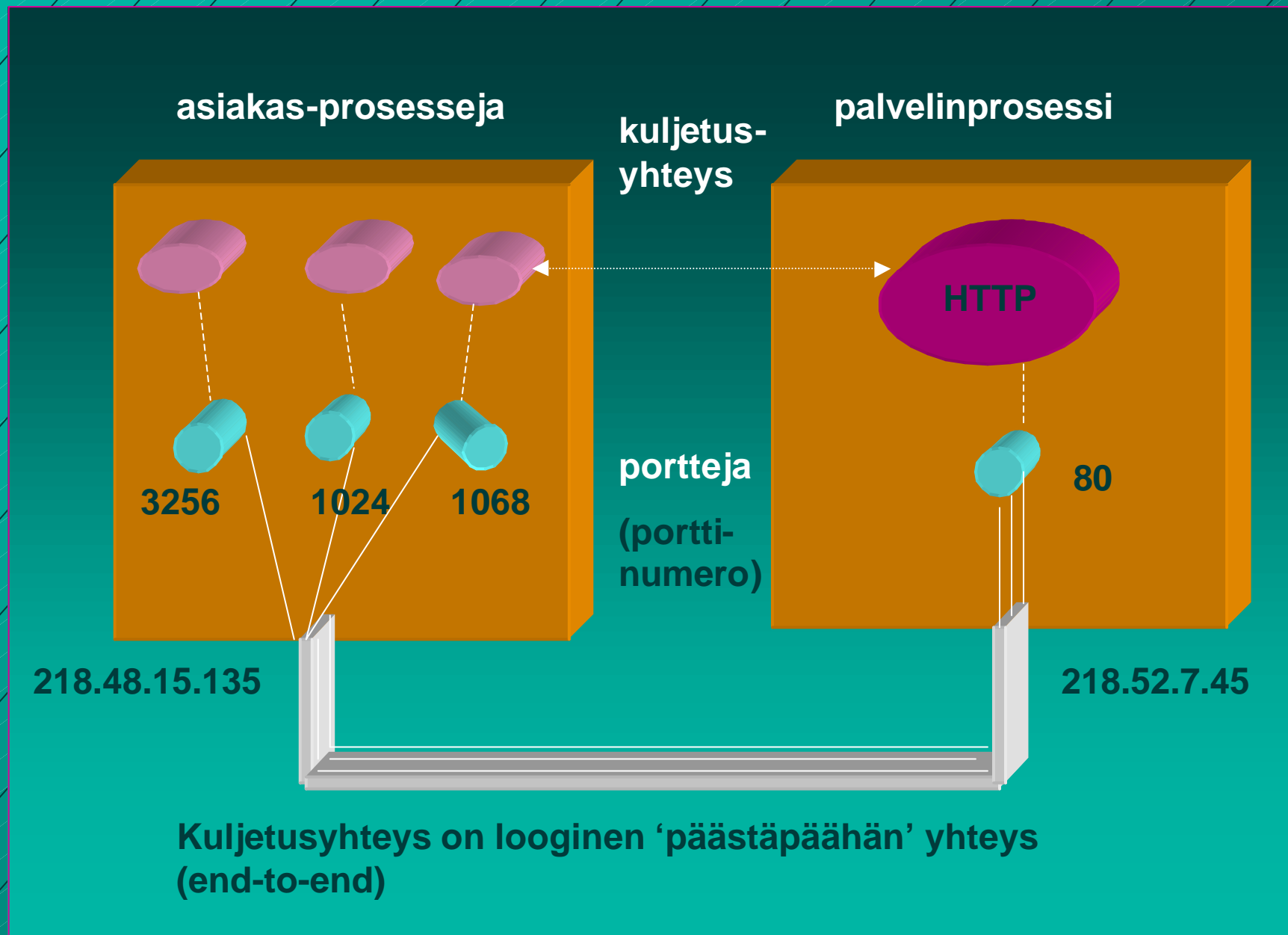


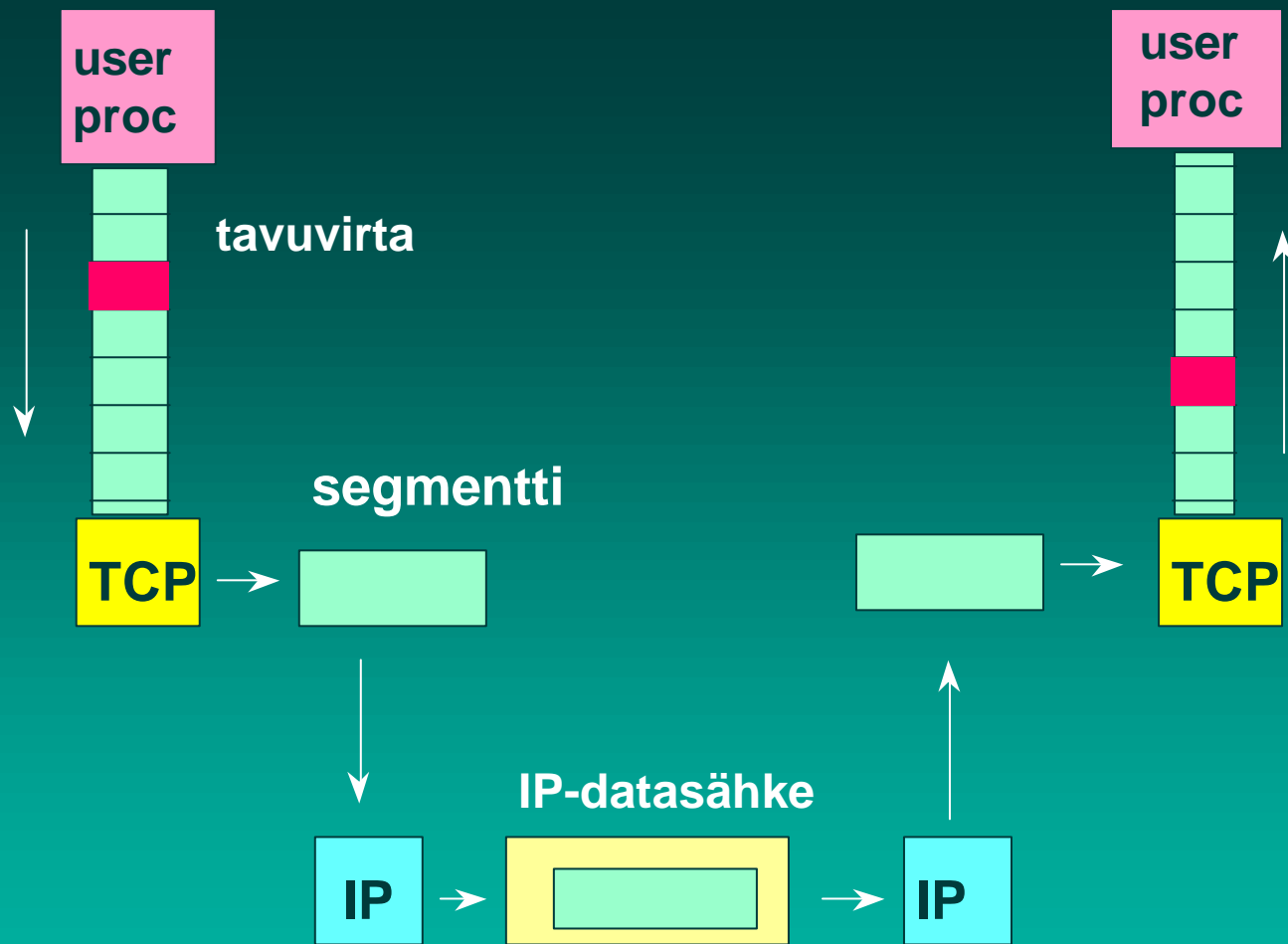
# 6. Kuljetuskerros

## 6.1. Kuljetuspalvelu

- 'End- to- end'
  - prosessilta prosessille
    - portti
  - verkkokerros koneelta koneelle
    - IP-osoite
- peittää verkkokerroksen puutteet
  - jos verkkopalvelu ei ole riittävän hyvä, sitä voidaan parantaa kuljetuskerroksella
    - kuljetuskerros huomaa verkkokerroksen kadottamat paketit ja pyytää niiden uudelleenlähetyistä



**Kuljetusyhteys on looginen 'päästäpäähän' yhteys (end-to-end)**



## Päästä-päähän tavuvirta

# kuljetuspalvelut parantavat verkkopalveluja

Sovelluksen näkemä QoS

kuljetuskerroksen palvelut

verkkokerroksen palvelut

kuljetuskerroksen palvelut

verkkokerroksen palvelut

## Vaatimuksia kuljetuspalvelulle

- Virheetön, luotettava
- järjestyksen säilyttävä
- kaksoiskappaleet karsiva
- mielivaltaisen pitkiä sanomia salliva
- vuonvalvonnan mahdollistava

## Verkkokerros kuitenkin

- kadottaa sanomia
- toimittaa sanomat epäjärjestyksessä
- viivyttaa sanomia satunnaisen pitkän ajan
- luovuttaa useita kopioita samasta sanomasta
- rajoittaa sanomien kokoa

## 6.2. Kuljetuskerroksen toiminta

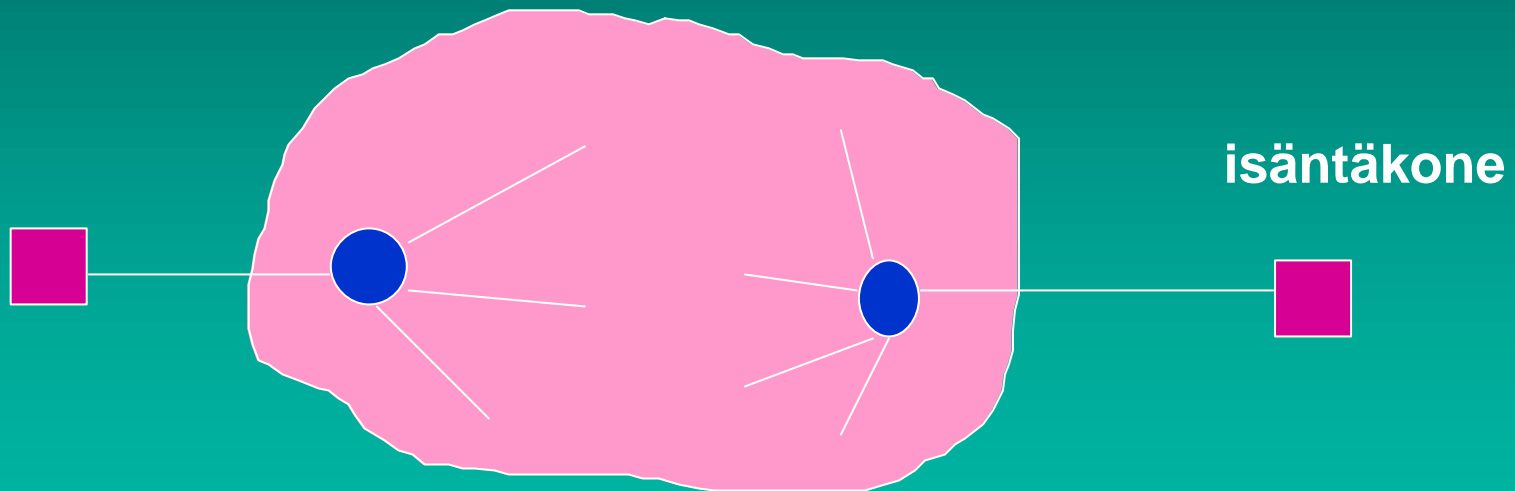
- monella tavoin siirtoyhteyskerroksen kaltainen
  - virhevalvonta
  - vuon valvonta jne
- mutta
  - siirtoyhteys: välissä fyysinen kanava
  - **kuljetusyhteys: välissä verkko**

## Siirtoyhteyskerros



reititin

## Kuljetuskerros



isäntäkone

# Verkko lisää monimutkaisuutta:

- kohteelle aina osoite
- yhteyden muodostus ja purku monimutkainen
- **verkko säilyttää sanomia**
  - sanoma voi jäädä verkkoon pitkäksi aikaa
  - vanha paketti voi ilmaantua paljon myöhemmin
- yhteyksiä reitittimen kautta on paljon ja ne muuttuvat dynaamisesti
  - vuonvalvonta ja puskurointi hankalaa
- ruuhka voi lamaannuttaa verkon => ruuhkan valvonta



## 6.2.1. Osoittaminen

- **Kuljetushteydenmuodostuksessa tiedettävä kuljetusosoite**
  - osoite on kaikkien tuntema ('häätännumero 112')
  - katsotaan luettelosta tai kysytään nimipalvelijalta (name server) ('numerotiedustelu 118')
- **Verkkokerroksen yhteyttä muodostettaessa tiedettävä kuljetusosoitetta vastaava verkko-osoite**
  - hierarkkinen kuljetusosoite paljastaa
    - <maa><verkko><isäntäkone><portti>
  - toinen nimipalvelija:
    - kuljetusosoite => verkko-osoite

## 6.3 Internetin kuljetusprotokollat

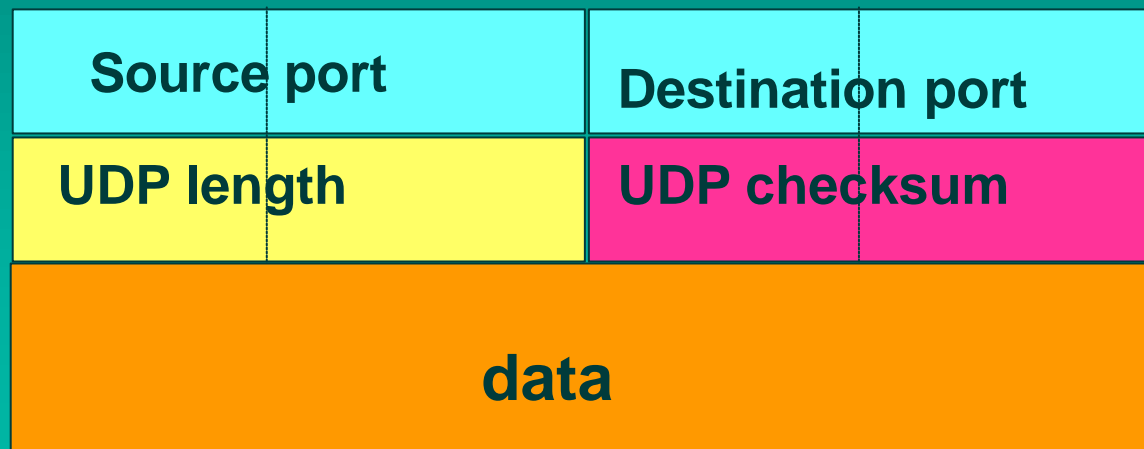
- **UDP** (User Datagram Protocol)
  - minimaalinen kuljetuspalvelu (best effort)
    - ei pyri parantamaan verkon tarjoamia palveluita
      - yhteydetön
      - kuljettaa sanomia prosessilta toiselle
      - ei takaa mitään: virheettömyyttä, järjestystä, jne
- **TCP** (Transmission Control Protocol)
  - luotettava tiedonsiirto
    - numerointi, kuittaukset, tarkistukset, ajastimet
    - vuonvalvonta, **ruuhkanvalvonta**

## 6.4.8. UDP

### ■ UDP (User Data Protocol)

- voidaan lähettää sanomia ilman yhteyden muodostusta

#### UDP-otsake



# TCP-protokolla

- yhteyden muodostus ja purku
- luotettavan tavuvirran toteuttaminen
- vuonvalvonta
- siirron optimointi
- TCP-segmentti
- ruuhkan valvonta
- TCP-palvelun käyttö

## 6.2.2. Yhteyden muodostus ja purku TCP:ssä

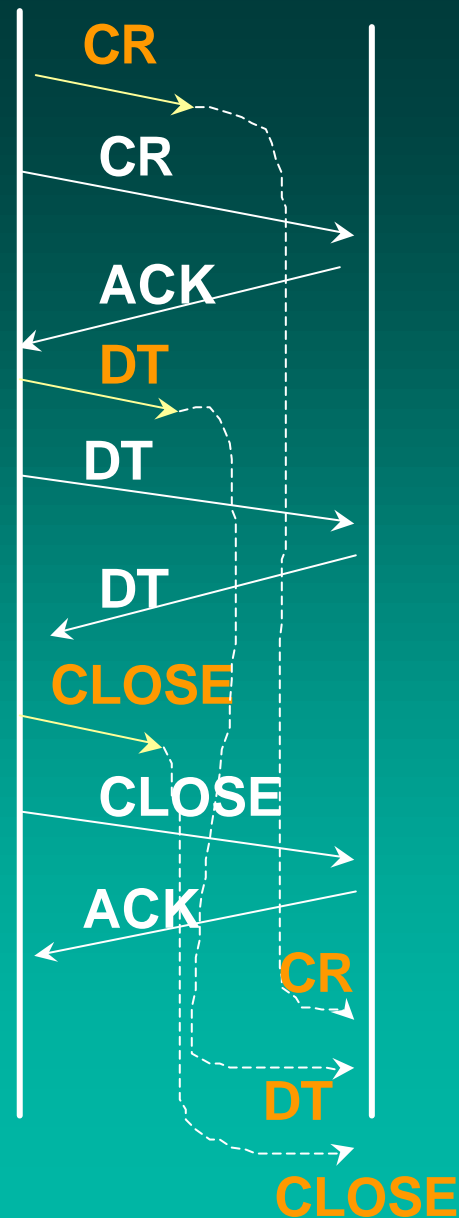
- TCP käyttää yhteyden muodostamiseen ja purkuun ns. **kolminkertaista kättelyä** (three-way handshake)
  - välissä oleva verkko tekee yhteyden muodostamisen ja purun hankalaksi
    - viivästyneet sanomat => sanomille elinaika
    - sanomien numeroinnista sopiminen
  - Bysanttilainen ongelma (two-army problem)
    - “hyökkään, jos olen varma, että sinäkin hyökkäät”
    - symmetrinen yhteyden purku = molemmat osapuolet tietävät, että toinenkin on varmasti purkanut yhteyden

## Yhteyden muodostus ruuhkaisessa verkossa

Jokainen paketti lähetetään kahteen kertaan

Kun yhteys on purettu, viivästyneet kaksoiskappaleet saapuvat

Ne tulkitaan uudeksi yhteydeksi, ja data otetaan vastaan kahteen kertaan!



**SYN =  
tahdistus-  
sanoma**

**SYN, Seqnro=x**

**SYN+ACK, Seqnro=y,  
ack=x+1**

**ACK, Seqnro=x+1,  
ack=y+1**

**Yhteyden muodostus**

**Kolminkertainen  
kättely**

**yhteyspyynnössä  
pyytäjän nro x**

**vahvistuksessa  
sekä pyytäjän  
että suostujan  
järj.numero**

**ensimmäisessä  
datalähetyksessä  
molemmat  
numerot**

#1

#2

Hyökätään aamulla  
kello 5!

OK, siis kello 5!

OK!

#2 hyökkää vain , jos  
tietää minun saaneen  
vastaussanomaa.

Entä, jos vastaus  
ei mene perille?  
Silloin #1 ei hyökkää!

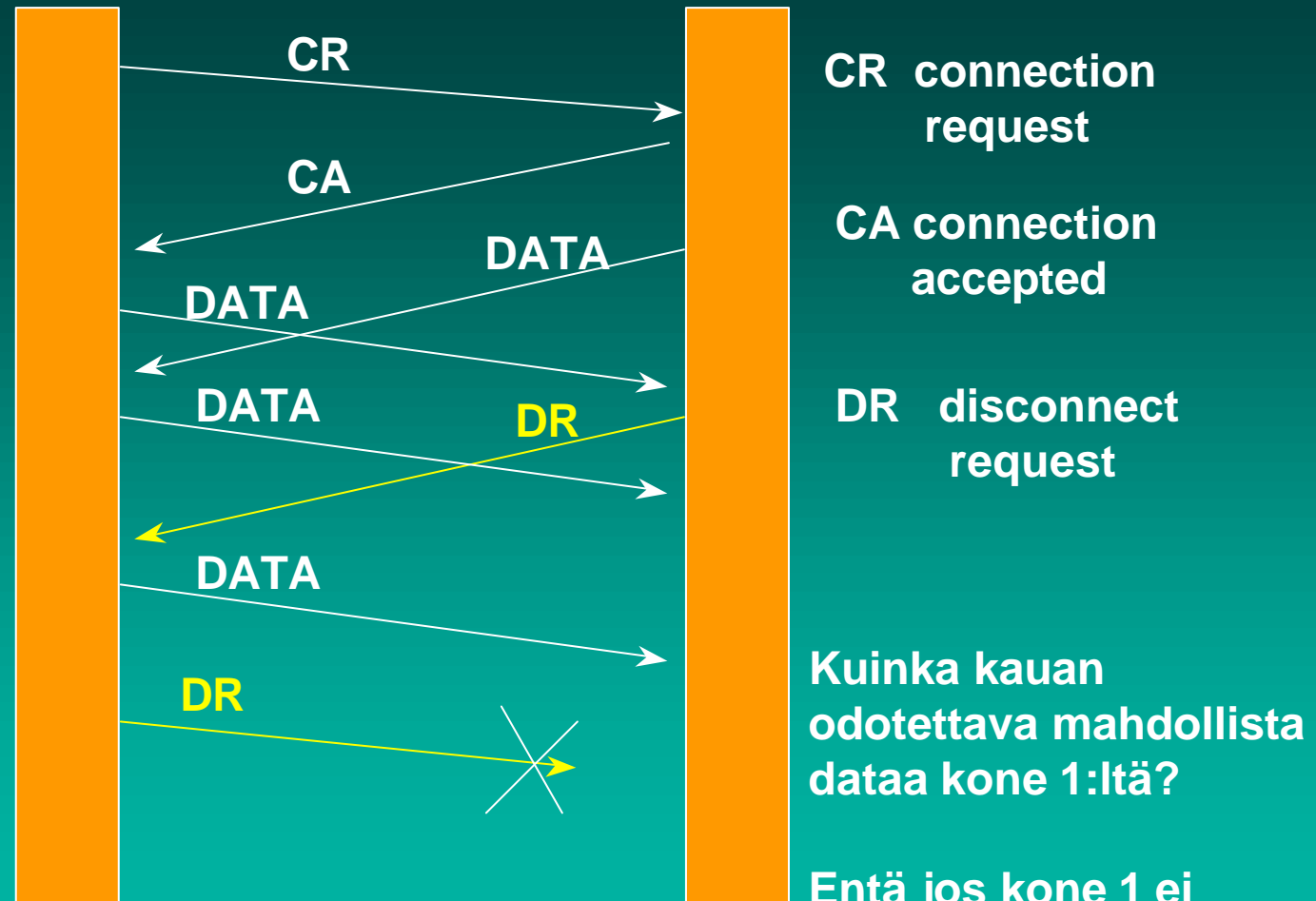
**Loogisesti ratkeamaton ongelma.**  
Kaikki riippuu aina viimeisestä sanomasta,  
jonka perillemeno ei voida taata!

**Bysanttilainen ongelma (two-army problem)**



Kone 1

Kone 2



Symmetrinen yhteyden purku

# Yhteyden purku

- molemmat suunnat puretaan erikseen
- TCP-segmentti
  - FIN = 1
    - ei enää dataa lähetettävä
    - kun saadan kuittaus => yhteys tähän suuntaan purettu
    - yhteys kokonaan purettu, kun molemmat suunnat purettu
- purussa käytetään ajastimia
  - $2 * \text{paketin maksimaalinen elinikä}$

**Kone 1**

**lähetä DR  
(=lopetuspyyntö)  
ja aseta ajastin**

**pura yhteys**

**lähetä ACK**

**FIN-lippu  
päällä**

**DR**

**DR**

**ACK**

**Kone 2**

**lähetä DR ja  
asetta ajastin**

**pura yhteys**

**Yhteyden purku kolminkertaista kättelyä käyttäen**

# Virheettömyys ja järjestys

## ■ Järjestysnumerot

- tavuvirta => tavunumerointi
- segmentin 1. tavun järjestysnumero
- yhteyden alussa satunnaiset numerot

## ■ kuittaukset

- kumulatiivinen ACK, ei NAK-kuittausta
- kuittauksessa seuraavaksi odotettava tavu

## ■ Go Back N

- virheelliset tai väärässä järjestyksessä tulleet tuhotaan
- myös valikoiva toisto mahdollinen

# TCP:n vuonvalvonta

- 'joustava' liukuva ikkuna (sliding window) (credit-vuonvalvonta)
- vastaanottaja kertoo, kuinka paljon suostuu vastaanottamaan
  - => kuittaus irroitettu vuonvalvonnasta
    - AdvertisedWindow-kenttä
      - paljonko saa lähettää = paljonko vastaanottajan puskureihin mahtuu
- myös ruuhkan valvonta rajoittaa lähettämistä

## Esimerkki

A

B

<ehdottaa 8 puskuria >



<ack = 0, buf = 4>



<seq = 0, data = m0 >



<seq = 1, data = m1 >



<seq = 2, data = m2 >



<ack = 1, buf = 3>



<seq = 3, data = m3 >



<seq = 4, data = m4 >



lupa vain  
sanomille 0-3

kuittaus sanomista  
0 ja 1, lupa  
sanomille 2- 4,

puskurit käytetty,  
A joutuu lopettamaan

## Esimerkki jatkuu

**A**

ajastin laukeaa,  
uudelleen sanoma 2

<seq = 2, data = m2> →

← <ack = 4, buf = 0>

← <ack = 4, buf = 1>

← <ack = 4, buf = 2>

lähettää sanoman 5

<seq = 5, data = m5> →

lähettää sanoman 6

<seq = 6, data = m6> →

← <ack = 6, buf = 0>

jos lupa katoaa, jää  
odottamaan!

==> lukkiutumistilanne



← <ack = 6, buf = 4>

**B**

kuittaa kaikki,  
mutta ei anna lupaa  
lähettää

lupa lähettää yksi  
sanoma (= 5)

lupa lähettää kaksi  
sanomaa (= 5 ja 6)

kuittaa, mutta ei  
anna lähetyslupaa

lähetyslupa  
sanomille 7-10

- jos ilmoitus lisäpuskureista katoaa, lähettäjä lukkiutuu odotustilaan
  - vastaanottaja voi luulla, ettei ole lähetettävää
- lukkiutumisen estämiseksi
  - kun ikkunankoko = 0 lähettäjä ei saa lähettää, paitsi
  - pikadataa (URG)
  - yhden tavun 'kyselyn', jonka vastaanottaja kuittaa ja samalla ilmoittaa ikkunan koon
    - ⇒ estää turhat lukkiutumiset



## 6.4.5 Siirron optimointi

- TCP saa optimoida lähettämisiään
  - ei tarvitse lähettää heti kun data on tullut
  - dataa kerätään puskuriin ja lähetetään sopivassa tilanteessa
  - PUSH-lipun avulla sovellus ilmoittaa, että data on lähetettävä heti

## Optimointi on usein tarpeen:

- Interaktiivinen editori => merkki lähetetään heti
  - 21 tavun TCP-segmentti => 41 tavun IP-paketti
  - joka kuitataan 40 tavun IP-paketilla
  - ilmoitus uudesta ikkunan koosta 40 tavun IP-paketilla
  - kaiutetaan merkki vielä 41 tavun IP-paketilla
- yhden merkin käsittely =>
  - 162 tavun siirtäminen
  - ja neljän segmentin lähettäminen

## ■ Ratkaisu: Naglen algoritmi

– jos data tulee tavuttain

- lähetä 1. tavu

- kerää sitä seuraavat tavut puskuriin ja lähetä vasta kun edellinen lähetys on kuitattu

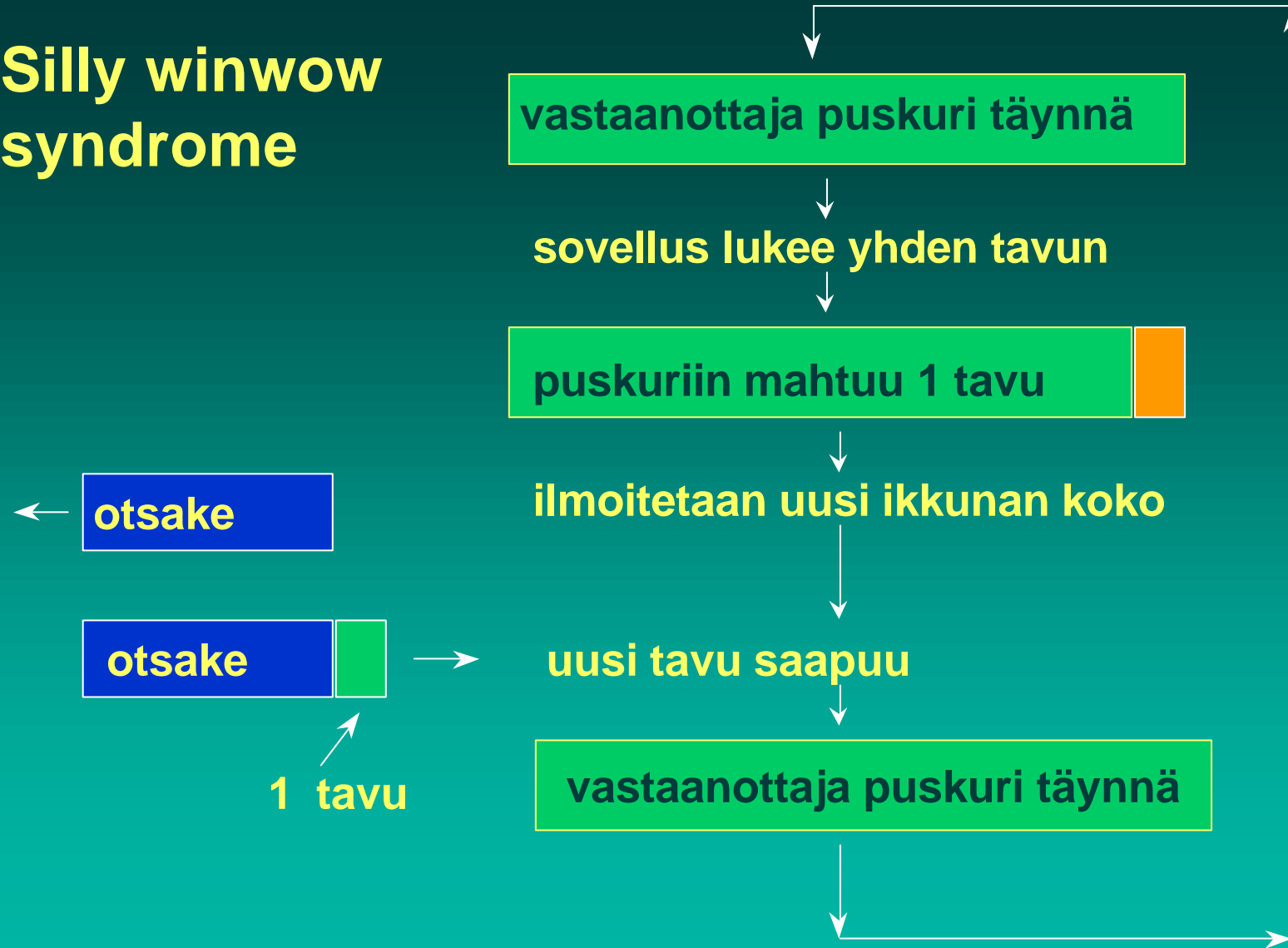
- paitsi jos lähetettävää on suurimman segmentin verran tai puolet ikkunan koosta

– hankala, jos hiirtä liikutellaan Internetin kautta!

# Silly window syndrome

- Tilanteessa, jossa
  - lähettäjältä dataa TCP:lle suurina lohkoina
  - vastaanottajalle vain tavu kerrallaan
- voi tuhota TCP:n suorituskyvyn
  - koko data lähetetään tavu kerrallaan
  - joka tavun välissä ilmoitus ikkunan koon kasvattamisesta yhdellä
- Siis: ei ilmoitusta yhdestä tavusta, lähettäjä ei lähetä yhtä tavua
  - koko segmentti
  - puolet puskurin koosta

# Silly window syndrome



## 6.4.2. TCP-segmentti

### ■ segmentti

- 20 tavun otsake
  - + optionaalinen osa
- dataosa
  - voi puuttua

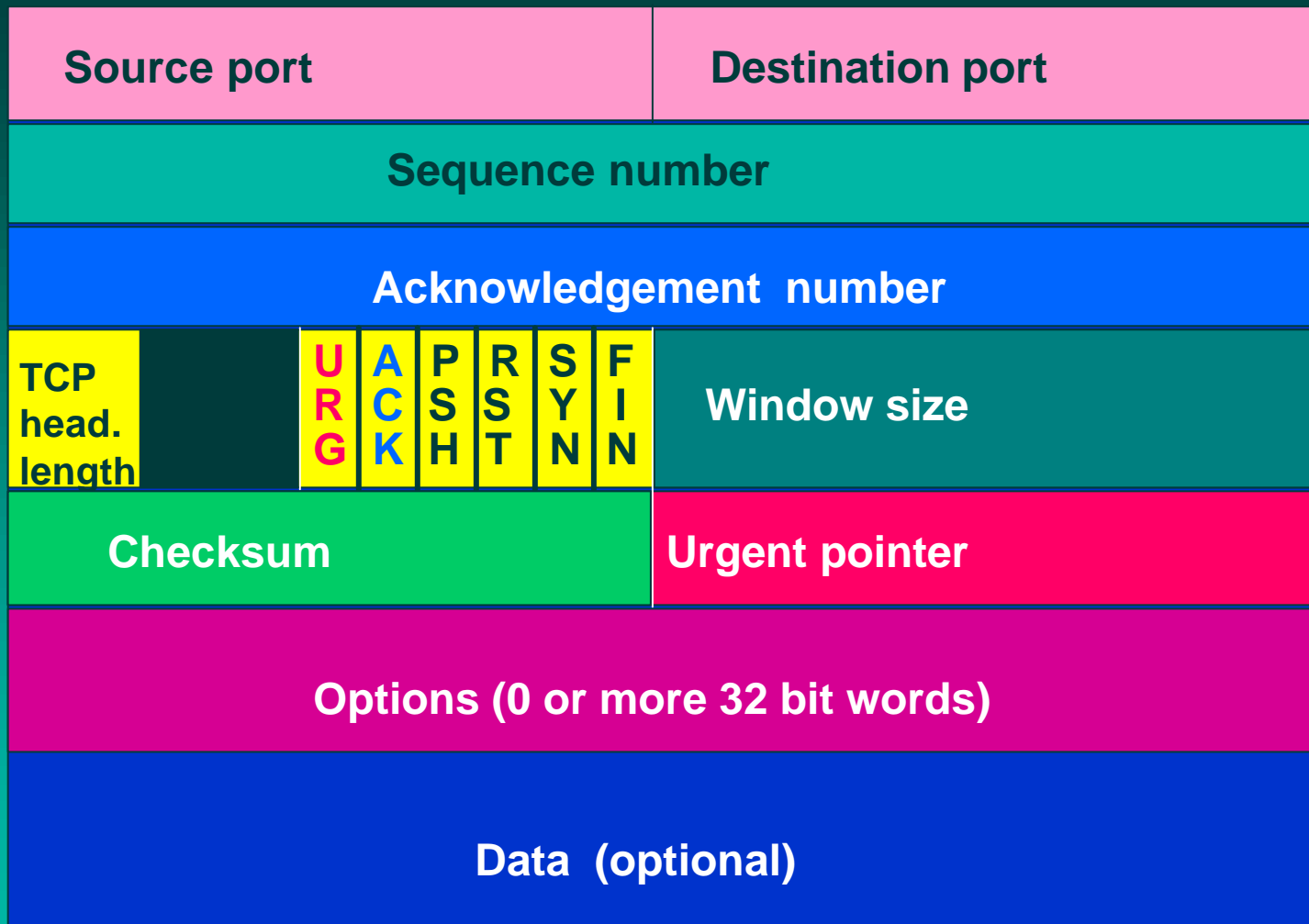
### ■ segmentin kokoa rajoittaa

- MTU (Maximum transfer unit)
  - verkon rajoitus maksimikoolle (muutama tuhat tavua)
- IP-paketin dataosa korkeintaan 65535 tavua

### ■ liian isot segmentit paloitellaan

- joka palalle IP-otsake => yleisrasite kasvaa

## TCP-otsakkeen kentät



# TPC-segmentin otsakekentät

- **Lähde- ja kohdeportit** (Source port, Destination port)
  - yhteyden päätepisteet
  - portti + koneen IP-osoite => 48 bittinen TSAP
- **Järjestysnumero** (Sequence number)
  - tavut numeroidaan => 32 bittiä
    - segmentin ensimmäisen tavun numero
- **Kuittausnumero** (Acknowledgement number)
  - seuraavaksi odotettu tavu
- **TCP-otsakkeen pituus** (TCP header length)
  - mahdollisten optiokenttien takia
- **6 bitin käyttämätön kenttä**



## ■ 6 lippubittiä

- **URG** onko pikadataa  
pikadatan sijainnin ilmoittaa  
**pikadatakenttä** (Urgent pointer)
- **ACK** onko kuittauskenttä käytössä
- **PSH** onko hetilähetettävää (pushed) dataa
- **RST** yhteyden uudelleenalustuspyyntö (reset), yleensä ongelmatilanne
- **SYN** käytetään yhteyttä muodostettaessa  
**SYN = 1, ACK = 0** connection request  
**SYN = 1, ACK = 1** connection accepted
- **FIN** käytetään yhteyden purkuun  
**FIN = 1** ei enää lähetettävää

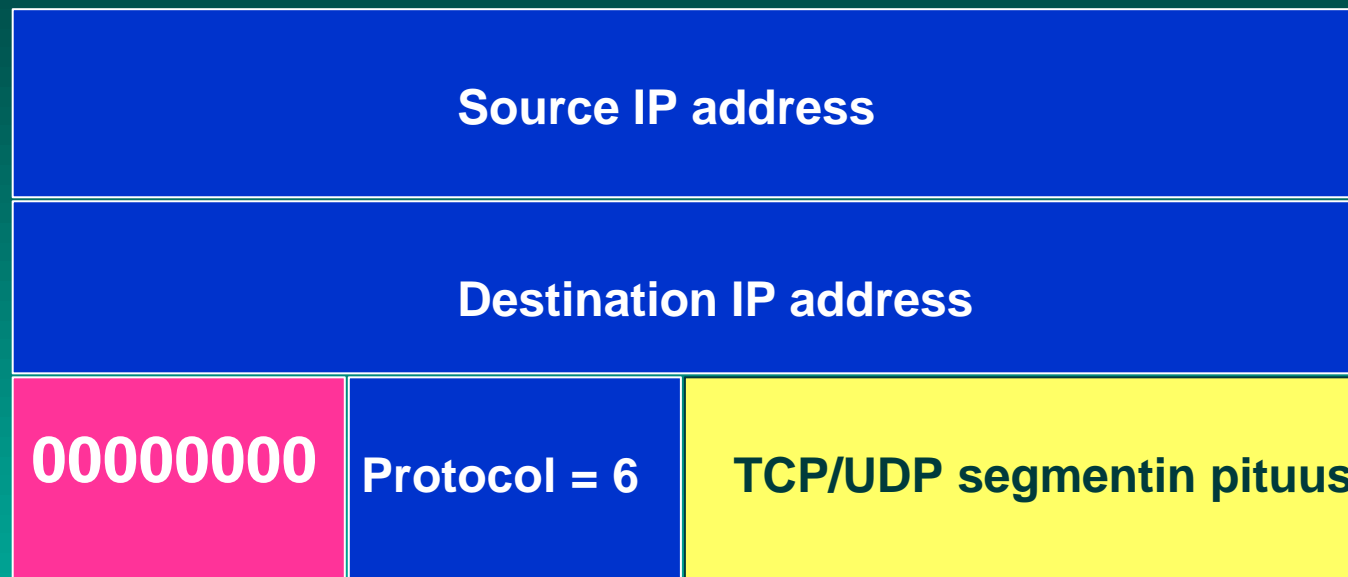
- **Ikkunan koko** (window size)

- vaihteleva ikkunankoko
- kuittaus irroitettu lähetysohjeesta

- **Tarkistussumma** (Checksum)

- lasketaan otsakkeelle, datalle ja ns. pseudo-otsakkeelle

# pseudo-otsake



**Auttaa havaitsemaan väärään osoitteeseen toimitetut paketit.**

**Sisältää IP-otsakkeen tietoja!**

## ■ **Optiokenttä** (options)

– voidaan lisätä piirteitä, joita ei ole varsinaisessa otsakkeessa

■ **suurin hyväksyttävä datakenttä**

■ **ikkunan koon moninkertaistaminen** (window scale)

– nopeille ja pitkän viipeen linjoille 64 ktavun ikkunan koko on liian pieni

■ **valikoivan toiston käyttö** 'go back N':n tilalla

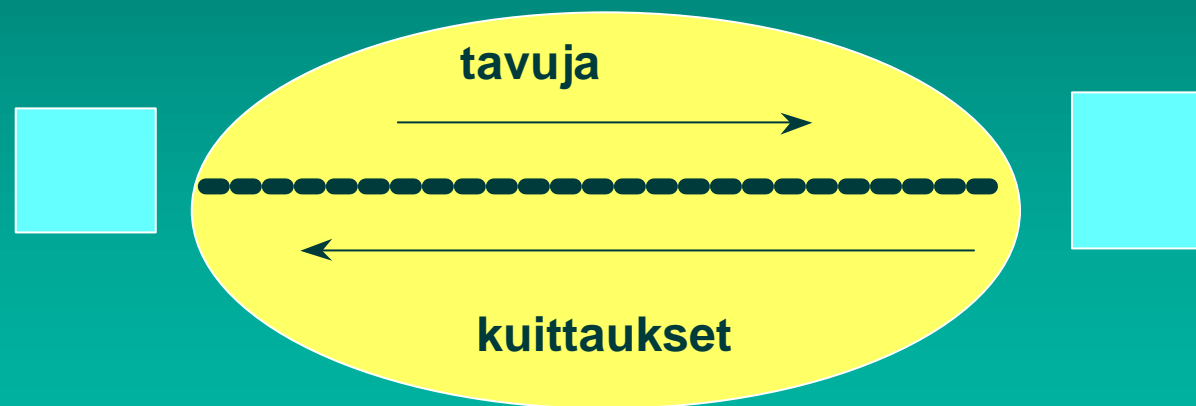
– vähentää turhia uudelleenlähetyksiä

## 6.4.6. TCP:n ruuhkan valvonta

- Liikaa kuormitusta => verkko ruuhkautuu => hidastetaan lähettämistä
- Ruuhkan havaitseminen
  - nykyisin siirtovirheet harvinaisia
    - poikkeuksena langattomat verkot
  - => uudelleenlähetykset johtuvat ruuhkasta
    - uudelleenlähetyksajastimen laukeaminen on merkki ruuhkasta

## ■ ruuhkaikkuna

- “paljonko tavuja (segmenttejä) lähettäjällä saa korkeintaan olla verkossa liikkeellä”
- kuittaus => ko. tavut jo poistuneet verkosta



## ■ Ruuhkaikkunan koko?

- Lähettäjän on itse pääteltävä ja arvioitava sopiva ruuhkaikkunan koko
  - kukaan muu ei sitä kerro!
  - timeout => on ruuhkaa
  - kuittaukset tulevat tasaisesti => ei ole ruuhkaa

## ■ Dynaaminen ruuhkaikkunan koko:

- ruuhkaikkunaa kasvatetaan kunnes törmätään ruuhkaan
- sen jälkeen ruuhkaikkunaa pienennetään reilusti
- ja aletaan uudestaan kasvattaa ruuhkaikkunaa

## Hitaan aloituksen algoritmi (slow start)

- Algoritmi pyrkii löytämään sopivan ikkunan koon yhteyden alussa tai ruuhkatilanteen jälkeen mahdollisimman nopeasti
  - ei ole niin kovin hidas, vaan alussa eksponentiaalinen!
    - alussa ruuhkaikkuna = yksi segmentti
    - kuitattu ruuhkaikkunallinen kasvattaa ruuhkaikkunan kaksinkertaiseksi

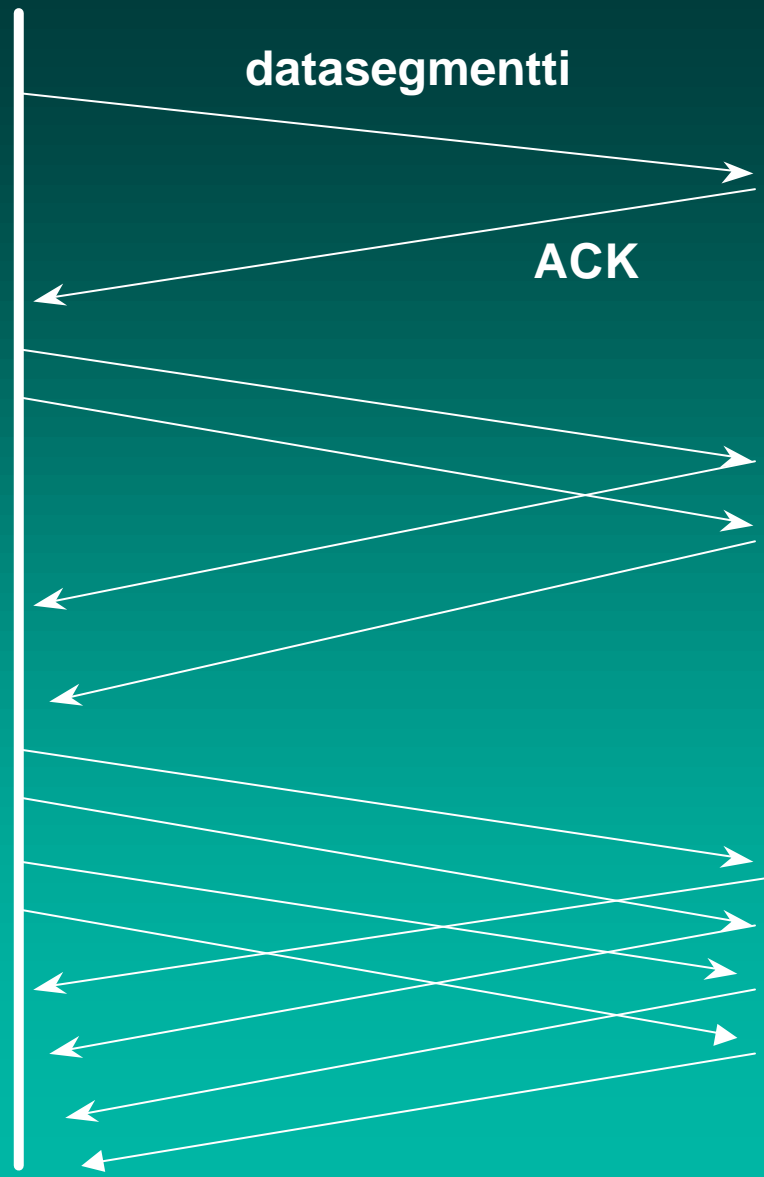


**lähettäjä**

**datasegmentti**

**vastaanottaja**

**ACK**



## ■ kynnysarvo (threshold)

- 'varoituservo' = tästä lähtien syytä varoa ruuhkaa
- aluksi 64 K
- kynnysarvoon saakka voidaan kasvattaa ruuhkaikkunaa eksponentiaalisesti
- kynnysarvon saavuttamisen jälkeen kasvatetaan ruuhkaikkunaa vain lineaarisesti
  - = kasvatetaan kuittausten jälkeen vain yhdellä
  - edetään hyvin varovaisesti!

## ■ jos ajastin ehtii laueta => ruuhkatilanne

- kynnysarvoksi puolet nykyisestä ruuhkaikkunan arvosta
- hitaalla aloituksella etsitään taas uusi sopiva ruuhkaikkunan arvo
  - ruuhkaikkunan arvoksi 1 segmentti
  - ruuhkaikkunaa kasvatetaan aluksi eksponentiaalisesti eli kaksinkertaistetaan kun ikkunallinen on kuitattu
- kynnysarvon saavuttamisen jälkeen kasvatetaan vain segmentti kerrallaan
- kunnes taas havaitaan ruuhka ja aloitetaan ruuhkaikkunan uuden arvon etsiminen

## Uudelleenlähetyksajastimen hallinta

- uudelleenlähetyksajastin (retransmission timer)
  - asetetaan aina kun segmentti lähetetään
  - ruuhkaa, jos kuittaus ei saavu ajoissa
- mikä on sopiva ajastimen aika?
  - kuittaus aika vaihtelee suuresti
  - vaihtelu on myös nopeaa
- dynaaminen arvo
  - saadaan jatkuvien verkon suorituskykymittauksien perusteella

## ■ RTT

- arvio kiertoviiveelle (round-trip time)
- mitataan jokaisen lähetetyn segmentin kiertoviive M

$$RTT = \alpha RTT + (1-\alpha)M, \text{ tyypillisesti } \alpha = 7/8$$

## ■ uudelleenlähetyksajastimen arvo $\beta RTT$

- aluksi  $\beta$  oli aina 2
- parannus: otetaan huomioon myös poikkeama D (deviation) oletetun ja saadun kiertoviiveen välillä  $|RTT-M|$

$$D = \alpha D + (1-\alpha)|RTT-M|$$

- ajastimen arvo =  $RTT + 4 * D$

- uudelleenlähetyksen vaikutus ajastimeen

- kumpaan segmenttiin kuittaus kohdistuu?

- Karnin algoritmi

- ei oteta huomioon uudelleenlähetyksen segmenttien kuittauksia RTT:n laskemisessa

# Parannuksia ruuhkanvalvontaan

- Nopea uudelleenlähetyks (Fast Retransmit)
  - ei odoteta ajastimen laukeamista ennen uudelleenlähetystä
  - vastaanottaja kuittaa jokaisen paketin
  - kun vastaanottaja huomaa puuttuvan paketin, se lähettää uudelleen edellisen paketin kuittauksen
    - Duplicate ACK (~ NAK)
  - kun lähettäjä saa useita (3 ) peräkkäisiä saman paketin kuittauksia => se havaitsee tästä paketin puuttuvan ja lähettää sen heti uudelleen
  - => nopeampi uudelleenlähetyks

## ■ Nopea toipuminen (Fast Recovery)

– kun kadonnut paketti huomataan nopealla toipumisella ei aloiteta alusta hitaalla aloituksella

- vaan pudotetaan ruuhkaikkuna puoleen
- ja jatketaan normaalilla lineaarisella kasvattamisella



- hidas aloitus ja ruuhkan valvonta ongelmallisia langattomassa yhteydessä
  - parannuksia
    - tarkempi kello
    - ruuhkan ennustaminen ennen ajastimen laukeamista
    - early warning system
- => 40-70 % parempia tuloksia

## TCP langattomassa verkossa

- monet TCP-toteutukset optimoitu luotettaville lankaverkoille => suorituskyky langattomissa verkoissa erittäin huono
  - ruuhkanvalvonta-algoritmi olettaa ajastimen laukeamisen johtuvan ruuhkasta
    - lähettämistä hidastetaan, jotta verkon kuormitus pieneneisi ja ruuhkaa ei syntyisi
  - langattomat yhteydet ovat epäluotettavia ja paketteja katoaa
    - kadonneet paketit syytä lähettää nopeasti uudelleen
    - lähetystä pitäisi päinvastoin nopeuttaa!

## 6.4.1. TCP:n palvelumalli

### ■ pistoke (socket)

- TCP-yhteyden päätepiste sovellukselle
  - lähettäjällä ja vastaanottajalla oma pistoke
- pistokenumero 48 bittiä
  - koneen 32 bitin IP-osoite
  - 16 bitin porttinumero
    - kiinnitettyjä portteja (well-known port)
      - » FTP 21
      - » TELNET 23
      - » HTTP 80

# TCP-yhteys

- kaksisuuntainen (full-duplex) kaksipisteyhteys
- tunnustetaan päätepisteinä olevien pistokkeiden tunnuksista (pistoke1, pistoke2)

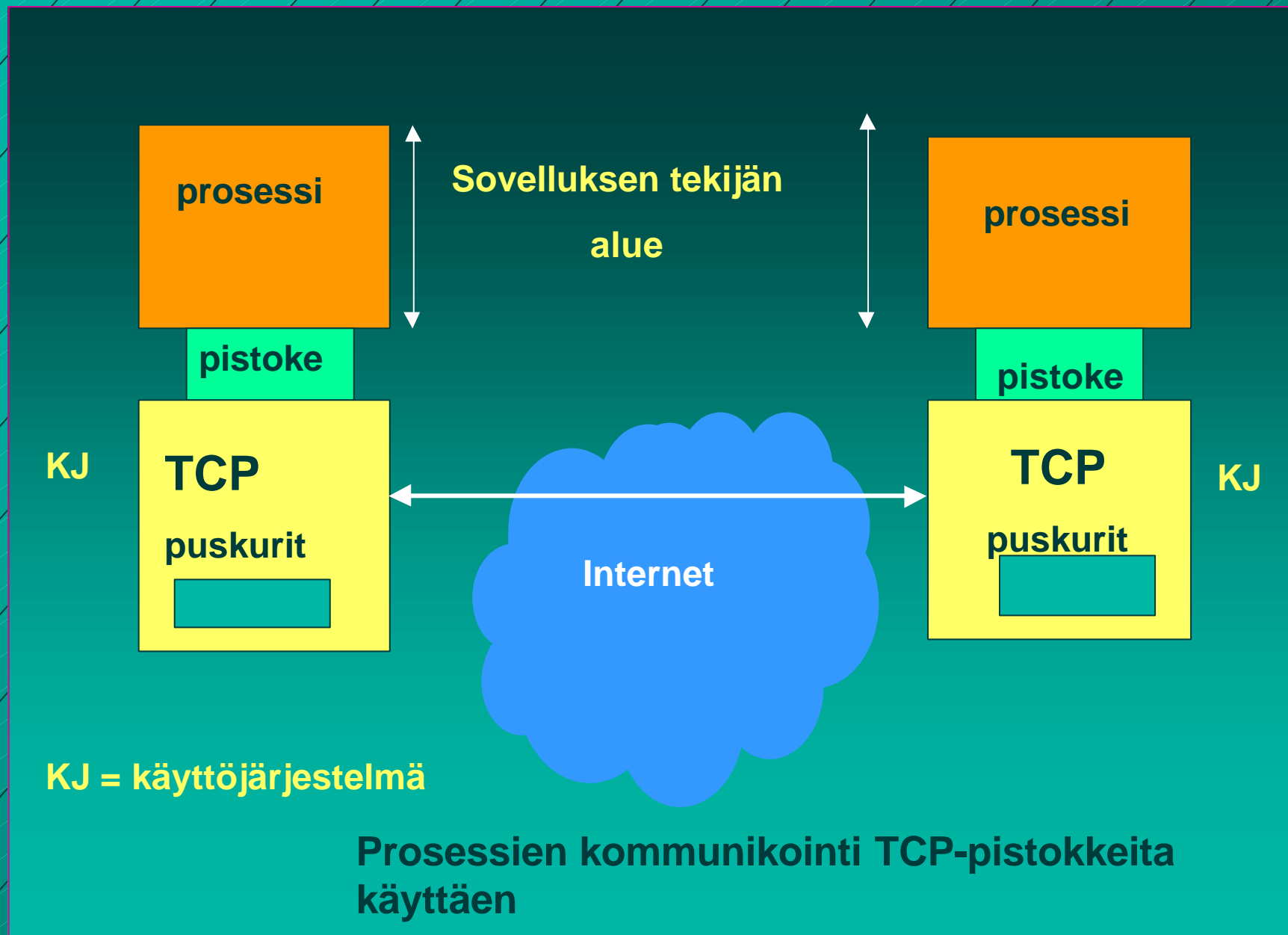


# Pistokerajapinta (Socket interface)

- Verkkopalvelun ja sitä käyttävän sovelluksen rajapinta
  - yleensä käyttöjärjestelmän tarjoama palvelu
  - pistokerajapinta alunperin Berkeley Unixin mukana, nyt lähes kaikissa käyttöjärjestelmissä
  - miten verkkoprotokollan tarjoamiin palveluihin päästään käsiksi sovelluksesta

# TCP:n pistokeprimitiivit

- SOCKET luo uusi yhteyden päätepiste pistoke
- BIND anna pistokkeelle osoite
- LISTEN halukas vastaanottamaan yhteyksiä
- ACCEPT jää odottamaan yhteysyrityksiä
- CONNECT yritä muodostaa yhteys
- SEND lähetä dataa yhteyttä pitkin
- RECEIVE vastaanota dataa yhteydeltä
- CLOSE pura yhteys (symmetrinen)



# Kuljetusyhteyden muodostus ja käyttö

SOCKET

BIND

LISTEN



Kuljetus-  
palvelun  
tuottaja  
esim. TCP



ACCEPT

conn.req.

CONNECT(SERVER)

conn. ack.

SEND(DATA)

RECEIVE

data

RECEIVE

SEND(DATA)

data



# TCP-yhteyden hallinta

- yhteys muodostetaan **kolminkertaisella kättelyllä**
- passiivinen osapuoli kuuntelee
  - SOCKET
  - LISTEN
  - ACCEPT
- aktiivinen osapuoli aloittaa yhteydenmuodostuksen
  - CONNECT

## ■ CONNECT-primitiivi

– parametreina

- IP-osoite ja porttinumero
- suurin hyväksyttävä segmentin koko
- muuta tietoa, esim. salasana



## ■ TCP-segmentti, jossa SYN-segmentti

- SYN = 1
- ACK = 0

client  
proc

server  
proc

SOCKET  
CONNECT

SOCKET  
LISTEN  
ACCEPT

Asiak-  
kaan  
pistoke

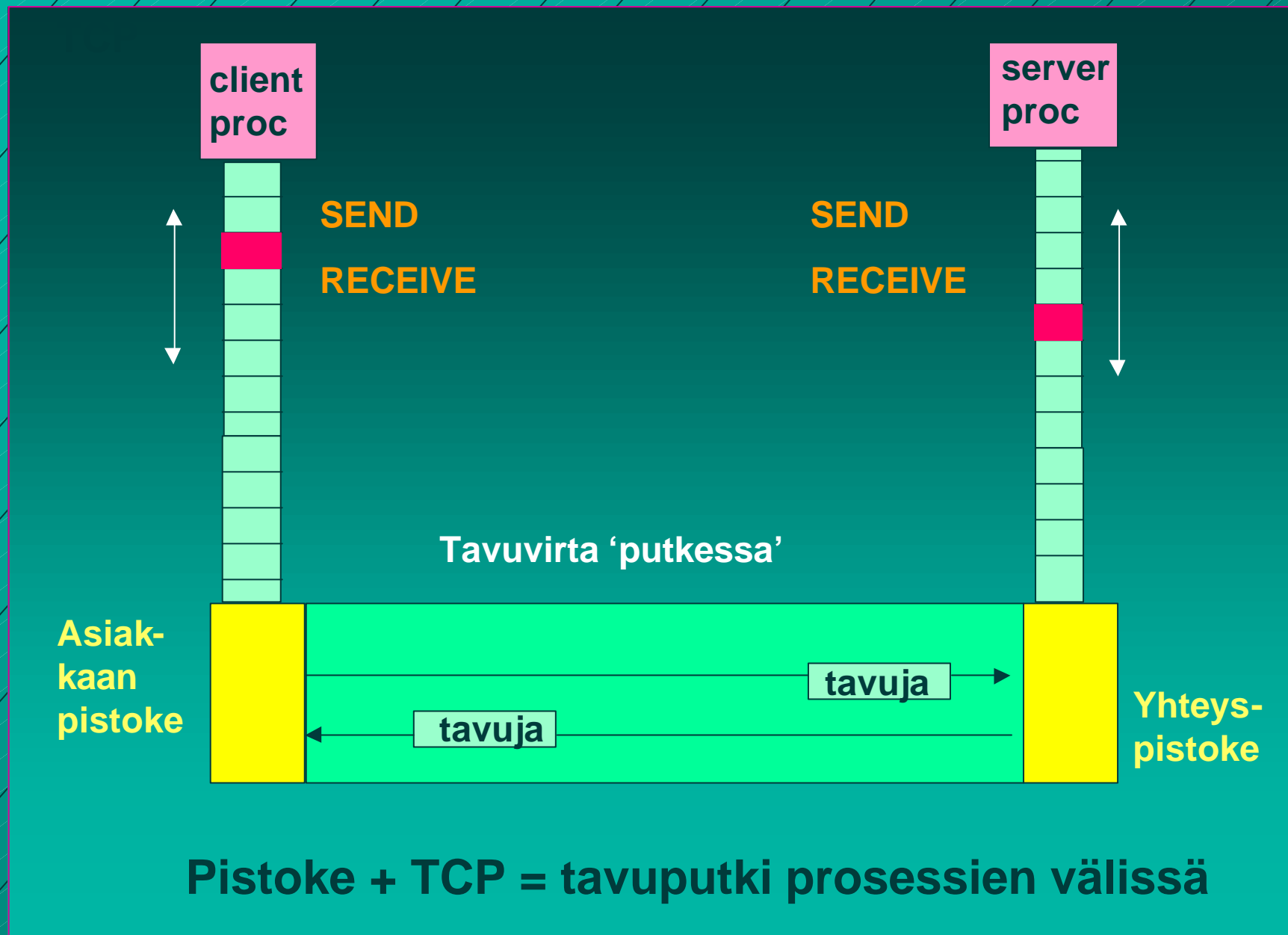
Kolminkertainen kättely =  
TCP-yhteyden muodostus

Yhtey-  
denotto  
pistoke

Datan siirto

Yhtey-  
spistoke

**Pistoke + TCP = tavuputki prosessien välissä**



- TCP-yhteys on tavuvirtaa, ei sanomavirtaa
  - lähetettäessä neljä 512 tavun pätkää vastaanottaja saa joko
    - neljä 512 tavun pätkää
    - kaksi 1024 tavun pätkää
    - yhden 2048 tavun pätkän

Segmentit lähetetään neljänä eri IP-pakettina

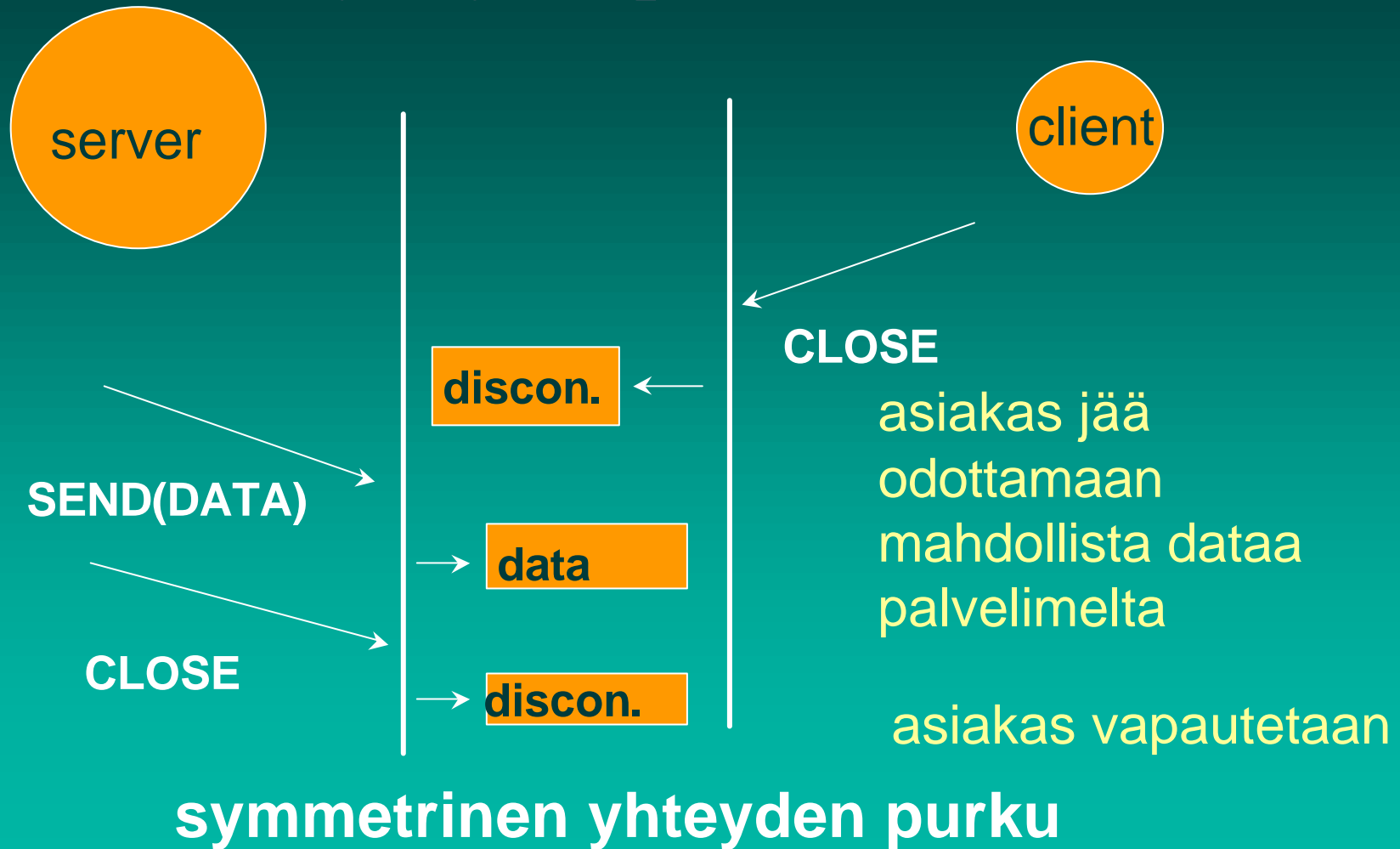
Ne luovutetaan vastaanottajalle yhdellä READ-kutsulla



neljä 512 tavun segmenttiä

yksi 2048 tavun data

# yhteyden purkaminen



# C-rutiineina

```
int socket(int domain, int type, int protocol)
```

## palvelin:

```
int bind (int socket, struct sockaddr *address,  
         int addr_len)
```

```
int listen(int socket, int backlog)
```

```
int accept(int socket, struct sockaddr *address,  
          int *addr_len)
```

## asiakas:

```
int connect (int socket, struct sockaddr *address,  
            int addr_len)
```

```
int send(int socket, char *message, int msg_len, int flags)
```

sanoman lähetys annetun pistokkeen kautta

```
int recv(int socket, char *buffer, int buf_len, int flags)
```

sanoma vastaanotto annetusta pistokkeesta ilmoitettuun puskuriin



# Pistokeohjelmointia Javalla

- Socket clientSocket = new Socket("hostname", 6789);
- clientSocket.close();
- ServerSocket welcomeSocket = new ServerSocket(6789);
- Socket connectionSocket = welcomeSocket.accept();
- (esimerkki kirjassa Kurose, Ross, Computer Networking, A Top-Down Approach Featuring the Internet)

# Pistokeohjelmointi

- Pistokeohjelmointia ja yleensä hajautettujen verkkosovellusten tekemistä opetellaan erillisellä kurssilla
  - **Verkkosovellusten toteuttaminen**  
(järjestetään keväällä 2001)