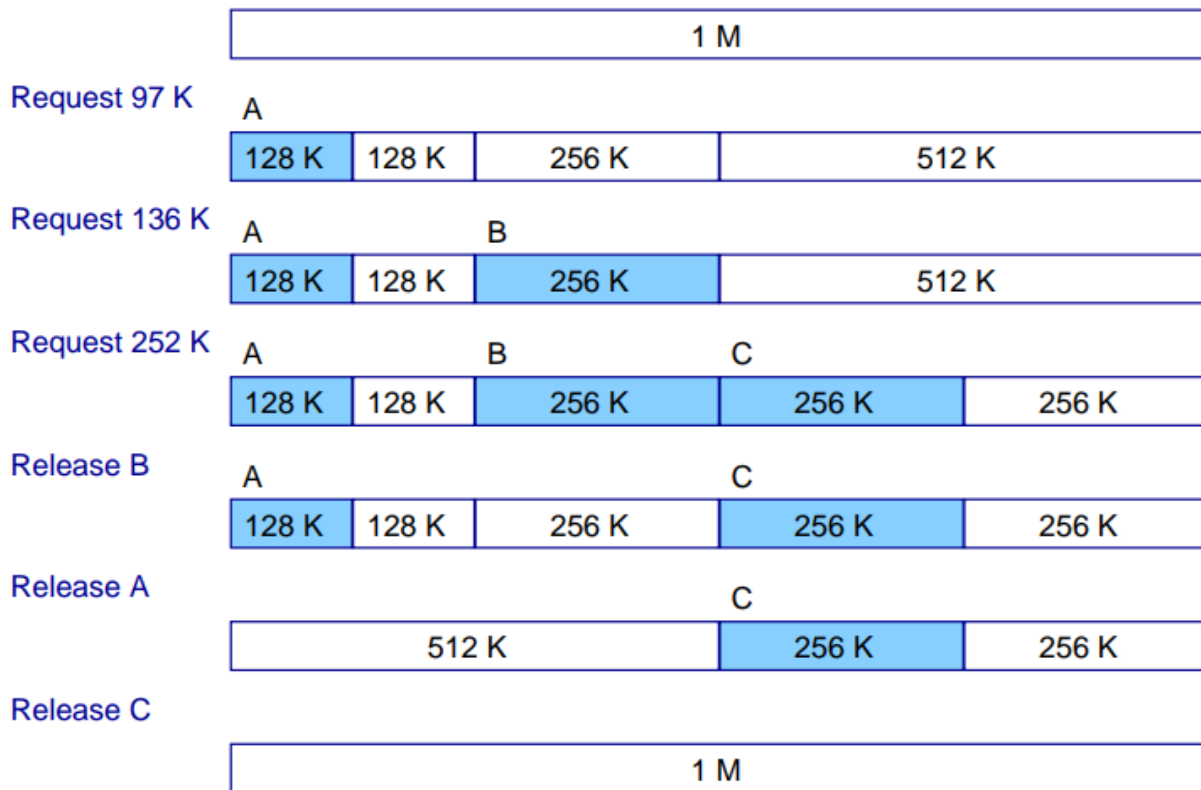
Variatsioonid:

- Haldame eraldi listi vabade ja hõivatud tükide kohta
  - Piisava suurusega vaba tüki leidmiseks pole vaja enam hõivatud tükke inspekteerida
  - Pole enam vajadust staatusbiti (P/H) järele
  - Vabade tükide listi jaoks pole eraldi mälu vaja, kuna neid tükke saab ise selleks ära kasutada!
  - Kõrvuti olevate aukude ühendamise on suhteliselt kulukas
- Haldame sagedasti esinevate tükisuuruste jaoks eraldi vabade tükide listi
  - Piisava suurusega tüki leidmine kiire

# "Buddy" süsteem

- Kogu olemasolevat mälu käsitletakse kui üht mälutükki suurusega  $2U$
- Kui küsitava mälu suurus on  $2U-1 \leq n \leq 2U$ , siis hõivatakse kogu tükk
- Vastasel korral jaotatakse tükk kaheks võrdseks osaks ja kogu protsessi korratakse kuni sobiva tüki tekkimiseni
- Tüki vabastamisel kontrollitakse kas "kaastükk" on vaba
  - Kui ei ole, siis lisatakse tükk antud suurusega vabade tükide listi
  - Vastasel korral tükid ühendatakse ning tekkinud tükki püütakse omakorda ühendada oma "kaastükiga"

# "Buddy" süsteemi näide



# Lehekülgede saalimine (paging)

- Protsessi füüsiline aadressiruum ei pea tingimata olema pidev — protsessi eri osad võivad asuda füüsilises mälus suvaliste kohtade peal laiali
- Jagame füüsilise mälu fikseeritud suurusega lehekülgedeks (kaadriteks — frame), suurus kahe aste (näit. 4096, 8192, 65536)
- Jagame loogilise mälu samasugusteks lehekülgedeks (page)
- Peame arvet vabade kaadrite üle
- Suurusega  $n$  protsessi jaoks otsime lihtsalt  $n$  vaba kaadrit
- Iga protsessi kohta teeme leheküljetabeli loogiliste lehekülgede ja füüsiliste kaadrite vastavusse seadmiseks
- NB! Lehekülgede kaupa hõivamisel tekib paratamatult sisemine fragmenteerumine

# Mälu eraldamine lehekülgede kaupa

0		0	$A_0$	0	$A_0$	0	$A_0$	0	$A_0$	0	$A_0$
1		1	$A_1$	1	$A_1$	1	$A_1$	1	$A_1$	1	$A_1$
2		2	$A_2$	2	$A_2$	2	$A_2$	2	$A_2$	2	$A_2$
3		3	$A_3$	3	$A_3$	3	$A_3$	3	$A_3$	3	$A_3$
4		4	$A_4$	4	$A_4$	4	$A_4$	4	$A_4$	4	$A_4$
5		5	$A_5$	5	$A_5$	5	$A_5$	5	$A_5$	5	$A_5$
6		6		6	$B_0$	6	$B_0$	6		6	$D_0$
7		7		7	$B_1$	7	$B_1$	7		7	$D_1$
8		8		8	$B_2$	8	$B_2$	8		8	$D_2$
9		9		9		9	$C_0$	9	$C_0$	9	$C_0$
10		10		10		10	$C_1$	10	$C_1$	10	$C_1$
11		11		11		11	$C_2$	11	$C_2$	11	$C_2$
12		12		12		12	$C_3$	12	$C_3$	12	$C_3$
13		13		13		13		13		13	$D_3$
14		14		14		14		14		14	$D_4$
15		15		15		15		15		15	

Page tables

**A**

0	0
1	1
2	2
3	3
4	4
5	5

**B**

0	6
1	7
2	8

**C**

0	9
1	10
2	11
3	12

**D**

0	6
1	7
2	8
3	13
4	14

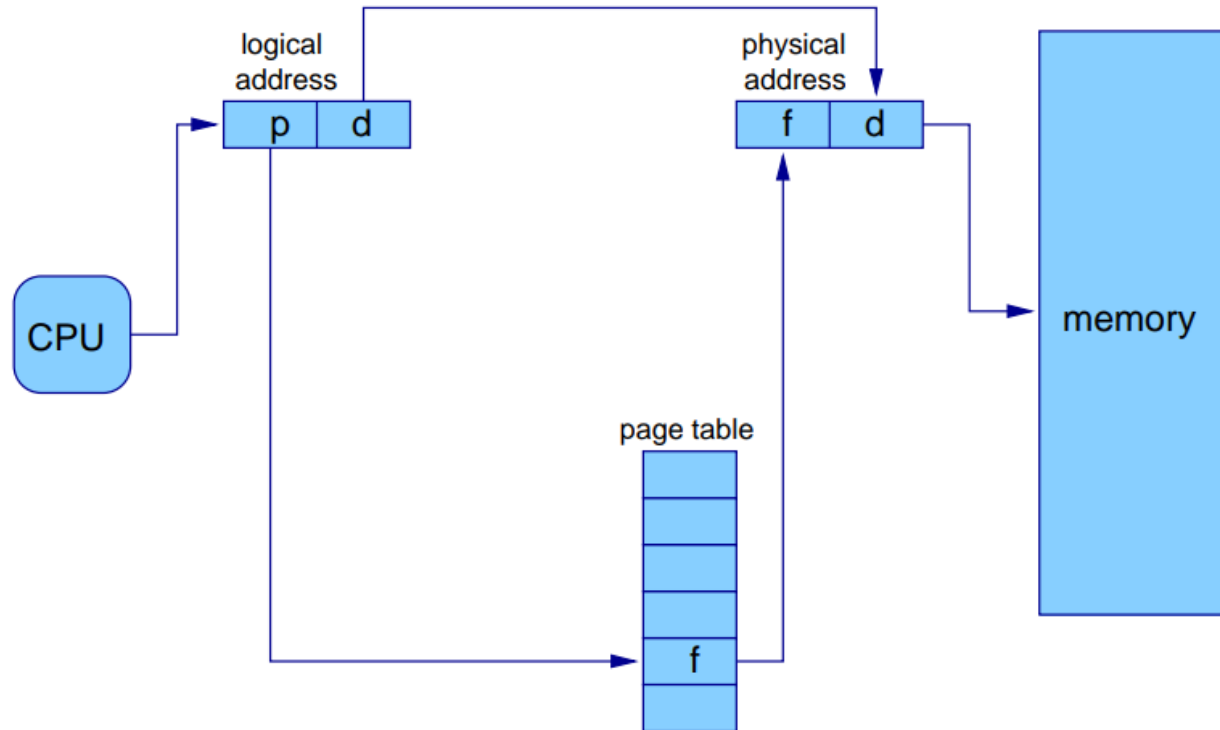
# Aadresside tõlkimine (1)

Mäluaadressi bitid jagatakse kahte gruppi:

- Lehekülje number (p) — kasutatakse indeksina leheküljetabelisse lehekülje füüsilise aadressi leidmiseks
- Nihe lehekülje sees (d) — liidetakse lehekülje füüsilisele aadressile täieliku füüsilise aadressi saamiseks

Kontekstivahetuse ajal programmeerime MMU vastavalt protsessi leheküljetabelile

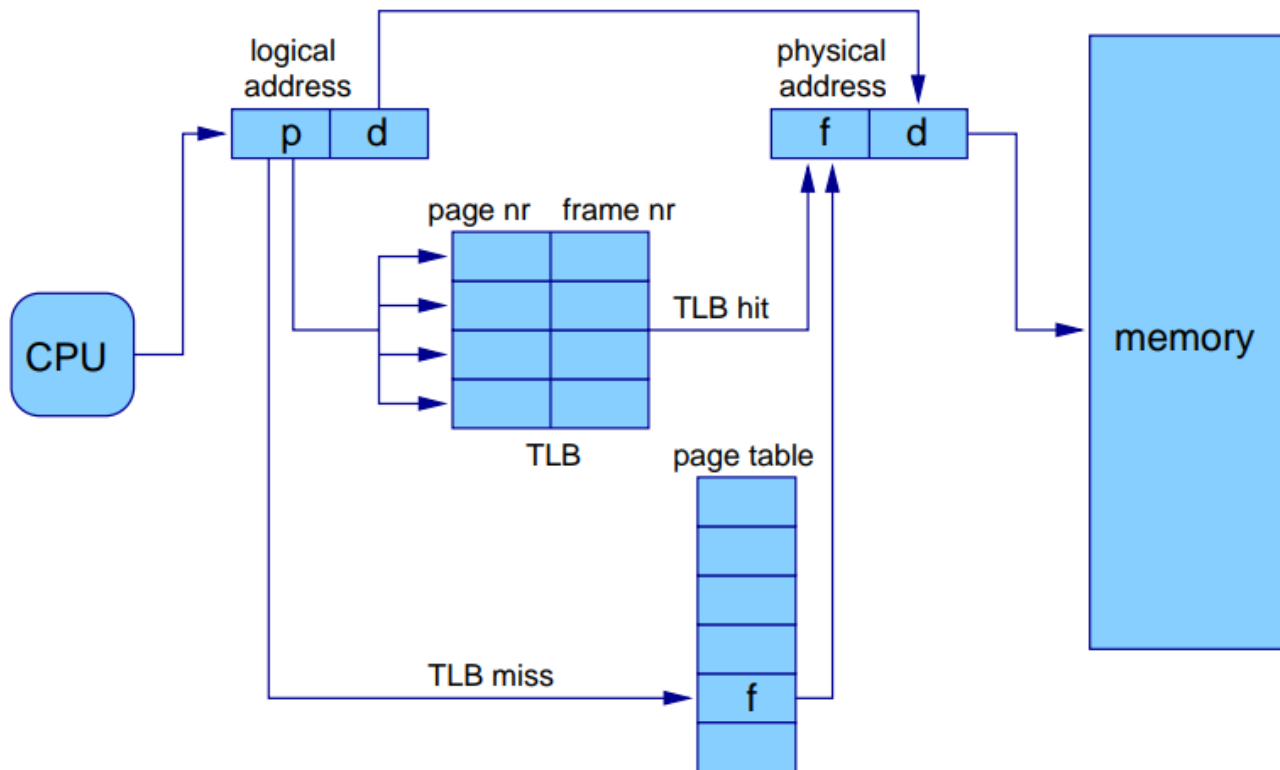
## Aadresside tõlkimine (2)



# Leheküljetabelite realiseerimine

- Leheküljetabeleid hoitakse mälus
- Leheküljetabeli algust näitab vastav register (PTBR)
- Leheküljetabeli pikkust näitab kah vahel register (PTLR)
- Iga soovitatav mälupöördus nõuaks kahte tegelikku pöördust
- Seda probleemi lahendab viimati- (sagedamini?) kasutatud kirjete puhverdamine protsessori või MMU sees
- TLB (Translation Lookaside Buffer) — assotsiatiivmälu abil realiseeritud leheküljetabeli kirjete puhver
- TLB miss — alati ei leita vastavat kirjet TLB-st, sel juhul tuleb ta mälust TLB-sse lugeda
- TLB flush
- ASID (Address Space ID), virtuaalmasina märgistatud TLB

# Aadresside tõlkimine TLB abil



# Efekiivne mälupöördusaeg

- Reaalne mälupöördus = 1 ajaühik
- Assotsiatiivmälust otsimine =  $q$  ajaühikut
- Assotsiatiivmälu edukate otsimiste osakaal (hit ratio) =  $\alpha$ 
  - mitmel protsendil juhtudest leiab tulemise assotsiatiivmälust.
- Efekiivne mälupöörduse aeg:

$$EAT = (1 + q)\alpha + (2 + q)(1 - \alpha) = 2 + q - \alpha$$

# Mälukaitse (1)

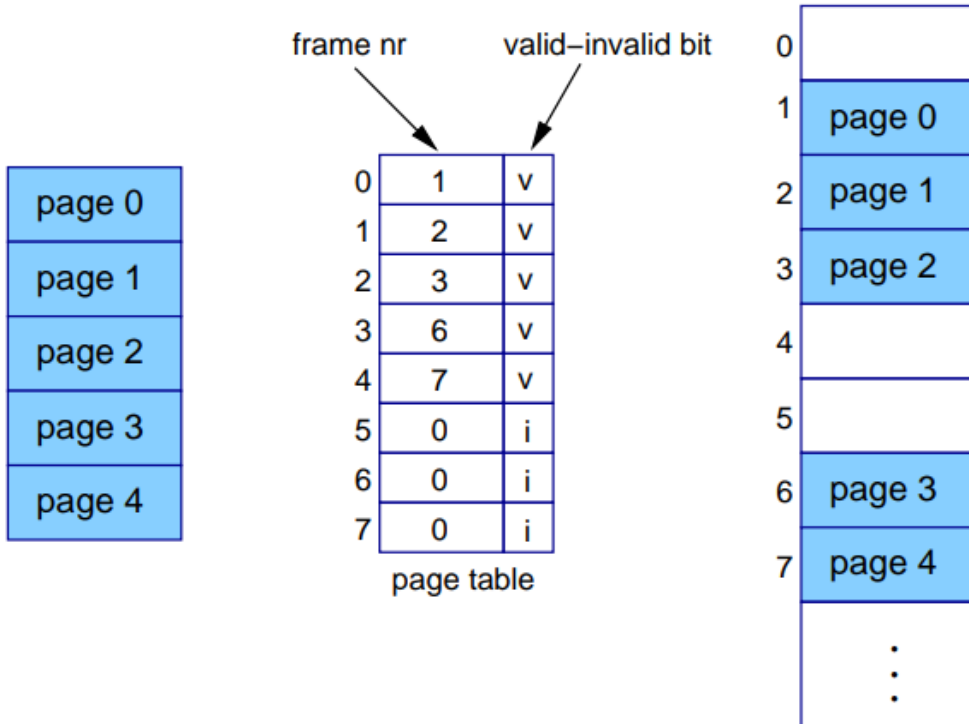
Mälukaitset saab (ja on mõtet) siduda iga leheküljega

Iga lehekülje juurde paneme lipu valid

- valid — lehekülg on protsessi mälupiirkonnas
- invalid — lehekülge ei ole protsessi mälupiirkonnas

Näiteks mälupiirkonna lõpu tähistamiseks

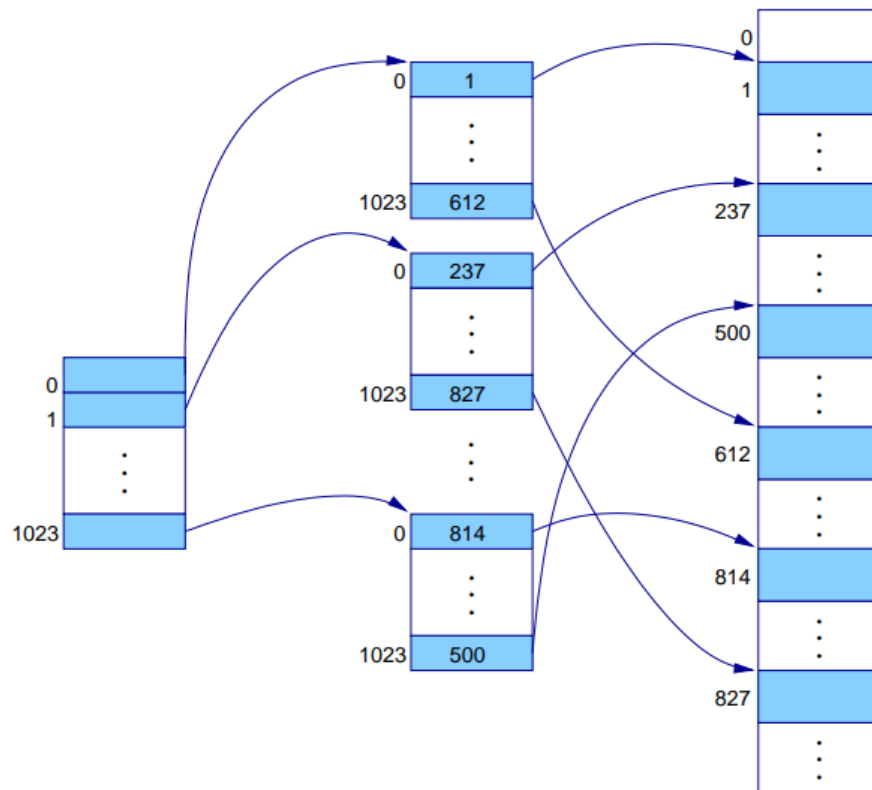
## Mälukaitse (2)



# Hierarhilised leheküljetabelid

- Kuna üks leheküljetabel läheks liiga suureks, teeme leheküljetabeli mitmetasemelise (hierarhilise)
- Kahetasemeline leheküljetabel — aadress jagatakse leheküljesiseseks aadressiks ning lehekülje number omakorda kaheks bitigrupiks (kummagi taseme jaoks üks grupp, määrab indeksi vastavas tabelis)
- Näiteks 32-bitine aadress, 4K lehekülg (12 bitti), ülejäänud 20 bitti jagatakse kaheks 10-bitiseks leheküljenumbriks (välimise leheküljetabeli indeks ja sisemise leheküljetabeli indeks)
- Füüsilised leheküljed võivad endiselt olla läbisegi (sõltumata kummagi tabeli järjekorrast)
- Kolmetasemelised, neljatasemelised, . . .

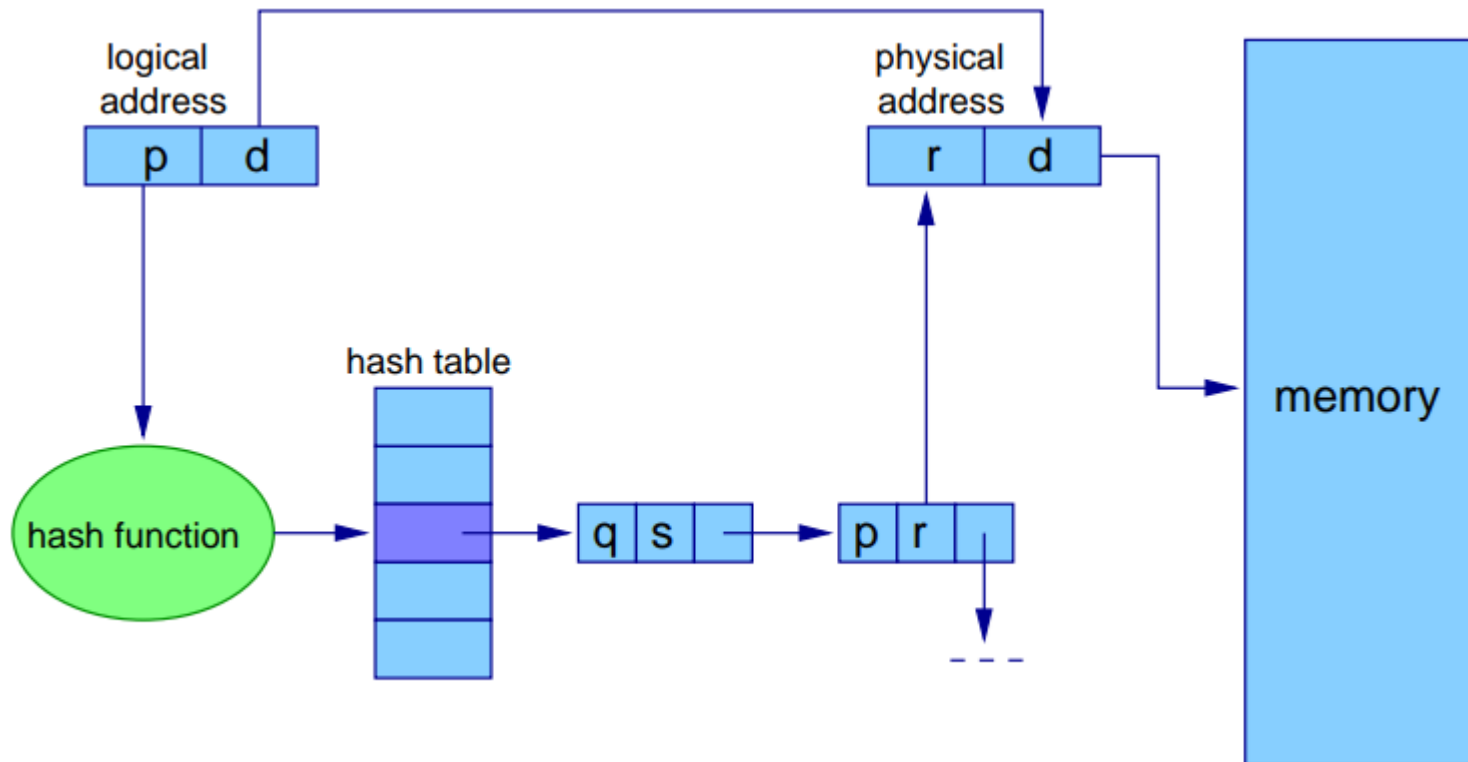
# Kahetasemeline leheküljetabel



# Paiksallvestusega leheküljetabelid (1)

- 64-bitisel süsteemil tuleks palju tasemeid leheküljetabeleid, pole efektiivne
- Virtuaalse lehekülje number salvestatakse paiskfunksiooni abil leheküljetabelisse
- Leheküljetabel on välisahelatega paisktabel (mitu erinevat lehekülge võivad samale positsioonile sattuda)
- Kasutamine kiire
- Kontekstivahetus (väga) aeglane

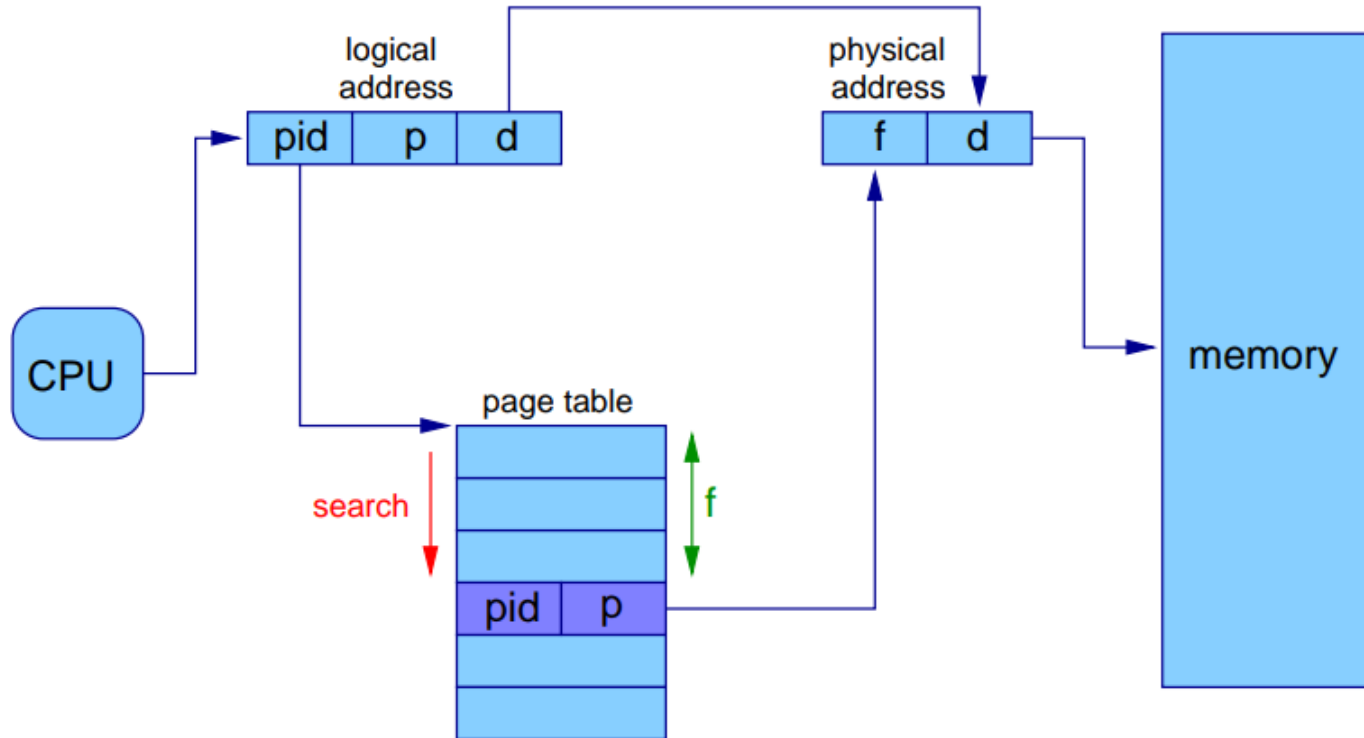
## Paiksaldvestusega leheküljetabelid (2)



# Pööratud leheküljetabel (1)

- Idee: leheküljetabelis on kirje iga füüsilise mälu lehekülje kohta
- Kokku 1 tabel, mitte igal protsessil oma tabel, mida siis vahetada tuleks
- Füüsilise mälu leheküljele vastav tabelikirje koosneb protsessi identifikaatorist ja protsessisisisest loogilisest aadressist
- Mälutarve väiksem
- Otsimisaeg suurem
- Otsimise kiirendamiseks paisktabel leheküljetabeli ees (muidugi ka TLB)

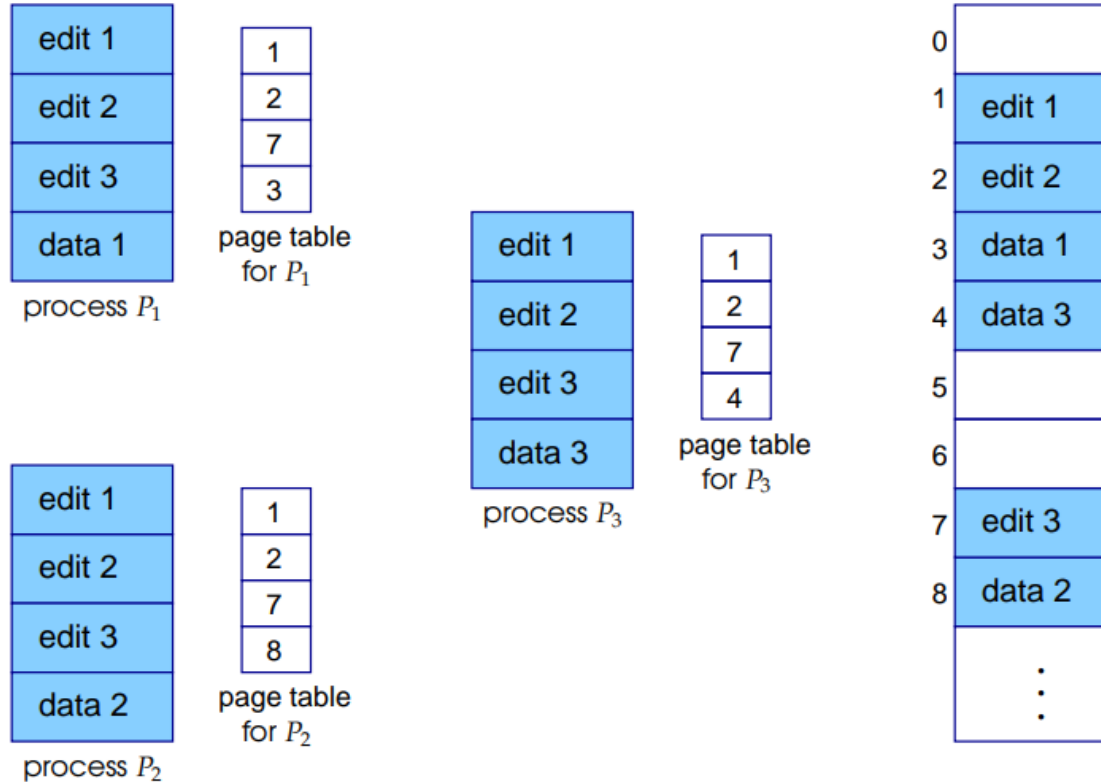
## Pööratud leheküljetabel (2)



# Jagatud mäluleheküljed (1)

- Koodi sisaldavaid lehekülgi on mõtet jagada
- Võimalik siis, kui kood on taassisenetav (reentrant) — ei modifitseeri iseennast
- Jagatud lehekülg on nähtav mitme protsessi aadressiruumist
- Sissekirjutatud aadressidega kood peab igas protsessis samal virtuaalaadressil asuma
- Positsioonist sõltumatu kood (PIC — Position Independent Code) võib igas protsessis olla erineva virtuaalaadressi peal
- Tänapäeval on enamik jagatud teeke positsioonist sõltumatud
- Igal protsessil oma andmed ja enamasti ka mingi osa mittejagatud koodi
- Pööratud leheküljetabelite puhul on jagamine raskem

# Jagatud mäluleheküljed (2)



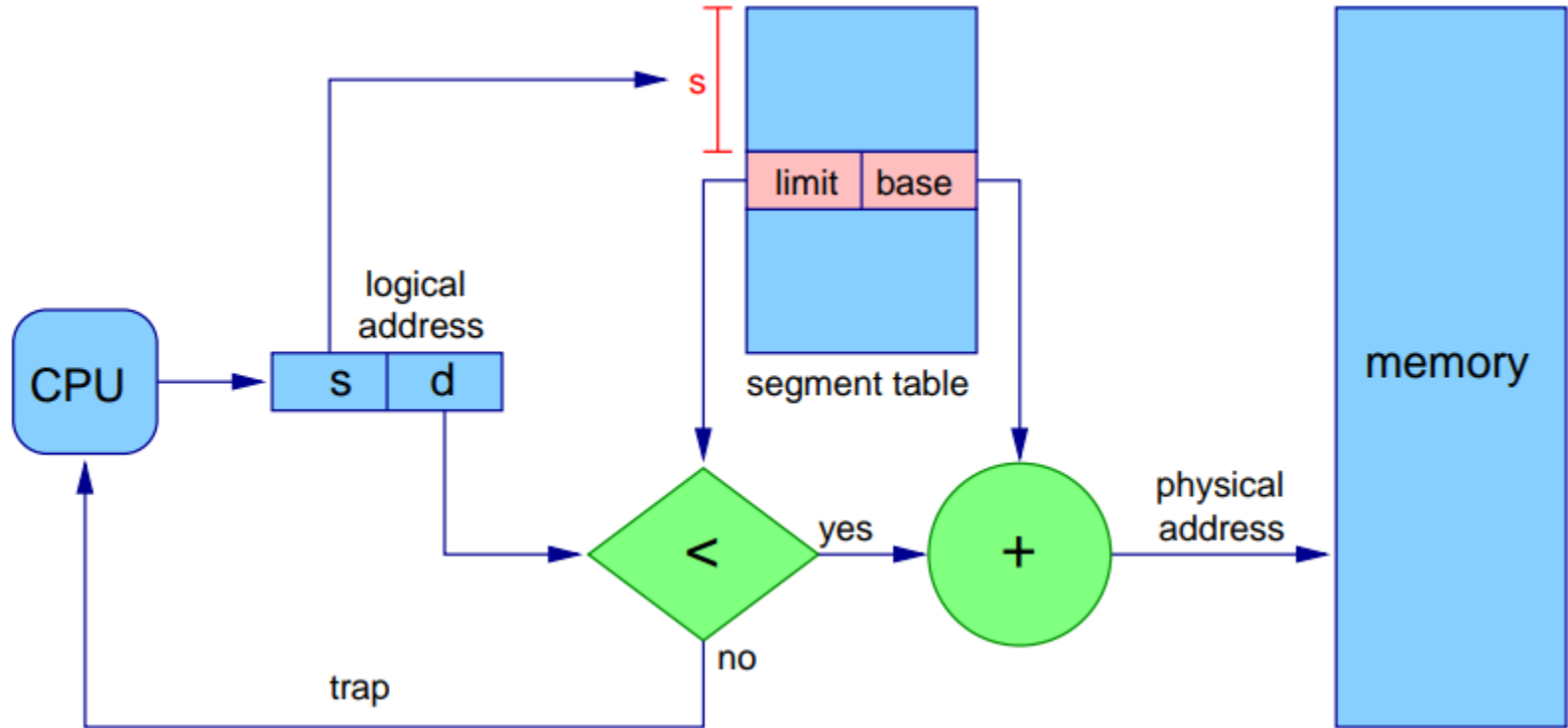
# Segmenteerimine

- Segmenteerimine on mäluhalduse skeem, kus ei kasutata mitte ühte suurt pidevat loogilist aadressiruumi, vaid paljusid segmente
- Läheb paremini kokku paljude kasutajate mõttemaailmaga
- Programm on komplekt segmente. Segment on loogiline ühik, näiteks
  - Põhiprogramm
  - Protseduur, funktsioon, meetod
  - Objekt
  - Lokaalsed muutujad, globaalsed muutujad
  - Magasin
  - . . .

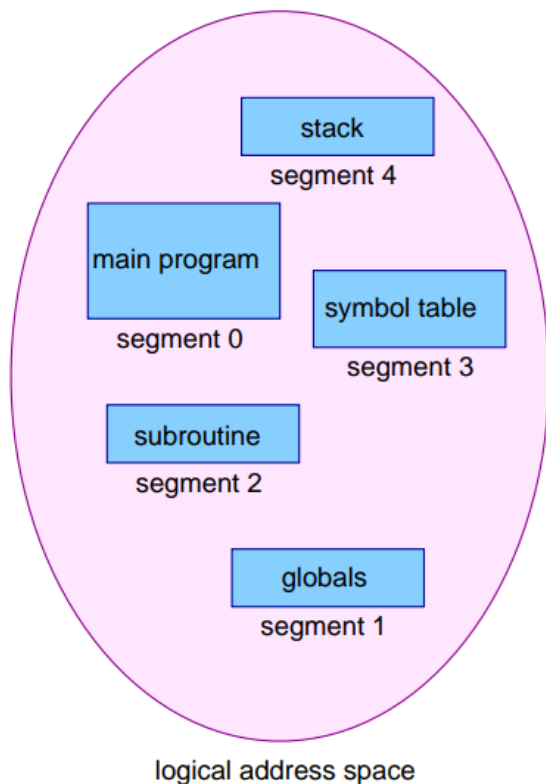
# Segmenteerimisega arhitektuur

- Loogiline aadress on paar <segmendi number, nihe>
- Segmenditabel — seab igale segmendi numbrile vastavusse mälutüki:
  - Baas — segmendi alguse füüsiline aadress
  - Limiit — segmendi pikkus
- Segmenditabeli algus näidatakse vastava registriga (STBR)
- Segmenditabeli pikkus enamasti registris (STLR) — sisuliselt on tegemist maksimaalse kasutatava segmendi numbriga
- Mälu jagatakse tervete segmentidega kaupa (first-fit, best-fit näiteks), seega tekib väline fragmenteerumine
- Segmente saab protsesside vahel jagada

# Aadresside teisendamise segmenteerimisel

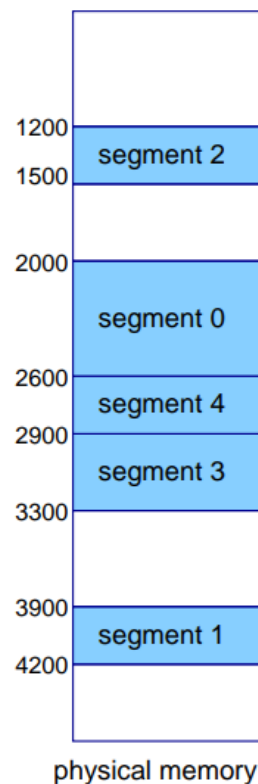


# Näide segmenteerimisest

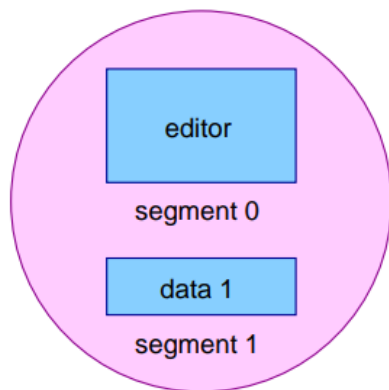


	limit	base
0	600	2000
1	300	3900
2	300	1200
3	400	2900
4	300	2600

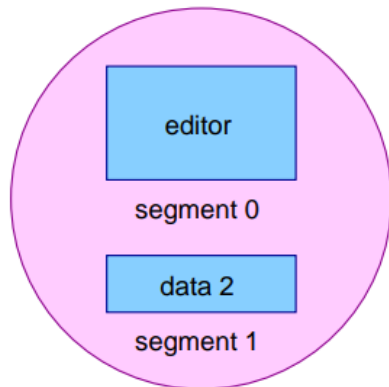
segment table



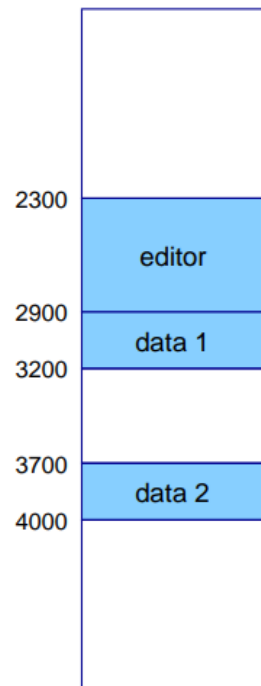
# Näide segmenteerimisest



	limit	base
0	600	2300
1	300	2900



	limit	base
0	600	2300
1	300	3700



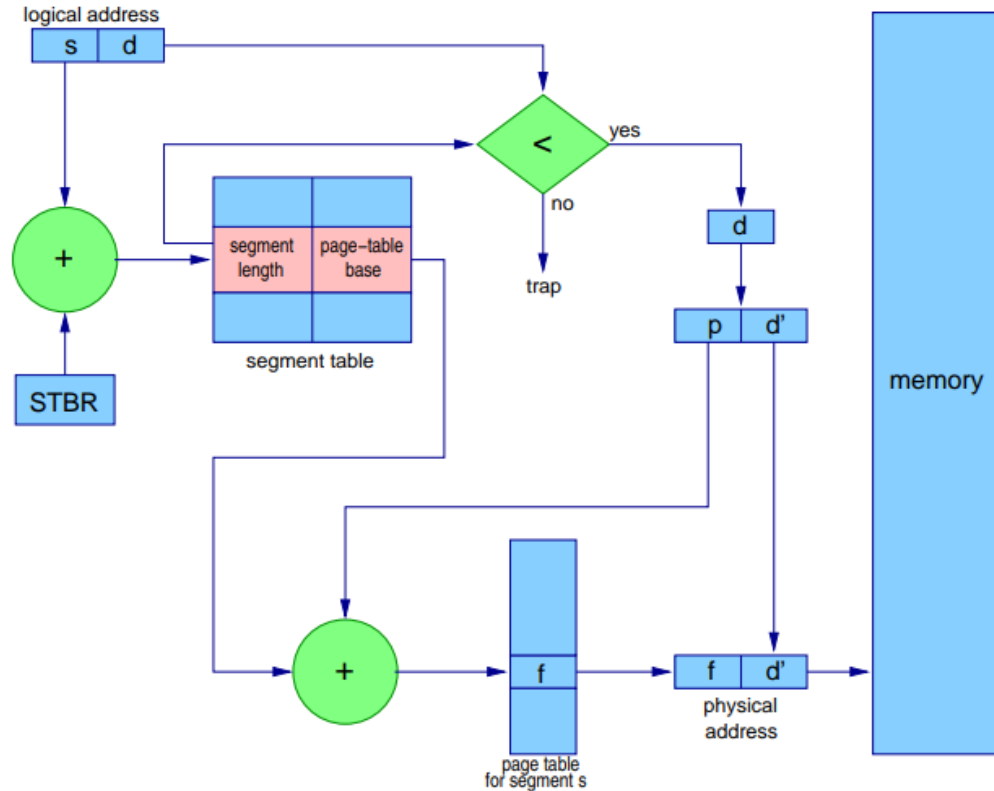
# Segmenteerimine ja mälukaitse

- Mälukaitse läheb üsna loomulikku rada — enamasti on tervel segmendil sama kaitsetase, seega seome loabiti(d) segmenditabeli kirjega:
  - valid — kas segment on kasutusel
  - lugemise/kirjutamise/koodi täitmise lubamise bitid (suvalises kombinatsioonis)

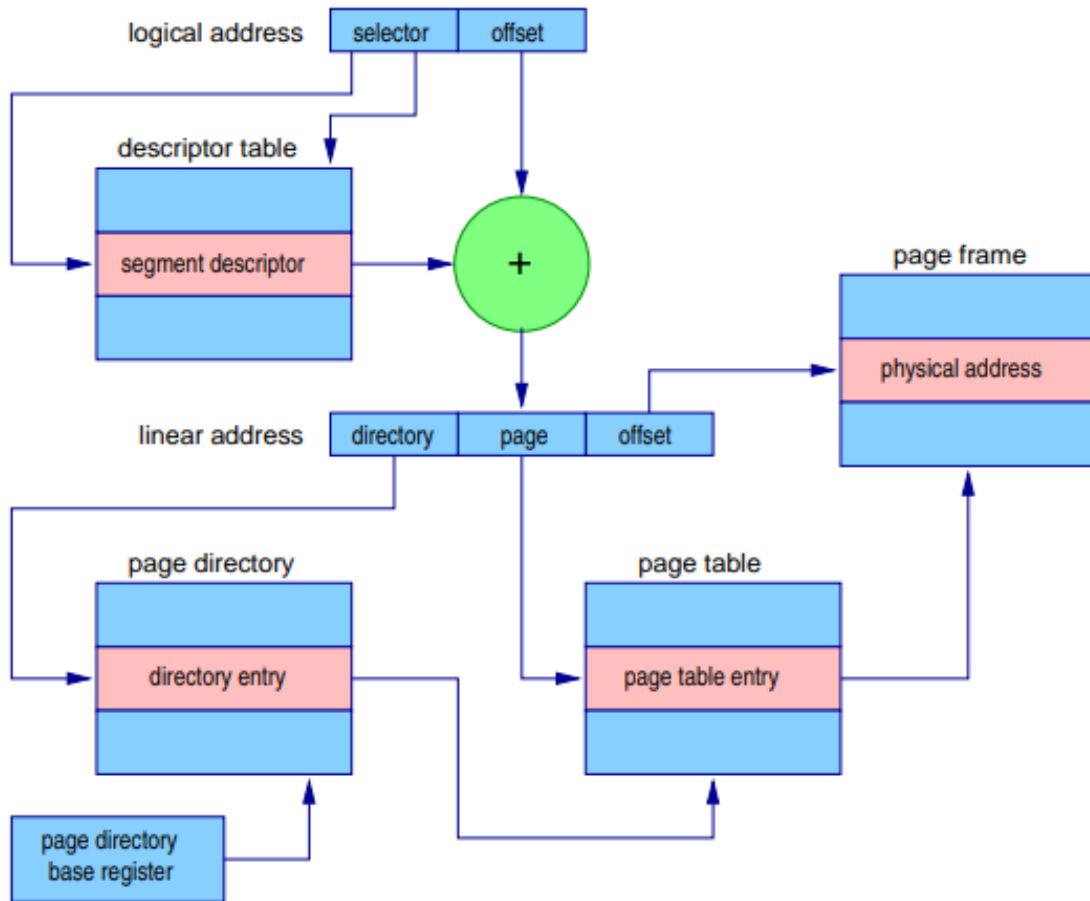
# Segmenteerimine koos lehekülgedega

- MULTICS lahendas välise fragmenteerumise probleemi segmenditabelite lehekülgedeks jagamisega — iga segmenditabeli kirje oli viide vastava segmendi leheküljetabelile
- Sarnane kombineeritud lähenemine on tänapäevalgi kasutusel
- Inteli 386 (tegelikult kogu edasine x86 haru) kasutab ka segmenteerimist ja lehekülgi mõlemaid, kuid üksteise järel: segmenditabelist saadakse tulemusena mingi aadress ning sellele rakendatakse kahetasemelist leheküljetabelit
- Mõlema head küljed ära kasutatavad, natukese mälu ja keerukuse (OS ja protsessor) hinnaga

# Segmenteerimine koos lehekülgedega MULTICS-is



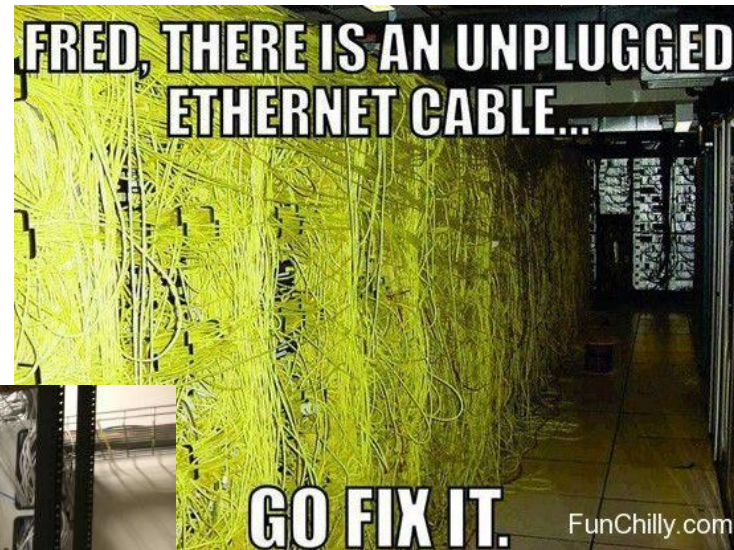
# Segmenteerimine koos lehekülgedega i386-s



Võrk

# Võrk

- Võrguprotokoll
- OSI mudell
- Võrguliidesed
- Edastusviisid
- Ethernet
- IP
- TCP
- UDP
- NAT
- IPv6
- Sockets



# Võrguprotokoll

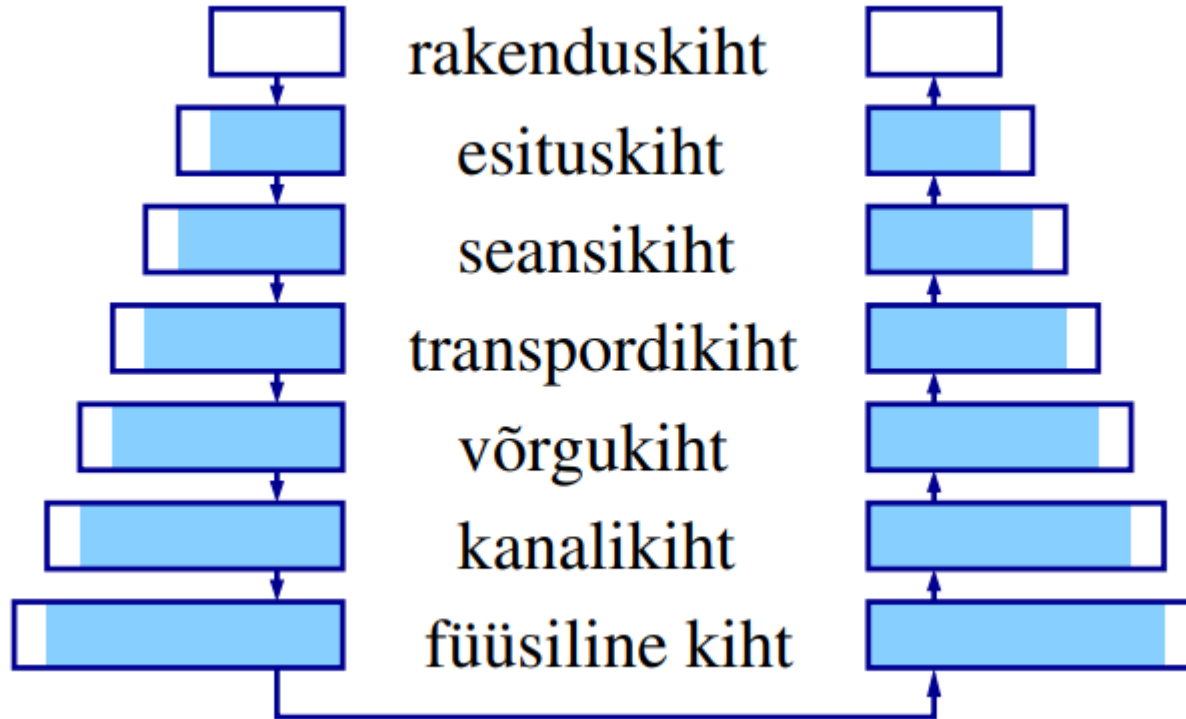
**Võrguprotokoll** on reeglite ja protseduuride kogum, mis määratlevad, **kuidas andmed liiguvad läbi võrkude**. Need protokollid võimaldavad erinevatel võrguseadmetel, nagu arvutid ja serverid, omavahel suhelda ning andmeid edastada. **Protokollid määratlevad, kuidas andmeid pakendatakse, saadetakse, vastu võetakse ja dekodeeritakse.**

Näiteid võrguprotokollidest:

- TCP (Transmission Control Protocol) - Tagab andmete usaldusväärse edastamise, korrigeerides vigu ja haldades andmevoogu.
- IP (Internet Protocol) - Vastutab andmepakettide saatmise eest ühest arvutist teise, määrates siht- ja lähteaadressid.
- HTTP (Hypertext Transfer Protocol) - Kasutatakse veebisaitide sisu edastamiseks veebiserveritest veebibrauseritesse.
- SMTP (Simple Mail Transfer Protocol) - Võimaldab e-kirjade saatmist ja vastuvõttu.
- FTP (File Transfer Protocol) - Kasutatakse failide ülekandmiseks arvutite ja serverite vahel.

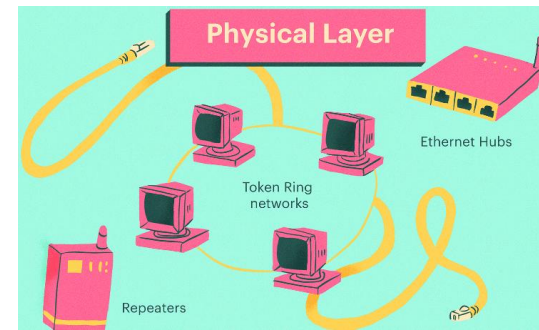
Iga protokoll täidab oma kindlat rolli võrgusuhtluses, tagades andmevahetuse sujuvuse ja efektiivsuse.

# ISO/OSI 7-kihiline mudel

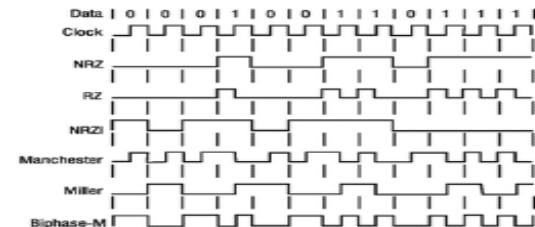
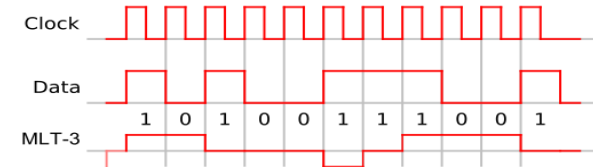


# Füüsiline kiht (physical layer)

- **Eesmärgid:** Füüsiline kiht võimaldab toorandmebittide edastamist füüsilise meedia kaudu.
  - a. Kahe seadme vaheline side, kandja kujundus, elektrisignaali määratlemine
- **Minimaalne ühik: 1 bitt**
- **Riistvara Tugi:** See kiht kasutab kaableid, hube, retranslaatoreid, võrgukaarte ja modemeid andmeedastuseks.
  - a. Cat5e või Cat6 Etherneti kaableid või Wi-Fi ruuter, switch, hub, Ethernet adapter, ADSL modem, Wi-Fi adapter
- **Operatsioonisüsteemi Tugi:** Operatsioonisüsteemid sõltuvad füüsilise kihi riistvarast ja püsivarast andmeedastuseks.
  - a. Windows või Linux kasutab võrgukaardi draivereid füüsilise kihi funktsioonideks.
- **Draiverite Tugi:** Füüsilise kihi seadmed vajavad nende tööks spetsiifilisi draivereid.
  - a. Intel Gigabit Etherneti võrgukaardi draiverid (e1000)
  - b. Broadcom, Realtek, Atheros, Qualcomm
- **API Tugi:** Füüsilise kihi API-d on tavaliselt madala taseme ja ei ole lõppkasutajale nähtavad.
  - a. Ethernet controller low level API
- **Protokollid:** Füüsiline kiht tegeleb andmete edastamise füüsiliste standardite ja spetsifikatsioonidega, mitte kõrgema taseme protokollidega.
  - a. RZ, NRZ, Manchester Miller, MLT-3

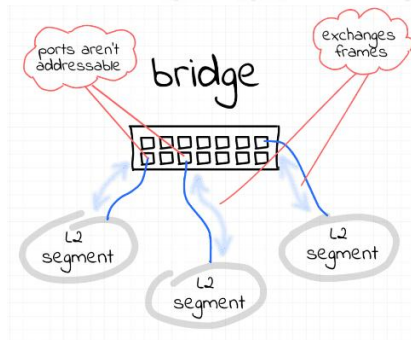
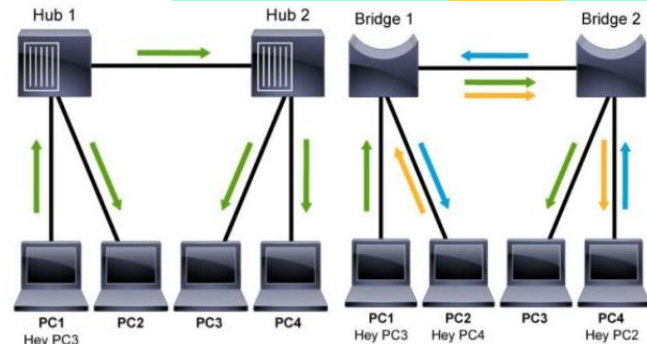
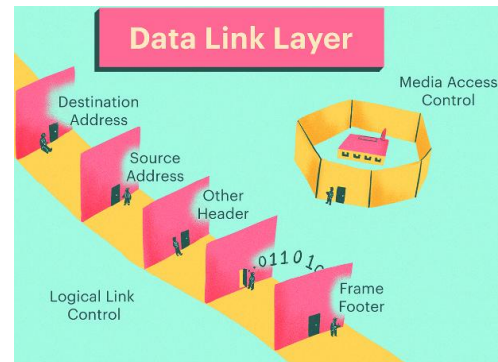


Cat5e Wire Diagram for T568B (Straight Through Cable)					
RJ45 Pin #	Wire Color (T568A)	Wire Diagram (T568A)	10Base-T Signal	100Base-TX Signal	1000Base-T Signal
1	White/Orange		Transmit+	BI_DA+	
2	Orange		Transmit-	BI_DA-	
3	White/Green		Receive+	BI_DB+	
4	Blue		Unused	BI_DC+	
5	White/Blue		Unused	BI_DC-	
6	Green		Receive-	BI_DB-	
7	White/Brown		Unused	BI_DD+	
8	Brown		Unused	BI_DD-	



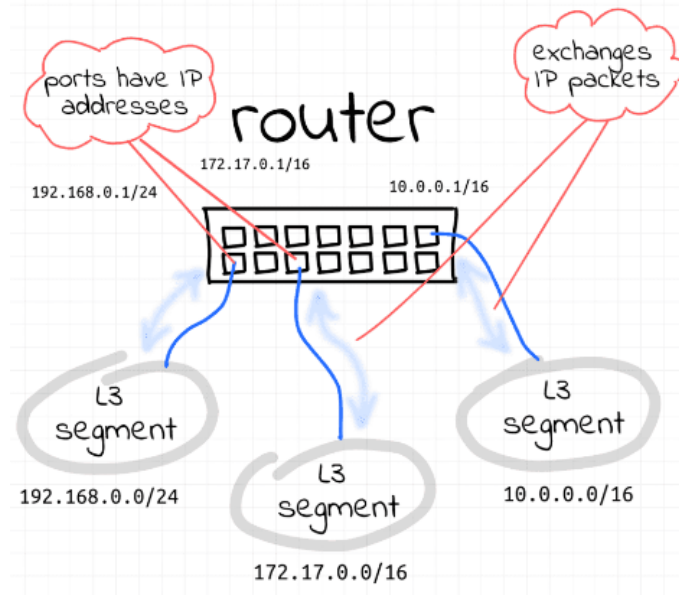
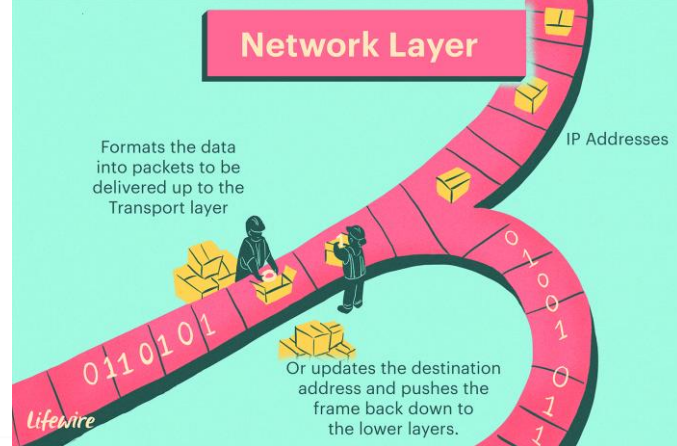
# Kanalikiht (Link layer)

- **Eesmärgid:** Andmesidelinkikiht tagab otseühendatud sõlmede vahelise usaldusväärse andmeedastuse ja veaparanduse.
  - a. Tüüpiline LAN over Ethernet: 1 switch ja mitu arvutit või 1 wi-fi AP ja mitu arvutit
- **Minimaalne ühik: 1 kaader (frame)**
- **Riistvara Tugi:** Kasutab võrgulüliteid, sildu ja võrgukaarte.
  - a. Switch, ethernet adapter, wi-fi APN, wi-fi adapter
- **Operatsioonisüsteemi Tugi:** Operatsioonisüsteemid kasutavad andmesidelinkikihi jaoks spetsiifilisi draivereid ja tarkvara.
  - a. Windows ja Linux kasutavad võrguadapteri draivereid linkikihi funktsioonide jaoks.
- **Draiverite Tugi:** Näiteks Etherneti lüliti draiverid.
  - a. Intel, Broadcom, Realtek, Atheros, Qualcomm
- **API Tugi:** Pakub API-sid võrgu seadmete juhtimiseks, nagu libpcap võrguliikluse jälgimiseks.
  - a. libpcap - võimaldab võrguliikluse jälgimist ja analüüsimist.
- **Protokollid:** Andmesidelinkikihi protokollid hõlmavad Etherneti, PPP ja MAC aadressimist.
  - a. Etherneti protokoll (IEEE 802.3) ja MAC (Media Access Control) aadressimine
  - b. Logical Link Control (LLC), Spanning-tree protocol (STP)



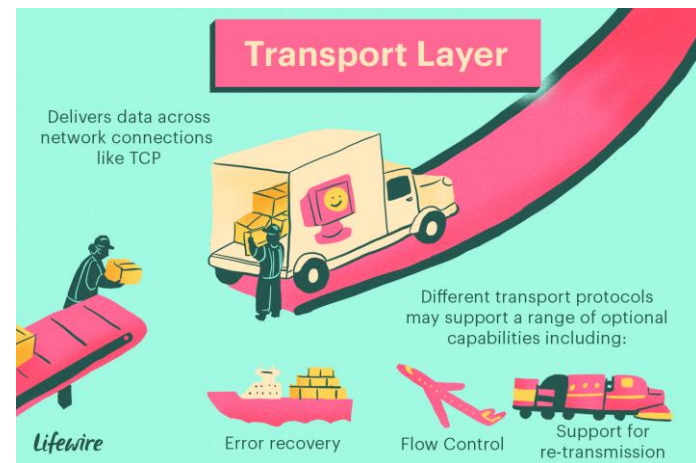
# Võrgukiht (Network layer)

- Tee otsimine võrgus rohkem kui kahe seadme vahel
- Kommuteerimine ja marsruudi leidmine
  - Igale pakatile eraldi
  - Ühe korra virtuaalse kanali loomisel
- Võrgusõlmede adresseerimine
- Pakettide edastamine erinevate võrkude vahel
- Pakettide tükeldamine ja kokkupanek



# Transpordikiht (Transport layer)

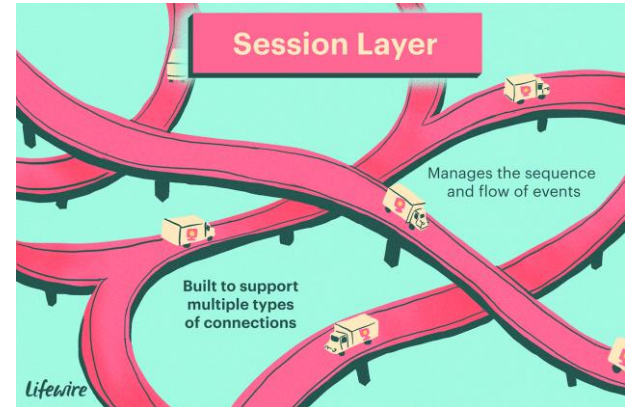
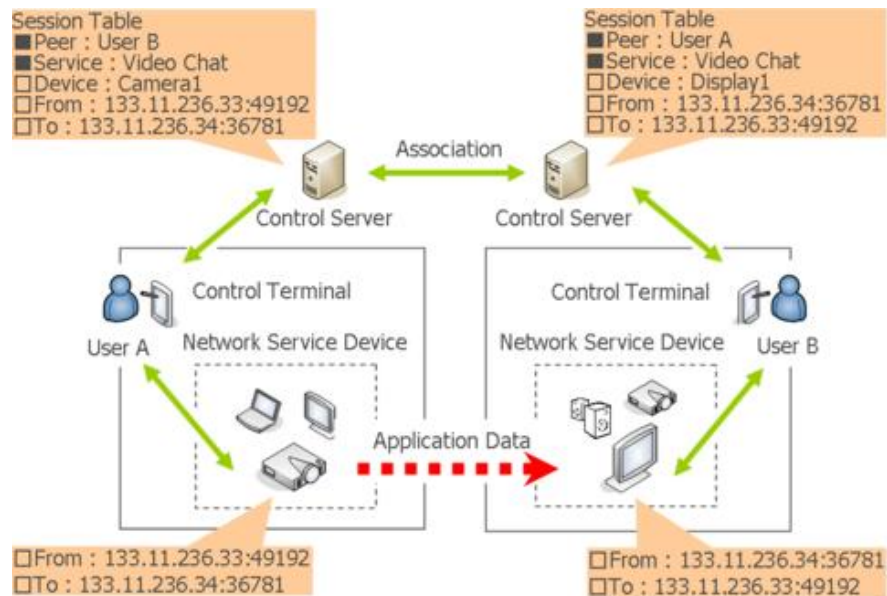
- Andmete läbipaistev transport kahe rakenduse vahel
- Vajadusel garanteerib andmete järjestuse
- Vajadusel garanteerib andmete uuestisaatmise
- Tegeleb otspunktide vahelise vookontrolliga
- Ummistuste lahendamine (congestion control)



TCP	UDP
Secure	Insecure
Connection-Oriented	Connectionless
Slow	Fast
Guaranteed Transmission	No Guarantee
Used by Critical Applications	Used by Real-Time Applications
Packet Reorder Mechanism	No Reorder Mechanism
Flow Control	No Flow Control
Advanced Error Checking	Basic Error Checking (Checksum)
20 Bytes Header	8 Bytes Header
Acknowledgement Mechanism	No Acknowledgement
Three-Way Handshake	No Handshake Mechanism
DNS, HTTPS, FTP, SMTP etc.	DNS, DHCP, TFTP, SNMP etc.

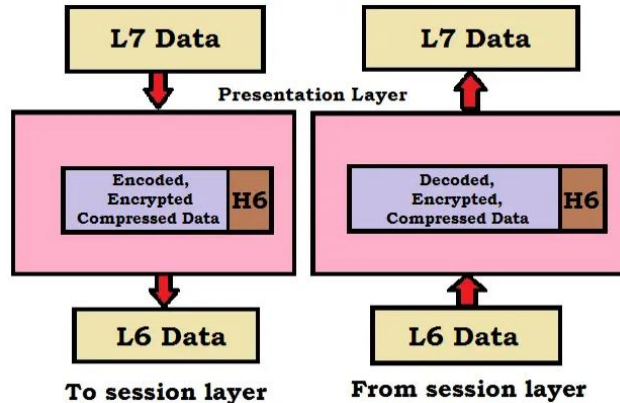
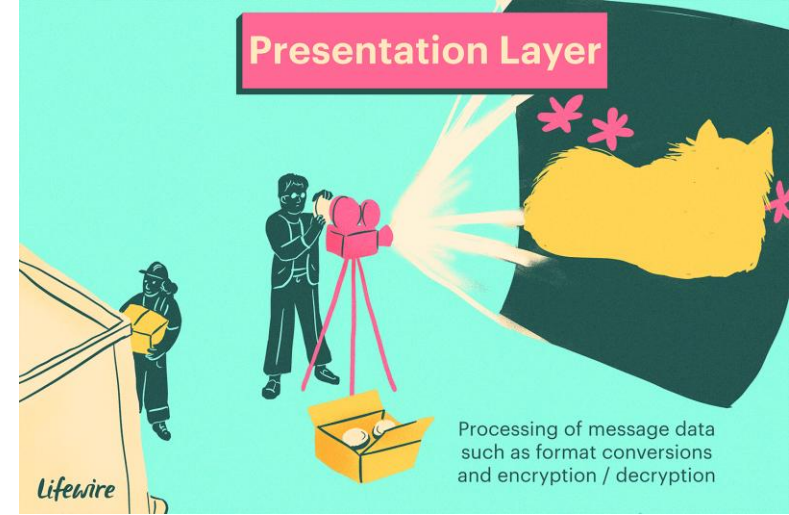
# Seansikiht (Session layer)

- Seansihaldus kahe osapoole vahel:
  - Loob, haldab ja lõpetab loogilisi seansse
- Tegeleb ka seansside jätkamisega vea korral

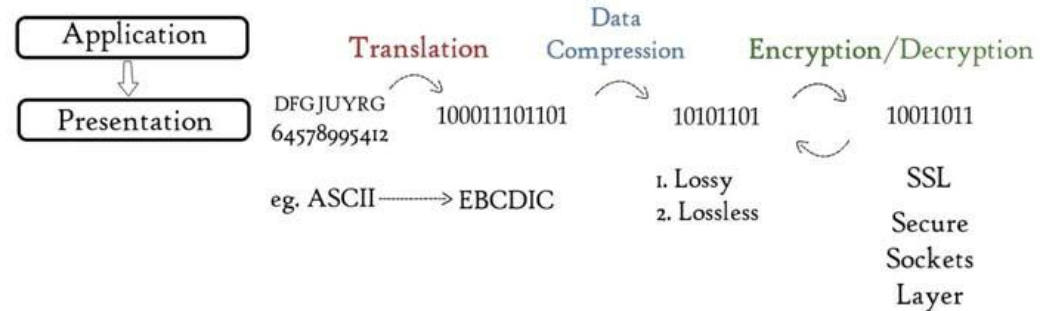


# Esitluskiht (Presentation Layer)

- Andmete esituskujust sõltumatu tõlkekiht
- Tegeleb andmete kodeeringuga, struktuurse esitusega
- Krüpteerimine • Nn. süntaksikiht

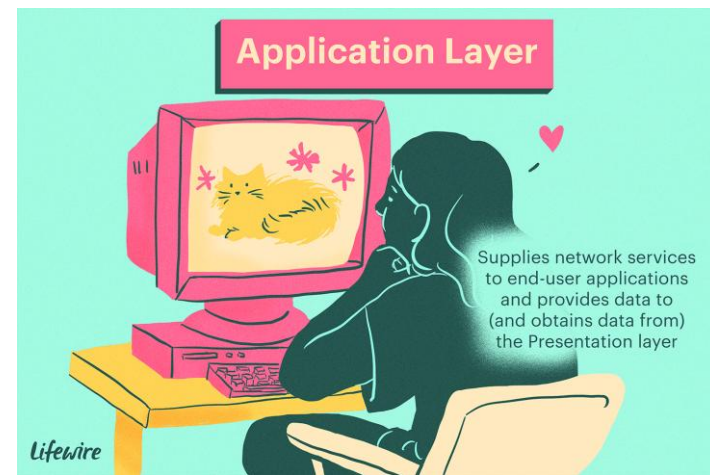


## Presentation Layer

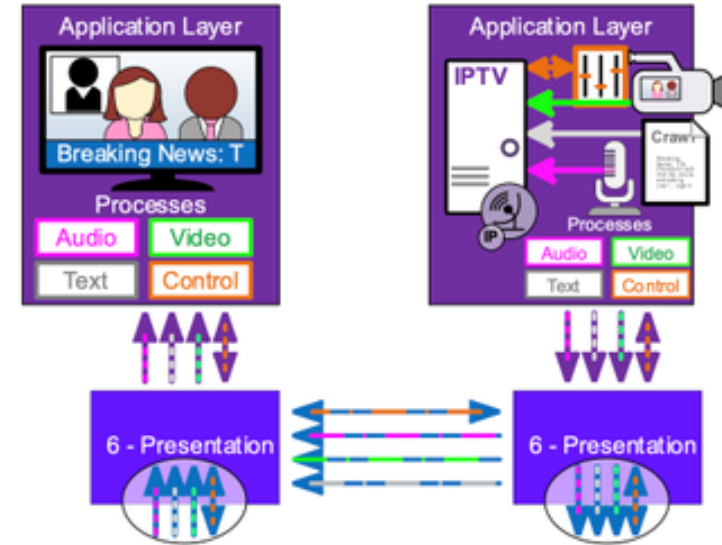
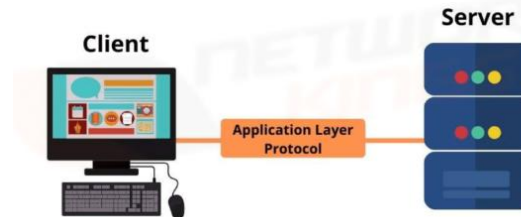


# Rakenduskiht (Application layer)

- Rakendusprogrammide spetsiifilised protokollid
- Iga rakendus saab defineerida oma protokollid
- Kõik ülejäänud aspektid on rakenduskihi hallata



## Application Layer Protocol



# Internet ja OSI mudel

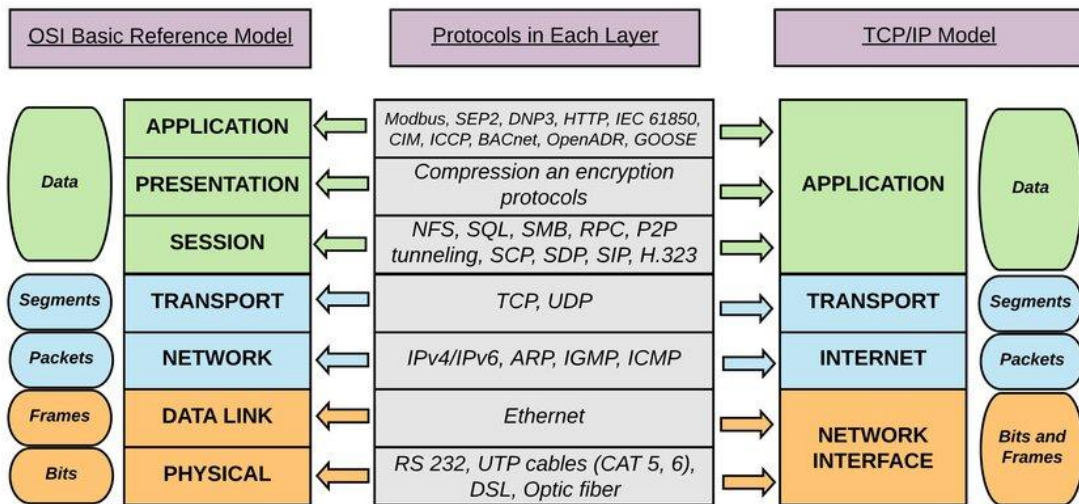
Füüsiline kiht — igasugused, näiteks Ethernet, WiFi

Kanalikiht — MAC aadressidega arvutite adresseerimine, Etherneti 802.3 kaadriformaat, ARP protokoll IP ja MAC vastavuse leidmiseks

Võrgukiht — IP: pakettide marsruutimine õigesse võrku

Transpordikiht: TCP (töökindel baidivoog), UDP (sõltumatute datagrammide saatmine)

Kolm ülemist kihti on kokku sulanud rakenduskihiks • Aegajalt on seansihaldust või esituskihi funktsionaalsust võimalik rakenduse protokollis eristada



# Võrguliidesed

Ühel arvutil võib olla üks või mitu erinevat füüsilist võrguliidest

Lisaks on tavaliselt kasutusel lokaalne arvutisisene võrguliides (loopback)

Enamasti on igal liidesel oma aadress

- Mitmesse L3 võrku ühendatud arvutil peab olema igal liidesel aadress vastavast võrgust

```
# Linux
```

```
me@myhost:~$ ip addr
```

```
...
```

```
me@myhost:~# ifconfig
```

```
...
```

```
# Windows
```

```
C:\Users\me> ipconfig
```

```
...
```

# Mis on rakenduse ja mis OS-i realiseerida?

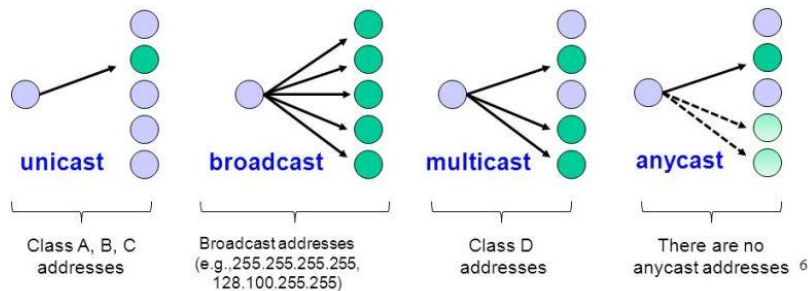
- Tavalahendus on realiseerida protokollivirna kihid 2-4 opsüsteemis
  - Rakenduste isoleerimine üksteise eest
  - Pordinumbrite hõivamine
  - Soklite jagamine protsesside vahel
- Alternatiiv: transpordikihi (4) kasutajarakendusse toomine
  - Vookontroll ja protokolliparsimine opsüsteemi seest protsessi sisse (end-to-end mudeli laiendamine parema skaleerimise huvides)
  - Opsüsteemi peab jääma minimaalne demultipleksimine ja jagatud ressursside haldus •
- Rakenduskiht on pea alati kasutaja tasemel, Internet mudelis koos sellega ka seansi- ja esituskiht

# Edastusviisid

- Ainuedastus (unicast) — andmepaketi saatmine üle võrgu ühelt saatjalt ainult ühele vastuvõtjale
- Leviedastus (broadcast) — andmepaketi saatmine üle võrgu ühelt saatjalt kõigile võrgusõlmedele mingis piirkonnas
- Multiedastus (multicast) — andmepaketi saatmine üle võrgu ühelt saatjalt valitud vastuvõtjate rühmale
- Suvaedastus (anycast) — andmepaketi saatmine üle võrgu ühelt saatjalt rühma lähimale vastuvõtjale

## Delivery modes

- Supported by IPv4
  - one-to-one (unicast)
  - one-to-all (broadcast)
  - one-to-many (multicast)
- Not supported by IPv4:
  - one-to-any (anycast)



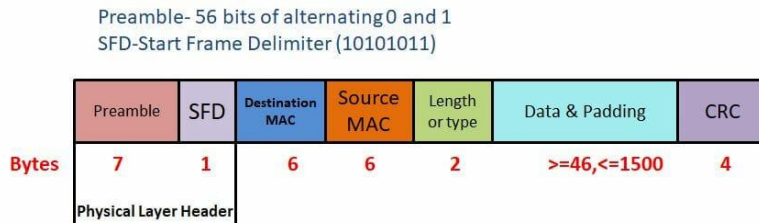
# Ethernet

Ethernet on tänapäeval protokollide pere, kus on sama kaadri formaat kuid erineva meedia ja kiirusega sidekanalid (kuni 100 Gbit/s seni standarditud)

- Lisaks alguse ja lõpu markeritele sisaldab üks Etherneti kaader:

- Preamble (fikseeritud baitide jadaa)
  - andmete edastuse alguses sünkroonida saatja ja vastuvõtja kellasid
- Start frame delimiter
  - Kaadri alguse eristaja
  - selle väärtus on alati "10101011"
  - aitab vastuvõtjal eristada Preamble'it
- Saatja MAC aadress
- Saatja MAC aadress
- Mittekohustuslik VLAN tag
  - (mitme loogilise võrgu tegemiseks)
  - samas kaablis)
- Andmeosa pikkus
- Andmeosa (42-1500 baiti)
- Kaadri kontrollsumma (32-bitine CRC)

## Ethernet Frame



# MAC-aadress

- Teise arvuti adresseerimiseks 2. kihi (L2) võrgus peab tema aadressi teadma (näiteks MAC-aadress Etherneti-laadsete protokollide puhul)
- MAC — Medium Access Control
- 48-bitine idee poolest unikaalne seadme identifikaator
- Näiteks 00:0a:e4:7e:a5:e0
- Igal võrguliidesel on aadress tootja poolt sisse programmeeritud
- Koosneb tootja prefiksist ja unikaalsetest baitidest tootja piires
- MAC-aadress ei paista kohtvõrgust (LAN) kaugemale!

# IP

- Internet Protocol
- Palju kohtvõrke on kokku ühendatud, IP-aadresside järgi leitakse tee läbi mitme võrgu õige seadmeni
- IPv4 kasutab 32-bitiseid aadresse
  - 127.0.0.1 = 01111111 00000000 00000000 00000001
  - 192.0.2.1 = 11000000 00000000 00000010 00000001
- IP-aadress jaguneb kaheks: võrguosa ja hostiosa
  - Võrguosa bitis on sama L2 võrgu piires kõigil hostidel samad
  - Hostiosa bitid tagavad sama L2 võrgu piires unikaalsuse

# IP võrgumask

- Võrgumask näitabki, missugused bitid antud võrgus fikseeritud on
- Näiteks levinuim  $255.255.255.0 = 3$  baiti fikseeritud (/24)
- $255.255.254.0 = 11111111\ 11111111\ 11111110\ 00000000 = /23$
- $255.255.255.240 = 11111111\ 11111111\ 11111111\ 11110000 = /28$
- Igas võrgus on eritähendusega aadressid „kõik nullid“ (vanasti kasutusel leviaadressina) ja „kõik ühed“ (leviaadress tänapäeval)

# Võrgumaski lihtne peast arvutamine

- Olgu meil võrgumask 255.255.255.224 kujul, tahame teada bittide arvu ja seda, mitu hosti on antud võrgus (koos leviaadressidega)
- Arvutus on lihtsalt peast tehtav:
  - 255 ja 0 väärtused on triviaalselt kõik 1-d või kõik 0-d, need saab baidi kaupa kokku arvutada
  - Neist erinev maski bait  $224 = 256 - 32$
  - Seega on antud võrgus 32 eri IP-d –  $32 = 2^5$ , seega on hostiosa pikkus 5 bitti
  - Maskis jääb võrguosale seega sellest baidist  $8 - 5 = 3$  bitti
  - Seega on numbriliseks maskiks /27 (sest  $8 + 8 + 8 + 3 = 27$ ).

# ARP

- Kui L2 võrgus tahab arvuti IP-aadressiga Y arvutile IP-aadressiga X paketti saata, on vaja leida sihtarvuti MAC-aadress, et ainult talle saata teisi segamata
- Selleks on IPv4 juures ARP (Address Resolution Protocol)
- ARP päringu puhul kisab klient L2 leviaadressile päringu, et „Kes teist on X? Öelge seda palun Y-le.“
- Kui X enda kohta päringut kuuleb, vastab ta Y-le.
- Y saab nüüd X-le üle L2 IP-paketti saata.
- Lisaks hoiab Y seda kirjet mõne aja meeles oma ARP tabelis.

# IP (mars)ruutimine

- (Mars)ruuterid edastavad liiklust mitme võrgu vahel
- Paketi teekonna igal sammul otsustab selle sammu ruuter, kuhu pakett edasi saata
- Selleks on ruutingutabelid sihtvõrkudega, kust valitakse iga paketi puhul sobivate hulgast kõige pikema maskiga kirje, näiteks:

<b>sihtvõrk</b>	<b>mask</b>	<b>kuhu</b>
192.168.0.0	255.255.255.0	eth0
192.168.1.0	255.255.255.0	10.0.0.2
192.168.0.0	255.255.0.0	10.0.0.5
10.0.0.0	255.255.255.0	eth1
0.0.0.0	0.0.0.0	10.0.0.1

# IP-vahemike jaotamine

- Hierarhiline:
  - ISP saab suure ploki, näiteks /16 kuni /19
  - ISP jagab igale kliendile väiksema ploki, näiteks /24
  - Iga klient võib oma võrku jagada alamvõrkudeks, näiteks /28
- Klient võib osta otse teenusepakkujust sõltumatu IP ploki ja selle maailmale mitme ISP kaudu kättesaadavaks teha
  - Kõrgema käideldavuse jaoks
  - AS (Autonomous System)
  - osapool, kellel on otse oma teenusepakkujust sõltumatu IP-vahemik ja kes korraldab selle ühendamist teiste AS-ide kaudu

# Ruutinguinfo levitamine

- Staatileine ruuting — seadmesse konfigureeritakse ruutingutabel administraatori poolt
- Dünaamiline ruuting — ruuterid vahetavad omavahel infot kättesaadavate võrkude kohta ja arvutavad ise, kust kaudu mingi võrk kõige otsem kättesaadav on
- „Core routers“ — Interneti tuumik, mis koosneb ruuteritest, mis teavad kõigi võrkude asukohti ilma vaikeruutingut kasutamata
- Sisemised ruutinguprotokollid (IGP — interior gateway protocol) — kasutamiseks organisatsiooni sees optimaalse tee leidmiseks, näiteks RIP ja OSPF
- Välised ruutinguprotokollid (EGP — exterior gateway protocol) — kasutamiseks organisatsioonide vahel suuremate plokkide granulaarsusega, näiteks BGP

# IP-aadresside jagamine kohtvõrgus

- Staatileine — igale seadmele eraldatakse IP käsitsi ning konfigureeritakse seade seda kasutama
- Dünaamiline — seade saab käivitamisel omale ajutise aadressi automaatselt
- DHCP (Dynamic Host Configuration Protocol) — protokoll aadressi, domeeninime, staatiliste ruutingute ja muude parameetrite küsimiseks
- DHCP server teab, mis IP-aadress missugusele MAC-aadressile vastavuses on (seos võib olla ajutine või permanentne)

# ICMP

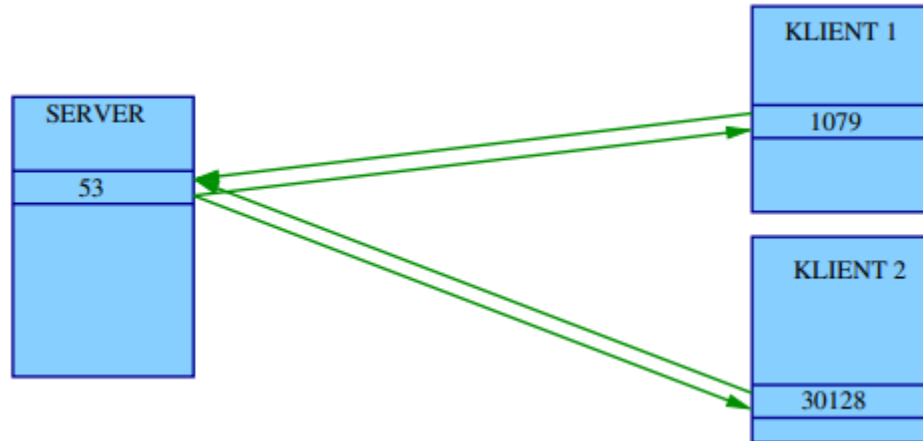
- Internet Control Message Protocol
- Ühelt arvutilt teisele saadetavate juhtsõnumite protokoll
- Näiteks:
  - echo request (ping)
  - echo reply (pong)
  - redirect (teine ruuter on otsem)
  - time exceeded
  - destination unreachable, network unreachable
  - destination unreachable, host unreachable
  - destination unreachable, administratively prohibited
  - destination unreachable, fragmentation needed but DF set (seda ei tohi filtreerida!)

# UDP

- User Datagram Protocol
- Lähtearvuti mingilt protsessilt sihtarvuti mingile protsessile teate saatmine • Lihtne (ainult fragmenteerimine)
- Pole töökindel (iga pakett võib kaotsi minna)
- Olekuvaba (ei mingit korduvsaatmist ega järjestust) • Kiire (ei vaja puhverdamist)
- Sobib ka ühesuunaliseks suhtluseks (leviedastuse ja multiedastuse puhul) • UDP ise ei sisalda ummistuste vältimist (congestion control)
- Sobib nii lihtsate päring-vastus tüüpi protokollide kui reaalaajavoogude jaoks

# UDP port

- Eristamaks arvutis mitut protsessi, on side kummaski otspunktis lisaks IP-aadressile kasutusel 16-bitised pordinumbrid
- Enamus UDP-d kasutavates rakendusprotokollides on teenuse pordinumber fikseeritud, kliendi pordinumber dünaamiline ning server vastab sellele kliendi pordile, kust päring tuli



# TCP

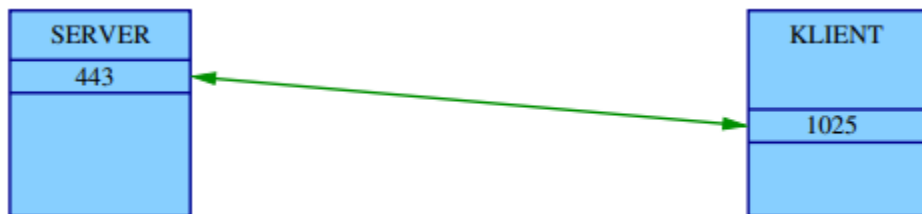
- Transmission Control Protocol
- Lähtearvuti mingilt protsessilt sihtarvuti mingile protsessile baidivoo kahesuunaline edastamine
- Pakettide piirid pole fikseeritud, tükeldus võib teel muutuda
- Töökindel (iga andmetükki kviteeritakse, kaotsi minekul saadetakse mingi aja pärast uuesti)
- Järjestatud (andmed jõuavad rakendusele kohale samas järjekorras nagu teisest rakendusest saadeti, TCP puhverdab vajadusel kuni vahepealsed andmed kah kohal on)
- Raskekaalulisem kui UDP — vajab ühenduse loomist andmete saatmiseks, tegeleb ummistuste vältimisega
- Tegeleb vookontrolliga (et üks rakendus ei saadaks rohkem kui teine jõuab vastu võtta)

# TCP detailid

- Selgelt ainult kahele osapoolele (ainuedastus)
- Ühenduse loomine: SYN, ACK, SYN+ACK
- Aknaga protokoll (kui palju andmeid on korraga teel?) vookontrolliks
- Kummaski suunas sõltumatute loenduritega baidivoog
- Slow start — ühenduse või iga järgmise suurema mahu saatmise algul hakatakse kiirust järjest kasvatama, kuni mõõdetakse ära teekonna läbilaskevõime

# TCP pordid

- Side kummaski otspunktis on protsesside eristamiseks lisaks IP-aadressile kasutusel 16-bitised pordinumbrid
- Kahe protsessi vahel võib olla mitu sõltumatut TCP ühendust erinevate kliendiportidega
- Igal teenusel on enamasti fikseeritud serveri pordinumber • Kliendi pordinumber on tavaliselt juhuslik



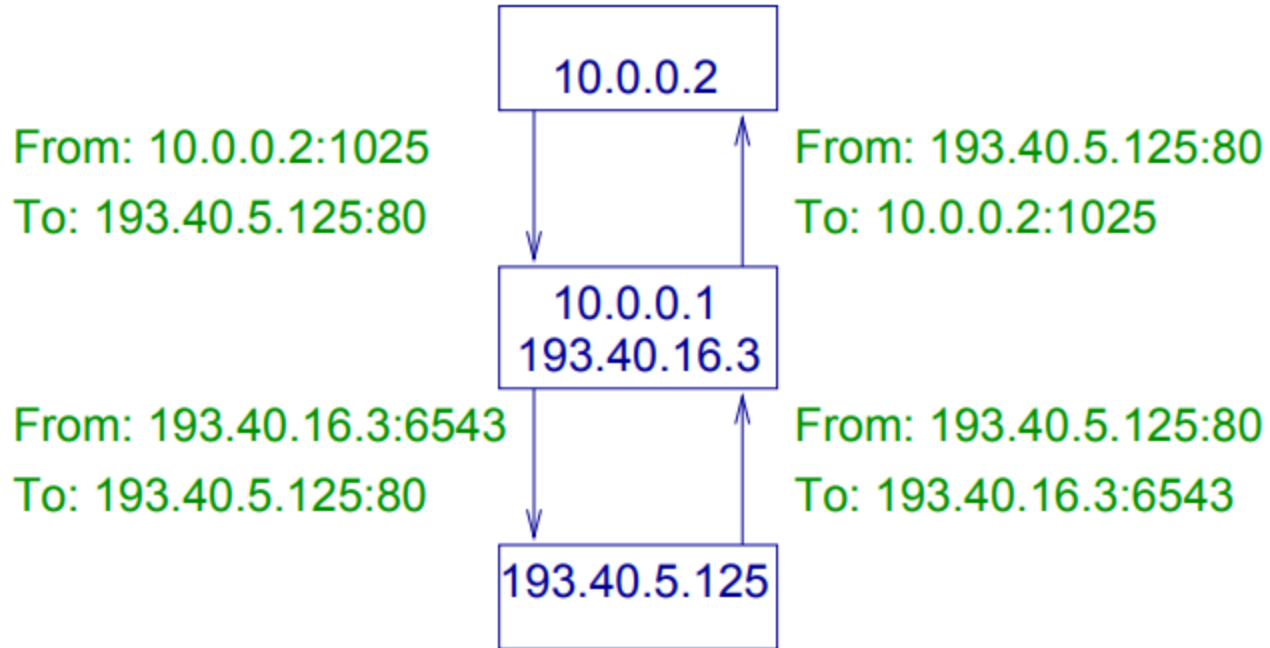
# Võrguaadresside tõlkimine — NAT

- IPv4 aadressidega on kitsas käes, vaja on aadresside kasutamist optimeerida
- Tahame sisevõrgu struktuuri teiste eest ära peita
- Tahame, et sisevõrgu masinad ei oleks väljast otse nähtavad
- Lahendus(?): kasutame sisevõrgus privaataadresse, mis Internetis ei esine •  
Vahel on siiski vaja pakette sise- ja välisvõrgu vahet liigutada
- Lahenduseks on aadresside tõlkimine ruuteris. Tõlkimist on kolme moodi:
  - Staatiline:  $n - n$  — tõlgitakse terve aadressiplokk
  - Dünaamiline:  $n - m$ ,  $m < n$  — avalikke aadresse on vähem
  - Tõlkimine porte kasutades:  $n - 1$  — kõik siseaadressid 1 välisaadressiks, varieeritakse lähtepordi numbrit

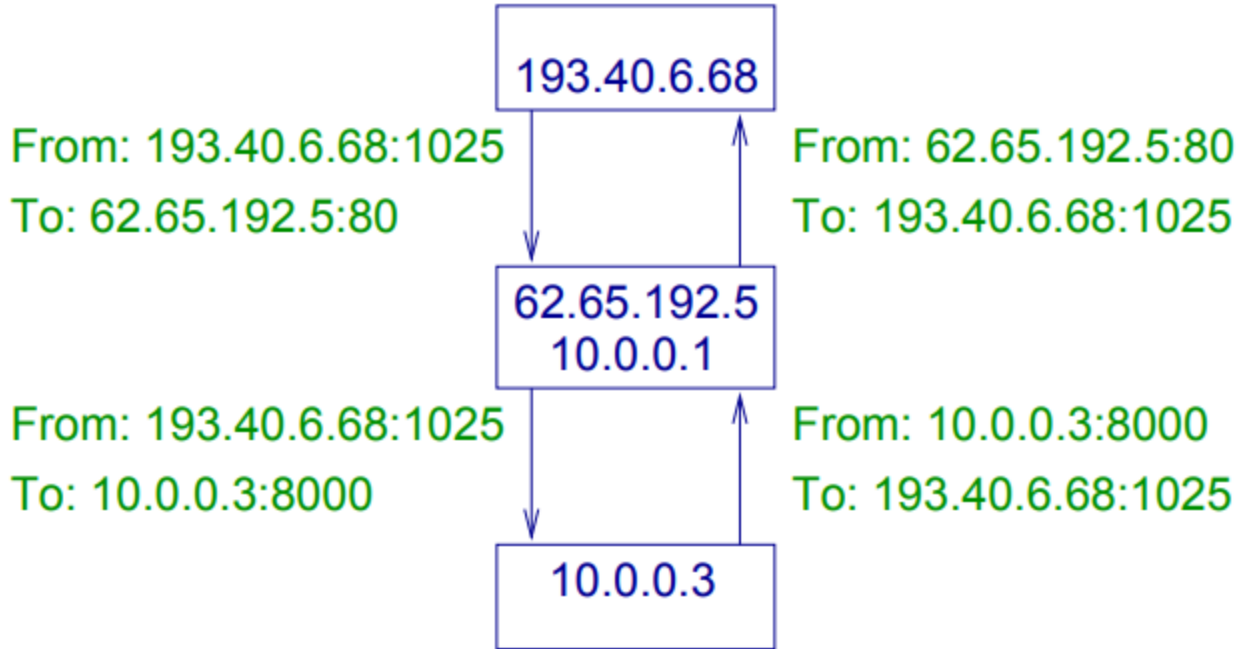
# NAT tehnoloogia

- Standarditega on paika pandud aadressivahemikud, mida võib vabalt oma sisevõrkudes kasutada:
  - 10.\*.\*
  - 172.16.\*.\*
  - 172.31.\*.\*
  - 192.168.\*.\*
- Neid aadresse Internetis ei ruudita
- Aadresse tõlkiv ruuter modifitseerib ühe osapoole IP aadressi (tõlgib ühe suuna andmed ning tõlgib tagasi vastused)
- Lähteadressi maskeerimise abil saame varjata klientarvutit (algatajat) → SNAT
- Sihtaadressi maskeerimise abil saame varjata serverarvutit → DNAT
- SNAT ja DNAT võib samas ruuteris ka järjest rakendada

# SNAT näide



# DNAT näide



# NAT probleemid

- Teeb katki TCP/IP mudeli, kus ainult ühenduse otspunktid teavad detaile
- Sunnib peale mingi osaliselt fikseeritud marsruudi otspunktide vahel
- Toob sisse ühe katki mineku punkti
- Toob sisse ühildumatuse paljude protokollidega
- Ei lahenda IPv4 aadresside kitsikust
- AGA:
  - Leevendab IPv4 aadresside kitsikust
  - Aitab lihtsalt ja praktiliselt võrku turvalisemaks teha

# IPv6

- Uus võrgukihi protokoll 128-bitiste aadressidega:
  - 2001:bb8:2002:2400:209:3dff:fe11:e8c5/64
  - 2002:5abf:a1ac:2::1/64 – ::1/128
- ARP asemel multiedastusel põhinev neighbour discovery
- Samad TCP, UDP ja enamus rakendustaseme protokolle
- Uued ICMPv6, DHCPv6
- Lisaks DHCPv6-le võimalus olekuvabalt aadresse konfigureerida
- Igale kohtvõrgule /64, ruuter reklaamib prefiksit
- Host paneb oma aadressi kokku prefiksist ja 64-bit kohalikust osast (MAC või juhuslik)

# IPv6 üleminek

- Ülemineku ajaks võetakse enamus võrkudes kasutusele nii IPv4 kui IPv6 aadressid
- Protokollid IPv6 tunneldamiseks üle IPv4 (automaatne tunneldamine: 6to4, Teredo, ISATAP)
- API tasemel suudab IPv6 sokkel teenindada ka IPv4 ühendusi
  - IPv4-mapped aadressid ::193.40.36.2 kujul IPv4-aadresside tähistamiseks API-s (aga mitte „traadil“)
  - Vastupidi ei saa
- IPv6 puhul on NAT tugevalt vastunäidustatud — aadresse jätkub, vältime NAT probleeme kui võimalik

# Soklid

- Sokkel (socket) — programmeerimisliides (API) võrguga suhtlemiseks
- Sokkel on sidekanali otspunkt (näiteks TCP ühenduse kummaski otsas on vastaval rakendusel sokkel)
- Sokli-API on protokollist sõltumatu, disainitud OSI mudeli järgi
- Pärit BSD Unixist, tänapäeval üldlevinud (+kohalikud täiendused eri OS-ides)
- Toetab palju protokolliperekondi
- Toetab voosokleid (stream socket) ja paketisokleid (datagram socket)
- Blokeeruvad ja mitteblokeeruvad soklid
- Lisaks nimelahendus (gethostaddr() jt, Interneti puhul kasutab DNS)

# Soklite näited

- Kombineerime protokolliperekonna ja sokli tüübi:
  - PF\_INET, SOCK\_STREAM — TCPv4
  - PF\_INET6, SOCK\_STREAM — TCPv6
  - PF\_INET, SOCK\_DGRAM — UDPv4
  - PF\_UNIX, SOCK\_STREAM — Unixi sisene stream-sokkel protsesside vahel
  - PF\_RAW, SOCK\_DGRAM — Etherneti kaadrid toorkujul

# Soklite API

- `socket()`
- `connect()`
- `send()`, `recv()`
- `sendto()`, `recvfrom()`
- `shutdown()`, `close()`
- `bind()`
- `listen()`
- `accept()`
- `ioctl()`

# Soklinäide: TCP

<b>server</b>	<b>klient</b>
socket()	
bind()	
listen()	
	socket()
	connect()
accept()	
send()	recv()
recv()	send()
...	...
close()	close()

# Soklinäide: tavaline UDP

<b>server</b>	<b>klient</b>
socket()	
bind()	
	socket()
	bind()
recvfrom()	sendto()
sendto()	recvfrom()
...	...
	close()

# Soklinäide: UDP ja sokli ühendamine

<b>server</b>	<b>klient</b>
socket()	
bind()	
	socket()
	connect()
recvfrom()	send()
sendto()	recv()
...	...
	close()