

# Tietoliikenteen perusteet

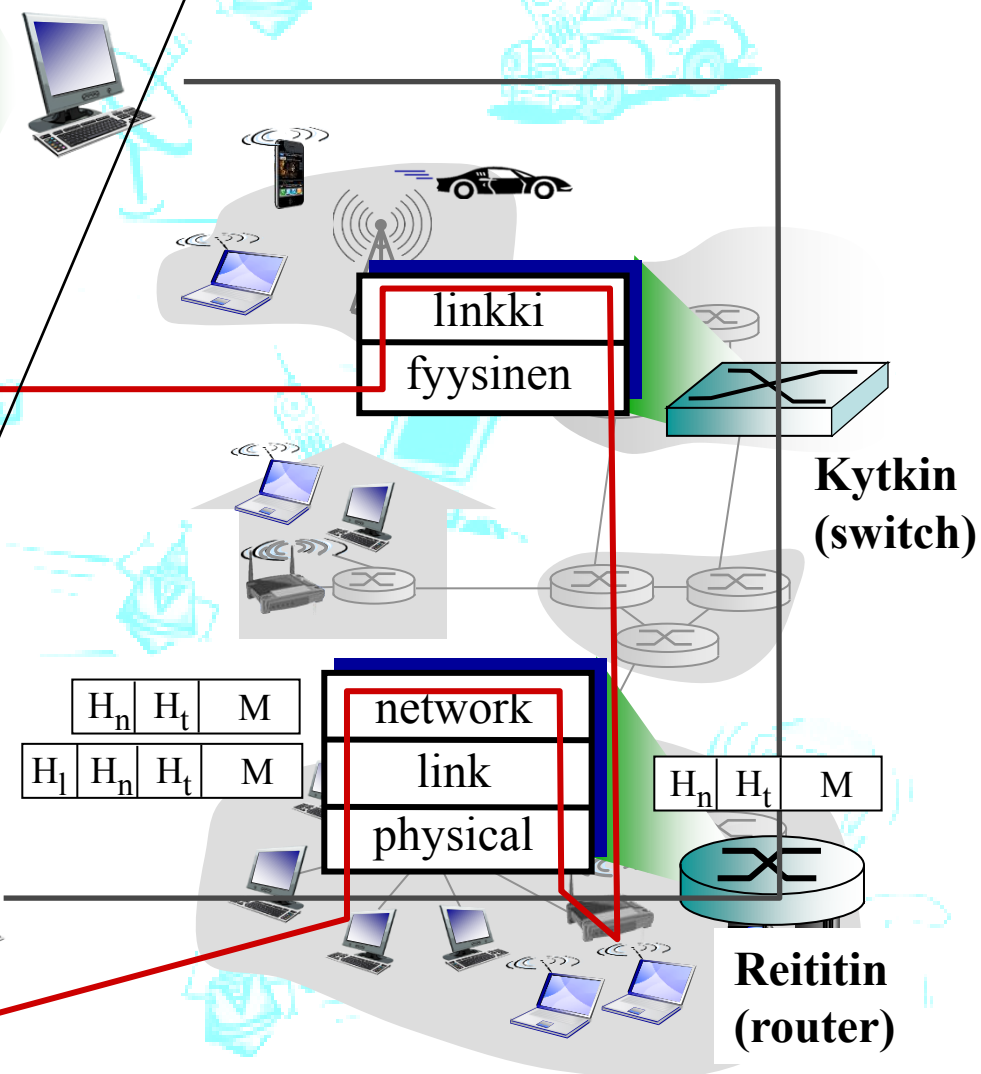
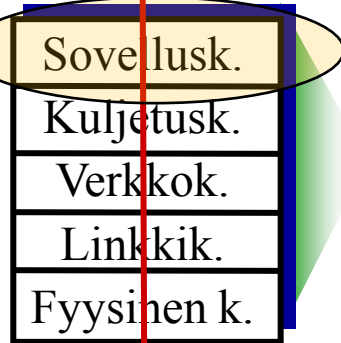
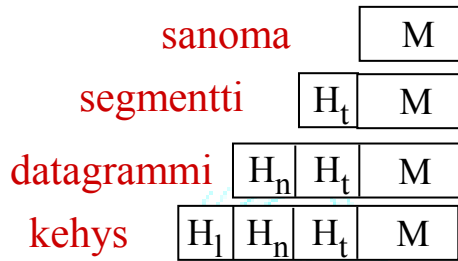


Luento 3: Sovelluskerros  
verkkosovelluksen periaatteet, WWW, pistoke

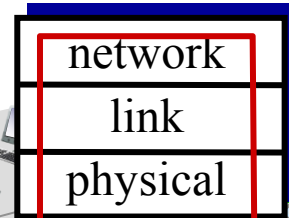
Syksy 2017, Timo Karvi

# Lähettäjä (source)

# Luennon sisältöä

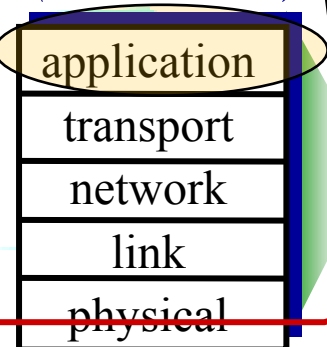
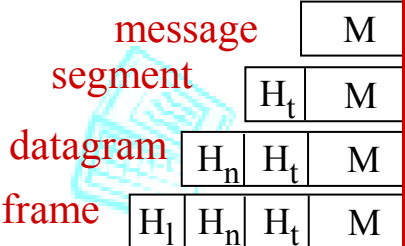


**Kytkin (switch)**



**Reititin (router)**

# Vastaanottaja (destination)



# Sisältöä

- Verkkosovellusten periaatteet
- World Wide Web ja HTTP
- Tiedostonsiirto ja FTP
- Sähköposti ja SMTP, IMAP, POP3
- Nimipalvelu ja DNS
- Vertaistoimijat (peer-to-peer)
- Pistoke ja sen käyttö

## Oppimistavoitteet:

- Osaa selittää asiakaspalvelija-malliin perustuvien verkkosovellusten toimintaperiaatteet
- Tuntee sovellusprotokollien syntaksia ja semantiikkaa
- Osaa selittää www:n ja sähköpostin toimintaideat



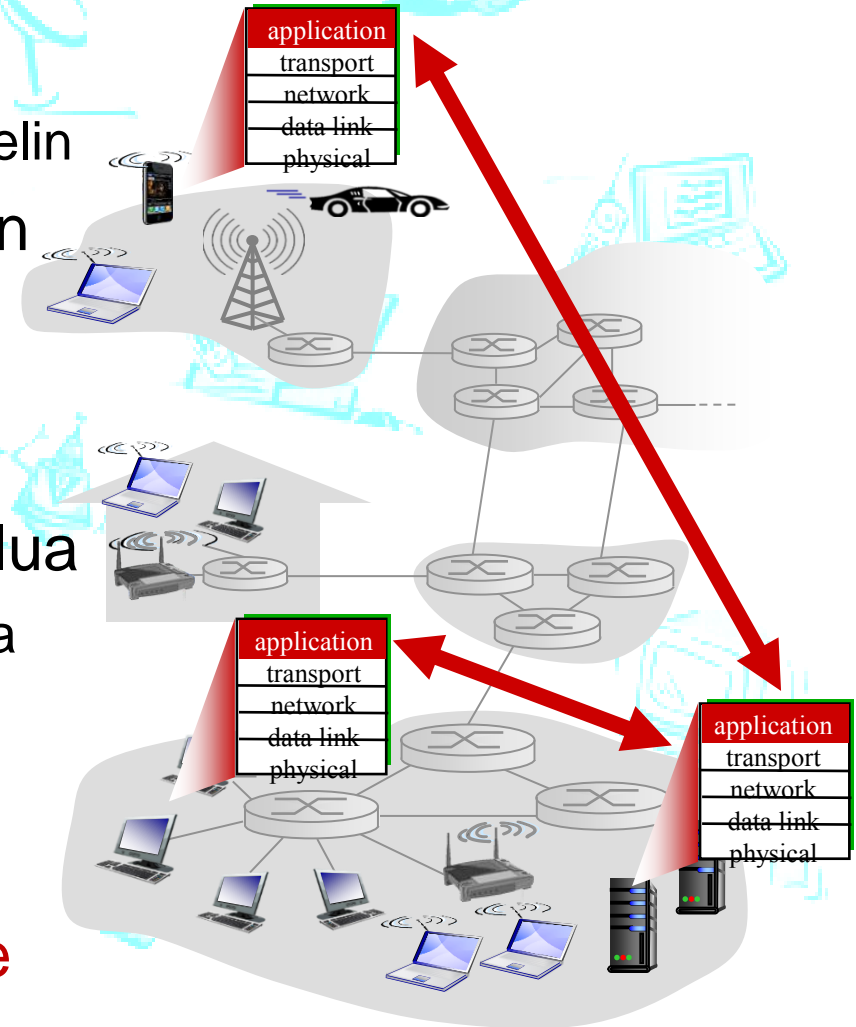


# VERKKOSOVELLUSTEN PERIAATTEET

# Verkkosovellus

- Sovelluksen ohjelmat eri isäntäkoneissa
  - esim. www-selain ja www-palvelin
- Sovellusprotokolla kuvaa näiden sanomanvälityksen
  - Syntaksi, semantiikka, järjestys
- Sanomat välitetään käyttäen verkon tarjoamaa kuljetuspalvelua
  - osa järjestelmän perusrakennetta
  - sovelluksista riippumatonta
- Reititys tapahtuu vasta verkkotasolla, mutta **sovellustasolla tiedettävä osoite**

Fig 2.1 [KR12]

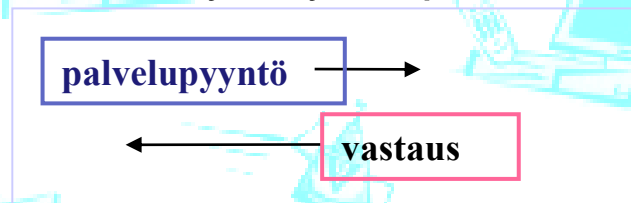


Google, e-Bay,  
Facebook,  
YouTube,  
Amazon, ..

# Sovellusarkkitehtuuri

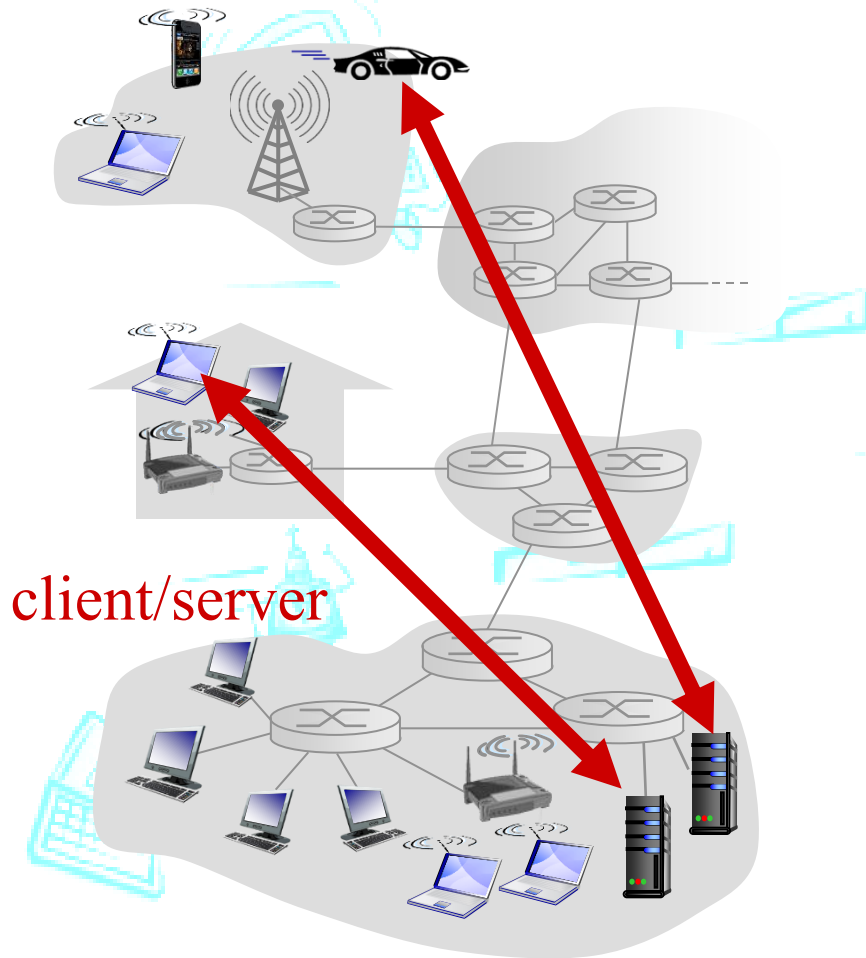


- **Asiakas-palvelija-malli** (esim. selain ja www-palvelin)
  - Aina toiminnassa oleva palvelinohjelma, jolla kiinteä, tunnettu IP-osoite
  - Asiakasohjelmat ottavat yhteyttä palvelimeen ja pyytävät siltä palvelua



- **Vertaistoimijamalli** (esim. BitTorrent, eMule, Skype)
  - Vertaisisännät kommunikoivat suoraan keskenään
  - Ei tarvitse olla aina toiminnassa, IP-osoite voi muuttua
  - Jokainen toimii sekä palvelijana että asiakkaana
- **Hybridimalli** (esim. Napster, pikaviestimet )

# Asiakas-palvelin arkkitehtuuri



## Palvelin (server):

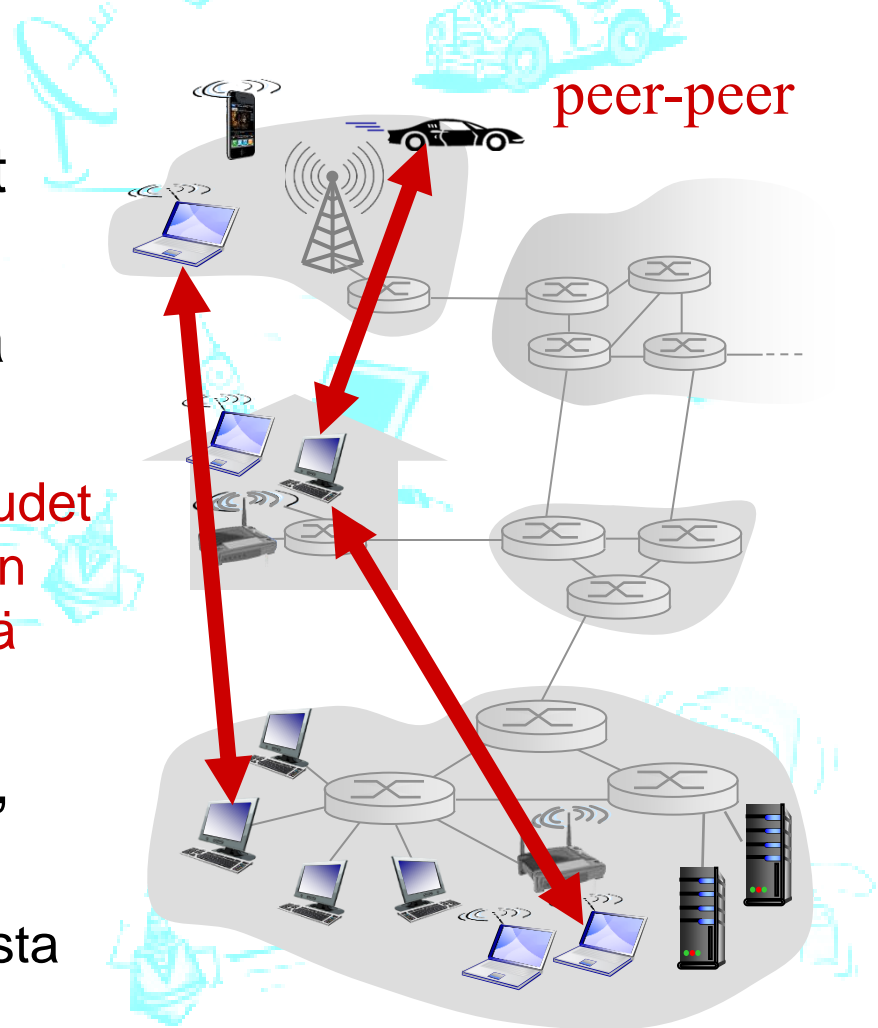
- Aina saatavilla (24/7)
- Pysyvä IP-osoite
- Palvelinkeskukset (data centers)

## Asiakkaat (clients):

- Ottavat yhteyttä palvelimeen
- Verkkoyhteys voi vaihdella
- Usein dynaaminen IP-osoite
- Eivät kommunikoi suoraan toistensa kanssa

# Vertaisverkko (P2P) arkkitehtuuri

- *Ei keskitettyä palvelinta*
- Vertaisisännät kommunikoivat suoraan keskenään:
- pyytävät palveluja toisiltaan ja antavat palveluja toisilleen
  - *Skaalautuu automaattisesti*– uudet vertaisisännät tuovat mukanaan sekä uutta palvelukysyntää että uutta palvelukapasiteettia
- Vertaisisäntiä liittyy ja poistuu, IP-osoitekin voi vaihtua
  - Verkon hallinta on monimutkaista





# Prosessien kommunikoinnista

*prosessi*: sovellus, jota suoritetaan isäntäkoneella

- Prosessit samalla koneella: prosessien välinen kommunikointi (**inter-process communication**) joko yhteistä muistia tai viestien vaihtoa käyttäen (KJ määrittelee vaihtoehdot)
- Prosessit eri koneilla: Kommunikointi aina viestien (**messages**) vaihdolla

asiakkaat, palvelimet

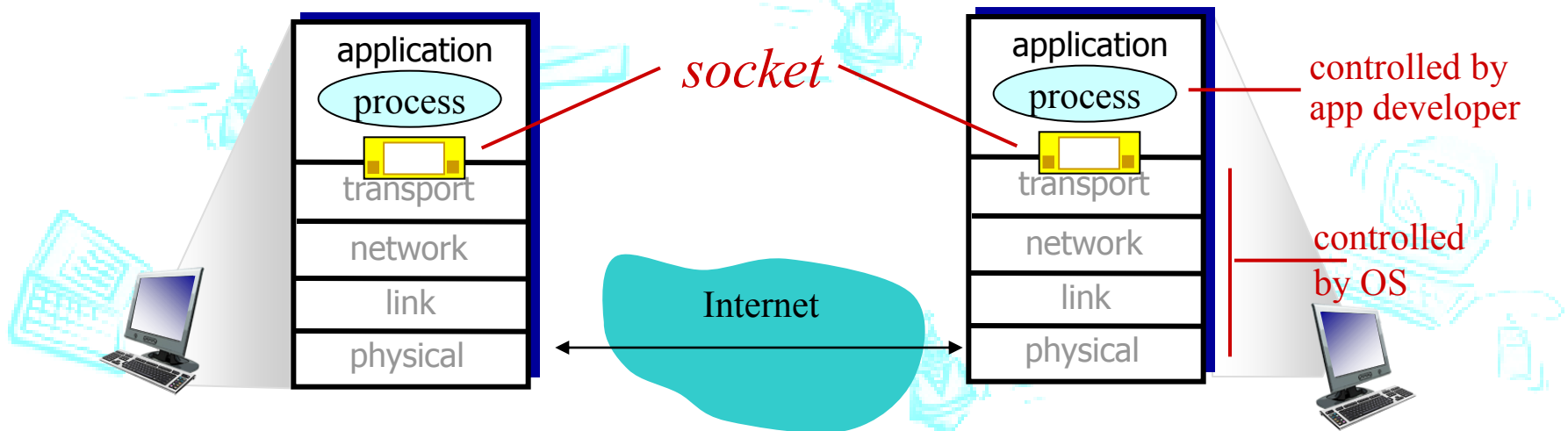
*Asiakasprosessi*: ottaa yhteyttä, aloittaa

*Palvelinprosessi*: odottaa yhteydenottoa

- ❖ HUOM: vertaisisännillä on sekä asiakas- että palvelinprosesseja

# Sovelluksen rajapinta tietoliikenteeseen: Pistokkeet (sockets)

- Prosessi lähettää ja vastaanottaa sanomia (messages) **pistokkeen (socket)** kautta
- Pistoketta voi ajatella ovena
  - Lähettävä prosessi tyrkkää sanoman ulos ovesta
  - Lähettävä prosessi luottaa että oven takana oleva kuljetuspalvelu toimittaa sanoman vastaanottajan ovelle

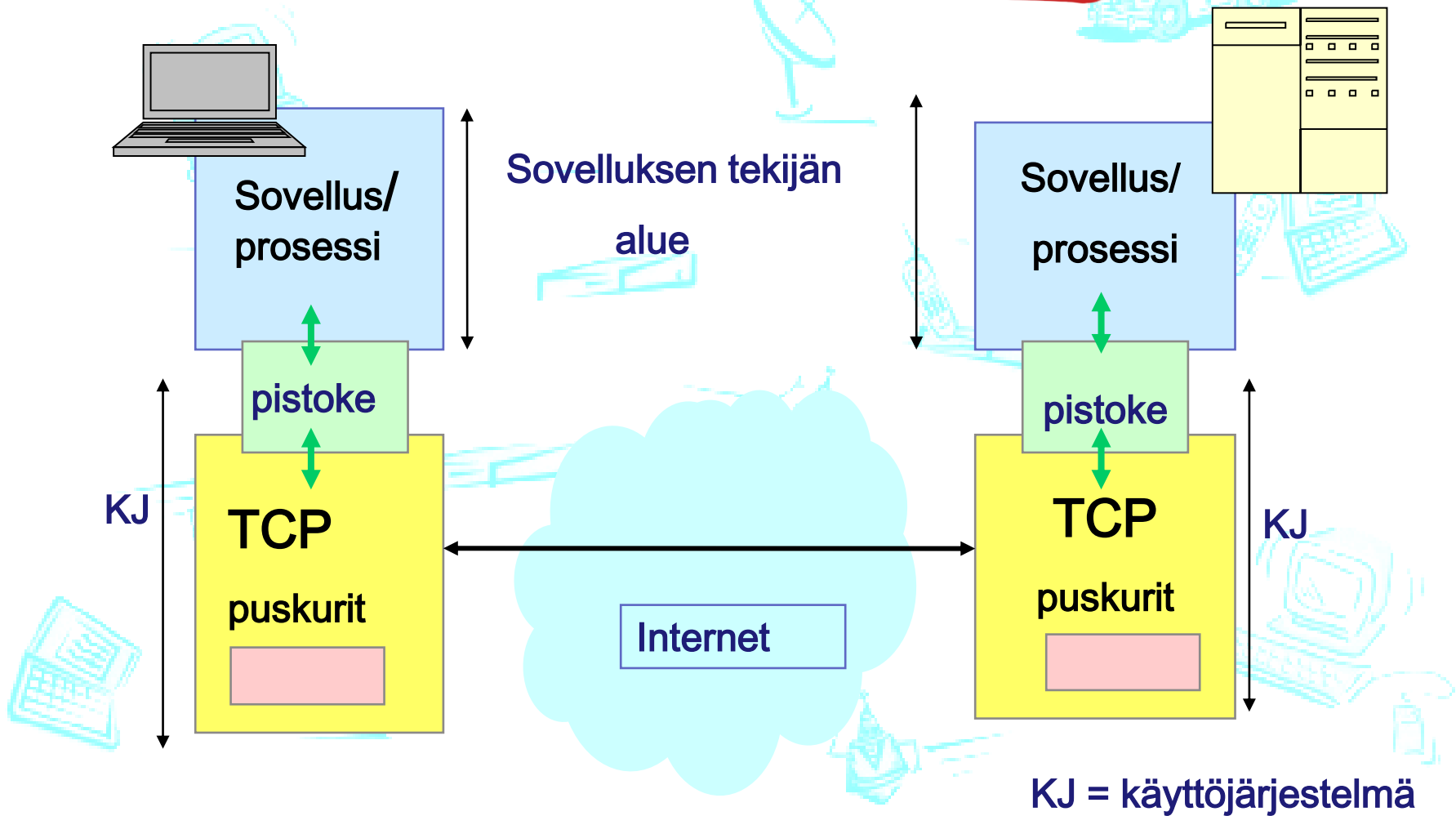


# Sovelluksen rajapinta tietoliikenteeseen: Pistoke (socket)

---

- **Pistoke (socket)** (verkkosovelluksen ohjelmointirajapinta, API)
  - yhteyden muodostaminen; lue /kirjoita sanoma
- Prosessi kirjoittaa verkkoon ja lukee verkosta lähes samalla tavoin kuin kirjoittaisi tiedostoon ja lukisi tiedostosta
  - 'luukku' tai 'ovi', josta dataa sisään /ulos
- **Lähetys (send)**: anna sanoma KJ:lle
- **Vastaanotto (receive)**: ota sanoma KJ:ltä.
  - Sovellus odottaa, jos sanoma ei ole vielä saapunut
- Kaksisuuntainen (full duplex)
  - Samaan pistokkeeseen sekä kirjoitetaan että siitä luetaan
- Ohjelmoija valitsee, käyttääkö KJ kuljetuskerroksella yhteydellistä vai yhteydetöntä palvelua!

# Prosessien kommunikointi TCP-pistokkeita käyttäen



# KJ:n rajapinta laitteistoon

KJ:n kannalta tietoliikenne normaalia siirrantää (I/O:ta)

## Lähtevä liikenne:

- **Prosessi pyytää** kuljetuspalvelua KJ:n palvelupyynnöllä send
- Kuljetuskerros hoitaa omat tehtävänsä ja kutsuu verkkokerroksen rutiinia
- Verkkokerros tekee hommansa ja kutsuu laiteajurin rutiinia
- **Laiteajuri** vie datan ja komennot verkkokortin ohjaimen rekistereihin
- **Verkkokortti** siirtää bitit linkille (linkkikerros ja fyysinen siirto)

## Tuleva liikenne:

- **Verkkokortti** ottaa vastaa linkiltä tulevat bitit (fyysinen siirto ja linkkikerros)
- ...ja aiheuttaa keskeytyksen.
- KJ:n **laiteajuri** siirtää bitit verkkokortilta keskusmuistiin
- Ajuri kutsuu verkkokerroksen rutiinia, joka suorittaa omat toimintonsa
- Verkkokerros kutsuu kuljetuskerroksen rutiinia, joka tekee omat toimionsa
- Sanoma **prosessille** vasta, kun se sitä pyytää palvelupyynnöllä receive

# Osoittaminen

- Sanomissa oltava lähettäjän ja vastaanottajan IP-osoite ja porttinumero

[www.iana.org](http://www.iana.org)

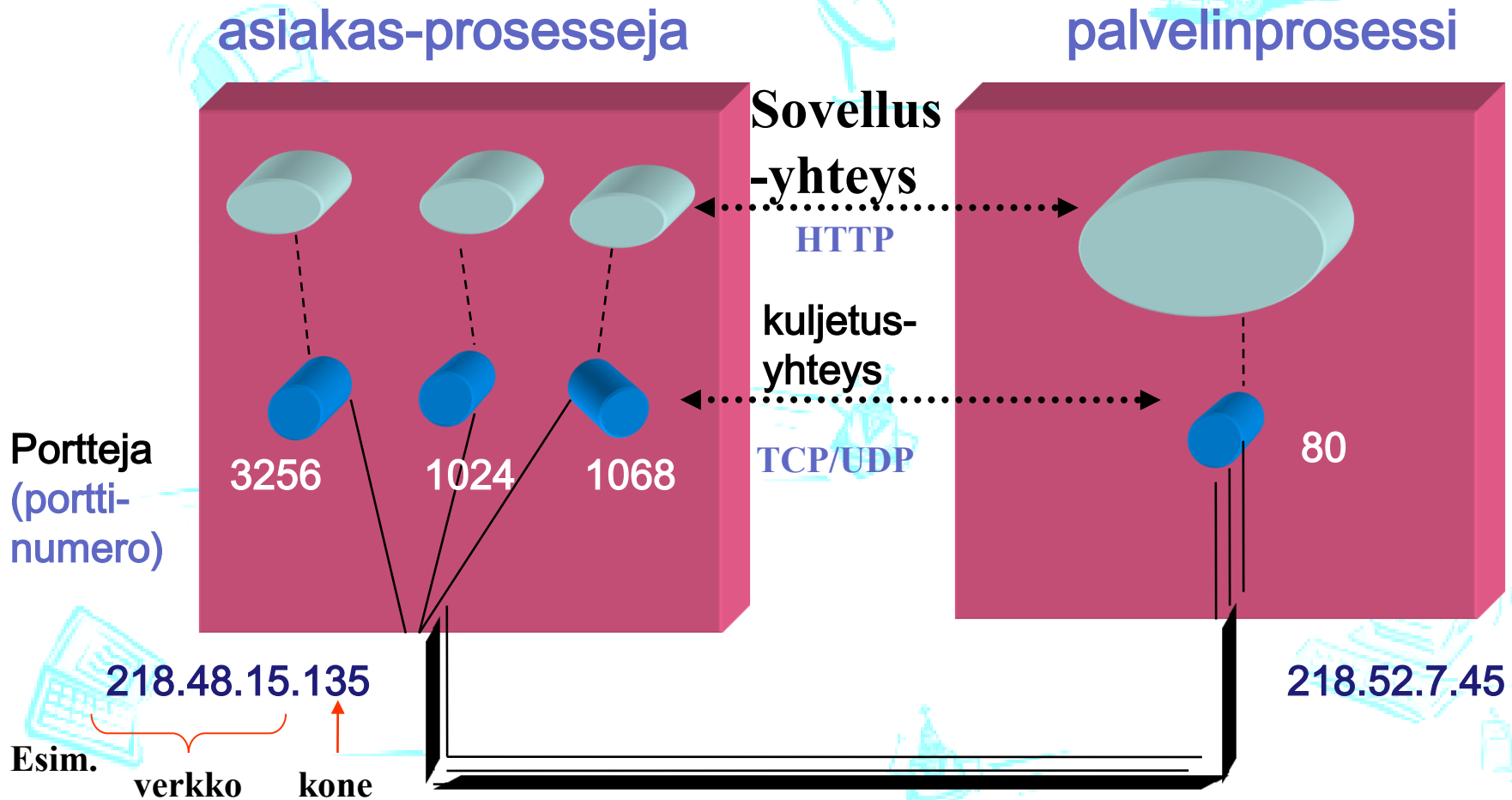
- **IP-osoite** → oikea kone

- koneen (verkkokortin) yksilöivä 32-bittinen tunniste
- osoitteen verkko-osa yksilöi verkon
- osoitteen koneosa yksilöi koneen verkossa

- **Porttinumero** → oikea prosessi

- Yleisillä palveluilla standardoidut tunnetut porttinumerot:
  - www-palvelin kuuntelee porttia 80,
  - Postipalvelin kuuntelee porttia 25
- KJ osaa liittää porttinumeron prosessiin

# looginen 'päästä-päähän' yhteys



# Kuljetuskerroksen palveluja sovelluksille

- Tiedon eheys (data integrity)

- Jotkut sovellukset tarvitsevat täysin luotettavan tiedon siirron (kuten www, sähköposti, tiedostojensiirto)
- Jotkut taas sietävät häviöitä tiedonsiirrossa (audio)

- Ajoitukset (timing)

- Jotkut sovellukset vaativat lyhyitä siirtoviipeitä (IP-puhelut, verkkopelit)

- Suoritusteho (throughput)

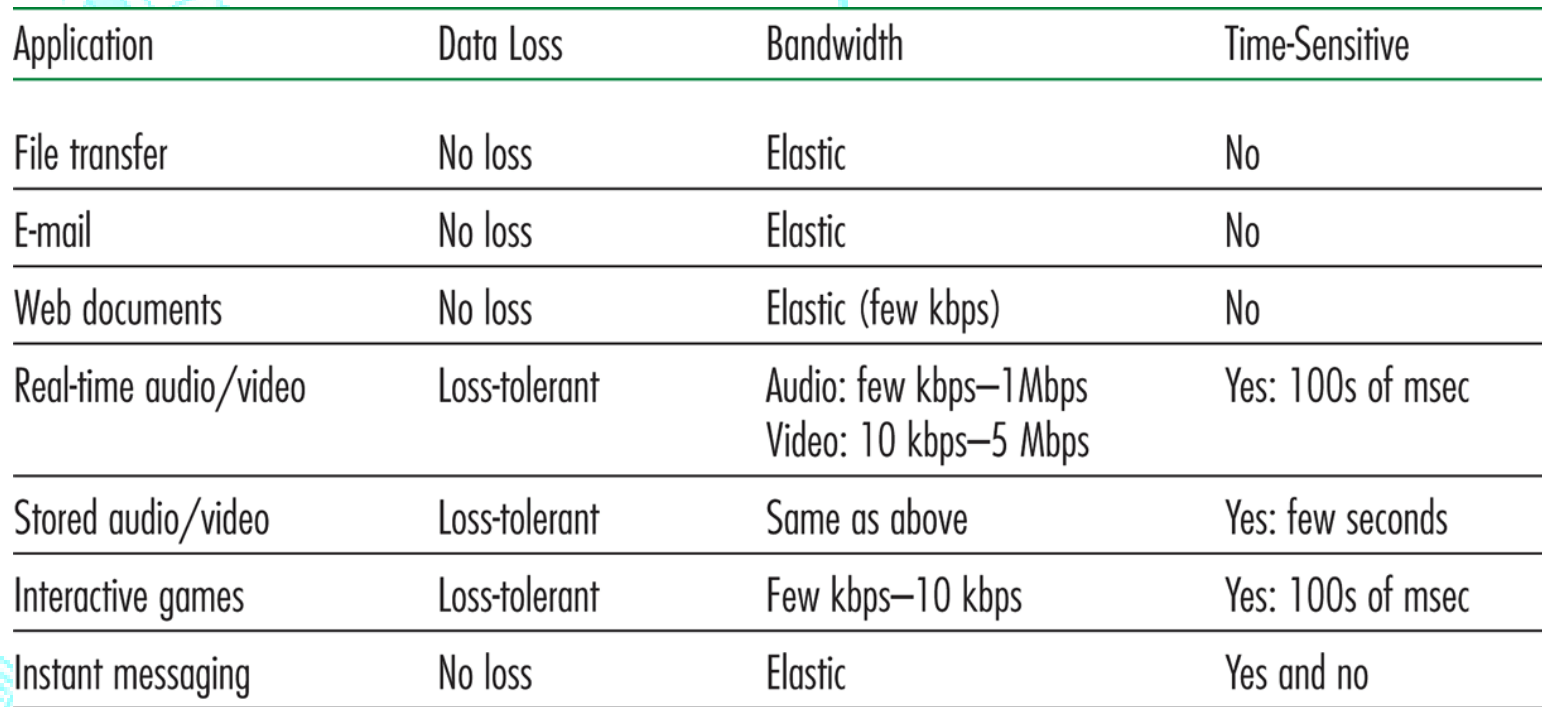
- Jotkut sovellukset tarvitsevat tietyn minimikapasiteetin toimiakseen (multimedia)
- Jotkut sopeutuvat siihen kapasiteettiin, jonka saavat

- Turvallisuus (security)

- Tiedon suojaus, salaus, eheys ...



# Kuljetuspalvelun laatuvaatimuksia



Application	Data Loss	Bandwidth	Time-Sensitive
File transfer	No loss	Elastic	No
E-mail	No loss	Elastic	No
Web documents	No loss	Elastic (few kbps)	No
Real-time audio/video	Loss-tolerant	Audio: few kbps–1 Mbps Video: 10 kbps–5 Mbps	Yes: 100s of msec
Stored audio/video	Loss-tolerant	Same as above	Yes: few seconds
Interactive games	Loss-tolerant	Few kbps–10 kbps	Yes: 100s of msec
Instant messaging	No loss	Elastic	Yes and no

**Figure 2.4** ♦ Requirements of selected network applications

# Kuljetusprotokollat: TCP

## (Transmission Control Protocol) [RFC 793]


- **Yhteydellinen palvelu** (connection-oriented)
  - Yhteyden muodostus ennen datan siirtoa (handshaking)
  - Kaksisuuntainen TCP-yhteys (full-duplex)
  - Yhteyden purku (shutdown)
- **Luotettava kuljetuspalvelu**
  - Järjestyksen säilyttävä tavuvirta sovellukselle
  - segmenttinumerot, kuittaukset, uudelleenlähetykset
- **Vuonvalvonta** (flow control)
  - Lähettäjä hiljentää vauhtia, jos **vastaanottaja** ei ehdi käsitellä
- **Ruuhkanvalvonta** (congestion control)
  - Lähettäjä hiljentää vauhtia, jos **reitittimet** eivät ehdi käsitellä

# Kuljetusprotokollat: UDP

(User Datagram Protocol) [RFC768]

- Kevyt kuljetuspalvelu, pieni yleisrasite
- Ei yhteyden muodostusta eikä purkua
- Ei takuita sanoman perillemenosta
  - Sanoman segmentit vain lähetetään verkkoon
  - Sanoman segmenttejä voi puuttua ja ne voivat saapua epäjärjestyksessä, virheelliset yleensä hylätään
- Ei vuonvalvontaa, ei ruuhkanvalvontaa
  - UDP voi lähettää niin paljon kuin haluaa
- Huom! TCP ja UDP ei takuita siirtonopeudelle eikä viipeelle => ei mitään aikatakuita (ns. 'best effort'-palvelu)
- Ei myöskään datan salakirjoitusta => SSL (Secure Socket Layer)

# Kumpi?



Applications	Application-Layer Protocol	Underlying Transport Protocol
Electronic mail	SMTP [RFC 2821]	TCP
Remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
File transfer	FTP [RFC 959]	TCP
Remote file server	NFS [McKusik 1996]]	UDP or TCP
Streaming multimedia	Often proprietary (e.g., Real Networks)	UDP or TCP
Internet telephony	Often proprietary (e.g., Net2phone)	Typically UDP



**Figure 2.5** ♦ Popular Internet applications, their application-layer protocols, and their underlying transport protocols

# TCP:n turvallisuus ja suojaaminen

## TCP & UDP

- Ei salausta
- Pistokkeelle annettu selväkielinen salasana kulkee Internetissä salaamattomana

## SSL

- Tarjoaa salatun TCP-yhteyden
- Datan eheys
- Osapuolten todennus

## SSL on sovelluskerroksella

- Sovellus käyttää SSL-kirjastoa, joka käyttää TCP:tä

## SSL-pistokerajapinta (API)

- ❖ Pistokkeelle annettu selväkielinen salasana (*cleartext passwd*) kulkee Internetissä salattuna
- ❖ Luku 7 (Luento 11 tai 12)



# WORLD WIDE WEB, WWW HTTP

# WWW ja HTML

## (Hyper Text Markup Language)

- WWW-sivu, WWW-dokumentti
  - HTML-tekstiä, jossa viittauksia muihin objekteihin
    - muu HTML-tiedosto, kuva- tai äänitiedosto, Java applet, ...
  - Sivun muodostuu usean tiedoston sisällöstä, jotka noudetaan palvelijalta
- Viittaus URL-osoitteella (Uniform Resource Locator)
- <http://www.someschool.edu/someDept/pic.gif>

koneen nimi

Viitatus objektin  
polkunimi

# HTML (HyperText Markup Language)

- Standardi siitä, kuinka sivun rakenne kuvataan
  - Muotoilut, eri osien sijoittelu sivuille
  - Viittaukset muihin objekteihin
- SGML (Standard Generalized Markup Language)
  - yleinen merkkauskieli
  - kertoo, kuinka dokumentit muotoillaan ~ladontamerkinnyt
- XML (Extensible Markup Language )
  - rakenteellinen tietosisällön kuvaus, myös merkitys kuvattu
- Näistä enemmän kurssilla:

**582304 XML-metakieli**



# HTTP (HyperText Transfer Protocol)

[RFC 1945, RFC 2616]

- WWW:n sovellusprotokolla
  - Tekstimuotoiset sanomat
  - pyyntö – vastaus
- Asiakas = joku selain
  - pyytää, noutaa ja näyttää objektit
- Palvelija = joku web-palvelu
  - etsii objektin (tiedoston) koneen hakemistosta
  - ja lähettää sen vastauksena asiakkaalle

Fig 2.6 [KR12]



# HTTP protokollan perustoiminta

## *Käyttää TCP:tä*

- Asiakas avaa TCP - yhteyden (luo pistokkeen) palvelimelle, porttiin 80
- Palvelija hyväksyy TCP-yhteyden asiakkaaseen
- HTTP-sanomia (sovellustason protokollan mukaisesti) lähetetään selaimen (HTTP-asiakas) ja www-palvelimen (HTTP-palvelija) välillä
- TCP-yhteys suljetaan

## *HTTP on “tilaton” (stateless) protokolla*

- Palvelija ei muista mitään edellisistä pyynnöistä

***HUOM!***

**Tilansäilyttävät protokollat ovat monimutkaisia!**

- ❖ Yhteyden historia (tila) täytyy pitää tallessa
- ❖ Jos palvelija tai asiakas vikaantuu, niiden käsitys tilasta voi olla erilainen ja vaatia yhdenmukaistamista

# HTTP ja yhteyden säilyvyys

---

## *Ei säilytä (non-persistent)*

- Asiakas avaa TCP-yhteyden jokaiselle HTTP-pyynnölle erikseen
  - Palvelin sulkee yhteyden aina vastausviestin jälkeen
- Jokaiselle elementille oma yhteys!

## *Säilyttävä (persistent)*

- Asiakas avaa yhden TCP-yhteyden, jonka yli siirretään kaikki tarvittavat elementit
  - Palvelin ei sulje yhteyttä vastausviestin jälkeen

# Vastausaika (response time) yhdele TCP-yhteydelle

Fig 2.7 [KR12]

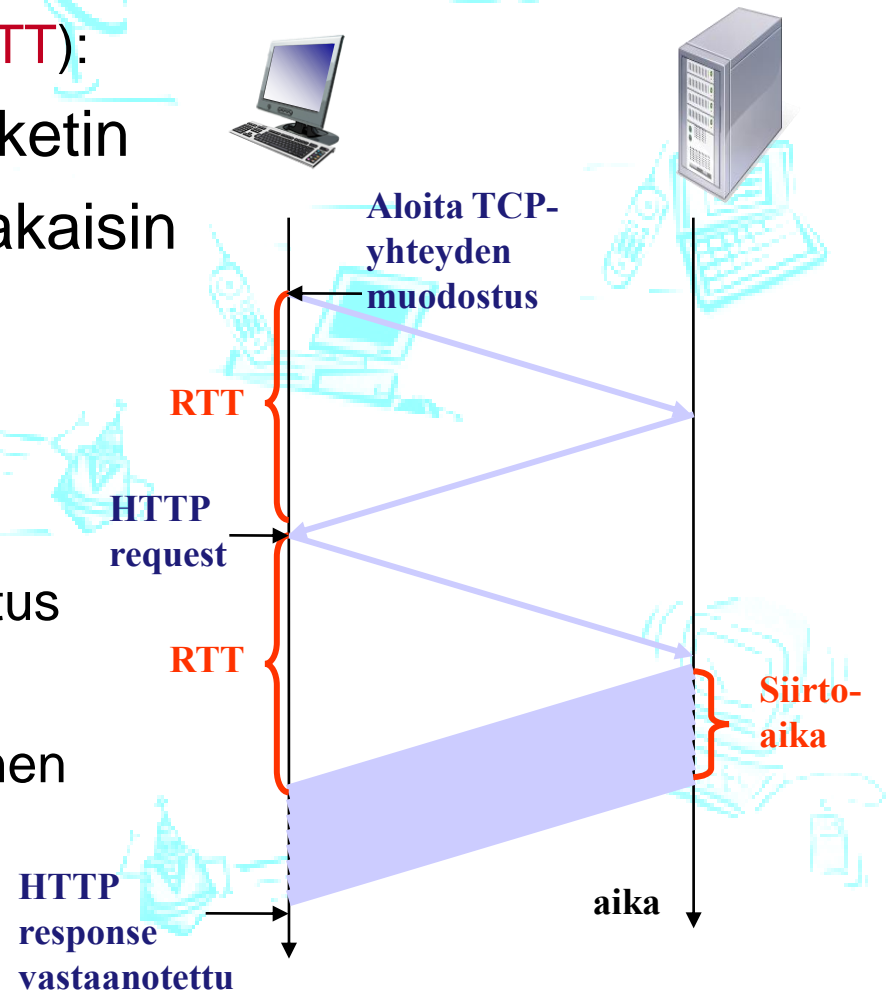
- **Kiertoviive** (Round-trip time, **RTT**):  
aika, joka kuluu pikkupaketin  
siirtoon palvelimelle ja takaisin

- **Vastausaika = 2 RTT  
+ siirtoaika**

1 RTT TCP-yhteyden muodostus

1 RTT pyyntö + ensimmäisten  
vastausbittien saapuminen

Tiedoston siirtoaika



# Suorituskyky?

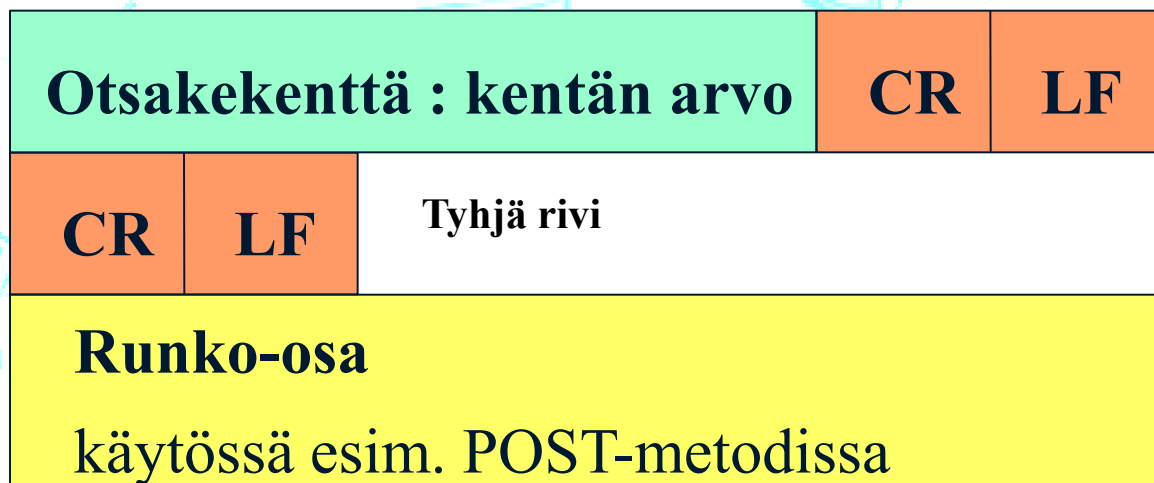
- Jos sivulla viitataan 10 objektiin (*ja yhteys ei säily*)
  - 11 peräkkäistä TCP-yhteyden muodostusta ja purkua?
    - KJ varaa ja vapauttaa puskuritilaa; muodostukseen kuuluu kaikkiaan 22 RTT
    - Avataan useita rinnakkaisia yhteyksiä? Puskuritilat yhteyksille
- Käytetään säilyvää TCP-yhteyttä (persistent)
  - Oletus uusimmissa standardeissa: Palvelin jättää yhteyden (toistaiseksi) sulkematta. Ajastin on säädettävissä.
  - Seuraavat samalle palvelimelle kuuluvat pyynnöt ja vastaukset käyttävät samaa yhteyttä
- Liukuhihnoitettu (pipelining) / liukuhihnoittamaton:
  - Seuraava pyyntö lähtee jo ennenkuin edelliseen on saatu vastaus / ei lähde.

# HTTP-pyyntö: yleinen rakenne

GET /jokuhakemisto/sivu.html HTTP/1.1



... Lisää otsakerivejä



SP='space' eli välilyönti

CR + LF = rivin päättäminen

13 ja 10 (desim.) historia tulostimen ohjauksessa

# Esimerkki: HTTP-pyyntö

Pyyntörivi: GET/ POST/ HEAD –metodi

GET /somedir/page.html HTTP/1.1

Host: www.someschool.edu

User-Agent: Mozilla/4.0

Connection: close

Accept-language: fr

otsakerivit

Yksi tyhjä rivi merkkinä  
sanoman loppumisesta

(Ylimääräinen carriage return, line feed)

Otsakeriveillä välitetään parametritietoja

# HTTP-pyyntömetodeja

(HTTP/1.1: <http://www.w3.org/protocols/rfc2616/rfc2616.html>)

- **GET** Nouda objekti (download),
  - nouda objekti vain jos annettu ehto pätee (conditional GET):  
If-Modified-Since, If-Unmodified-Since,  
If-Match, If-None-Match, or If-Range
- **HEAD** Nouda vain otsaketiedot
- **POST** Voidaan myös lähettää tietoa (esim.lomakkeen täyttö)
  - olemassa olevien dokumenttien kommentointi
  - sanomien lähettäminen uutisryhmiin tai ilmoitustauluille
- **PUT** Talleta objekti palvelimelle (upload)
  - polkunimi pyyntörivillä, talletettava tieto runko-osassa
- **DELETE** Poista objekti palvelimelta

Web-julkaisu -  
työkalut käyttävät



# Otsakerivit

Otsakekenttä : kentän arvo

CR

LF

**Host:** **WWW.jokupaikka.fi** kone, jossa dokumentti on

**Connection:** **close** sulje yhteys lähetyksen jälkeen

**User-agent:** **Mozilla/4.0** selaimen tyyppi

**Accept-language:** **fi** dokumentin kieli

Lisätietoja:

[http://en.wikipedia.org/wiki/List\\_of\\_HTTP\\_headers](http://en.wikipedia.org/wiki/List_of_HTTP_headers)

# Esimerkki web-sivusta

```
<html>
<title>Registration</title>
<body>
  <h1>Registration</h1>
  <h2>Please enter your name and email address.</h2>
  <form method="GET" action=http://.....>
  First:<input type = "text" name="first">
  Last:<input type = "text" name="last">
  Email:<input type="text" name="email">
  <input type="submit" value="Submit">
</form>
```

# HTTP-vastaus: yleinen rakenne



statusrivi

versio	SP	statuskoodi	SP	fraasi	CR	LF
--------	----	-------------	----	--------	----	----

otsakerivit

Otsakekentän nimi:	SP	kentän arvo	CR	LF
--------------------	----	-------------	----	----

... Lisää otsakerivejä

Otsakekentän nimi:	SP	kentän arvo	CR	LF
--------------------	----	-------------	----	----

tyhjä rivi

CR	LF
----	----

<b>Runko-osa</b> (entity body)
--------------------------------

# Esimerkki: HTTP-vastaus

Statusrivi

HTTP/1.1 200 OK

Vastauksen  
aikaleima

Connection: close

Date: Thu, 22 Feb 2007 12:00:15 GMT

Server: Apache/1.3.0 (Unix)

Last-Modified: Mon, 29 Jan 2007 09:23:24 GMT

Content-Length: 6821

Datan  
aikaleima

Content-Type: text/html

data data data data data ...

Otsake-  
rivejä

Pyydetty data,  
esim. HTML-  
tiedosto

# HTTP: statuskoodeja ja fraaseja

Vastaussanomien statusrivillä (1. rivillä) esim.:

- 200 OK** Pyyntö onnistui, pyydetty objekti mukana vastauksessa
- 301 Moved Permanently:** Objekti on siirretty, uusi URL on mukana vastauksen otsakekentässä **Location**. Asiakas tekee uuden noudon uudesta URL:sta
- 302 Moved Temporarily** Siirretty tilapäisesti
- 400 Bad Request** Palvelija ei ymmärtänyt pyyntöä
- 403 Forbidden** Ei ole oikeutta lukea pyydettyä tiedostoa
- 404 Not Found** Pyydettyä objektia ei löydetty
- 500 Internal Server Error** Virhe palvelimessa
- 505 HTTP Version Not Supported** Palvelija ei tue asiakkaan käyttämää HTTP-versiota. Syntaksissa on jotain liian uutta tai liian vanhaa.

# Kokeile itse (=leiki selainta)

1. Telnet yhteys kiinnostavaan www-palvelimeen:

```
telnet cis.poly.edu 80
```

avaa TCP yhteyden koneeseen cis.poly.edu (HTTP:n oletusporttiin) 80.

Kaikki mitä kirjoitat siirtyy yhteyttä pitkiin ko. koneeseen ja porttiin

2. Kirjoita GET HTTP pyyntö:

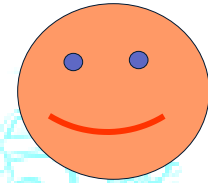
```
GET /~ross/ HTTP/1.1  
Host: cis.poly.edu
```

Tämä on minimaalinen (mutta kelvollinen) GET pyyntö palvelimelle.

Ethän unohda tyhjää riviä viestin lopusta (kaksi rivinvaihtoa.)

3. Tutki saamaasi HTTP vastausta www-palvelimelta!  
(vaihtoehto telnetille: Wireshark - analysointiohjelma)

# Evästeet (cookies)



- **HTTP on tilaton protokolla**
  - Palvelija ei talleta mitään istuntoon liittyvää
- **Selain (eli asiakas!)**
  - Tallettaa asiakaskoneelle (tiedostoon) palvelimen pyynnöstä ja sen tarpeita varten käyttäjäkohtaista tietoa (= evästeen)
  - Lähettää tiedot palvelijalle joka pyynnön yhteydessä.
  - Eväste voi olla sidottu tiettyyn verkkotunnisteeseen tai polkuun

- **Evästeiden talletus ja lähetys**
  - HTTP-vastauksessa otsakerivi: Set-cookie: "tieto"
  - HTTP-pyynnössä otsakerivi: Cookie: "tieto"
- **Palvelin**
  - Ylläpitää tietokantaa käyttäjistä (back-end database)
  - Yksikäsitteiset käyttäjätunnisteet (tav. numero)

Varmenteet viikolla 6

# Mihin evästeitä käytetään?

- Käyttäjien tunnistamiseen
  - Palveluntarjoaja muistaa käyttäjän edellisestä sanomasta
  - Ensimmäisellä käyttökerralla tietojen kyselyä
  - Jatkossa **tunnistuseväste mukana sanomissa**
- Istunnon vaiheen (tilan) tallentamiseen
  - Todennus vain kertaalleen esim. www-postinlukuohjelman yhteydessä
- Ostoskorina
  - Selaile palveluntarjoajan sivuilla ja kerää ostokset koriin.
  - Lähetä lopuksi tilaus

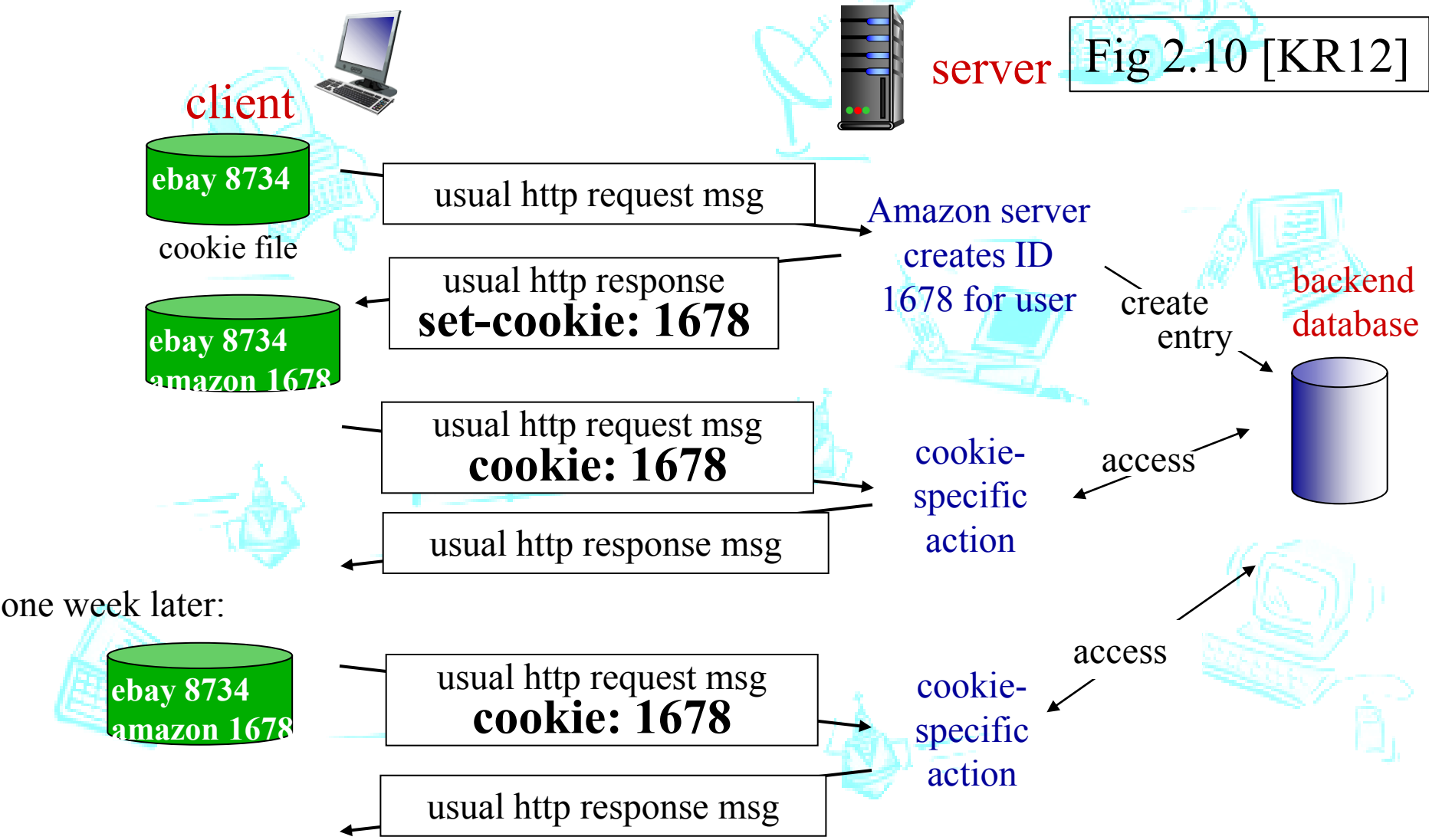
- Väärinkäyttö? Mainosposti?
- Yksityisyys?
  - Palveluntarjoaja saa koottua tietoa käyttäjästä
  - Hakukoneilla voi kerätä lisää.

<http://www.cookiecentral.com>



# Evästeet: esimerkki

Fig 2.10 [KR12]



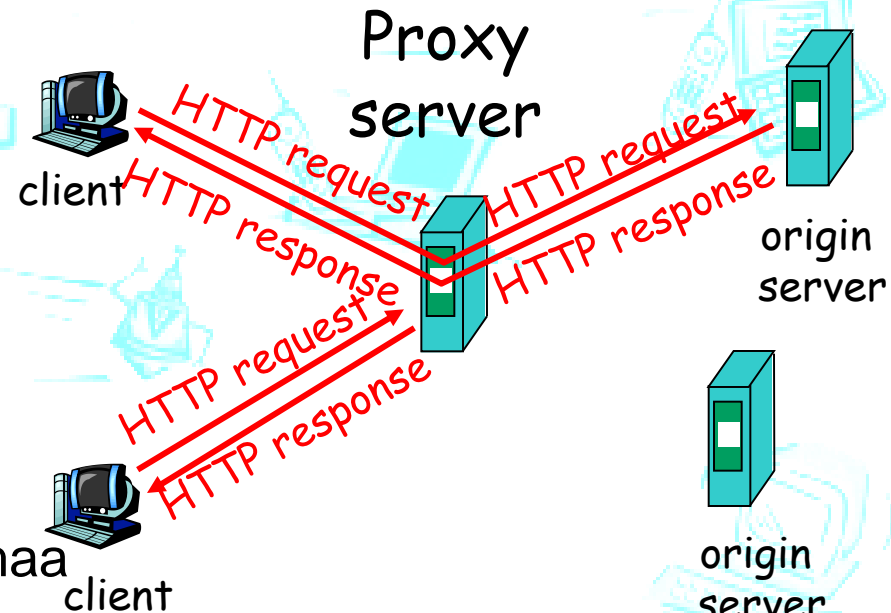
# Evästeiden vaaroja

- Evästeisiin ei pidä varastoida mitään sensitiivistä tietoa selväkielisenä.
- Evästeiden avulla hyökkääjä saattaa anastaa istunnon.
- Sulje istunto aina kun olet lopettanut.
- Poista evästeet koneeltasi säännöllisesti. Tämä palvelee muiden seikkojen ohella yksityisyyden suojaa.

# Proxy-palvelin eli verkkovälimuisti

Fig 2.11 [KR12]

- Säilyttää kopioita haetuista objekteista
- Pyyntö ohjautuu
  - ensin välimuistiin
  - haetaan verkon yli vasta, jos ei löydy välimuistista
- Etuja
  - lyhentää vastausaikaa
  - vähentää verkkoliikennettä
  - vähentää palvelimen kuormaa

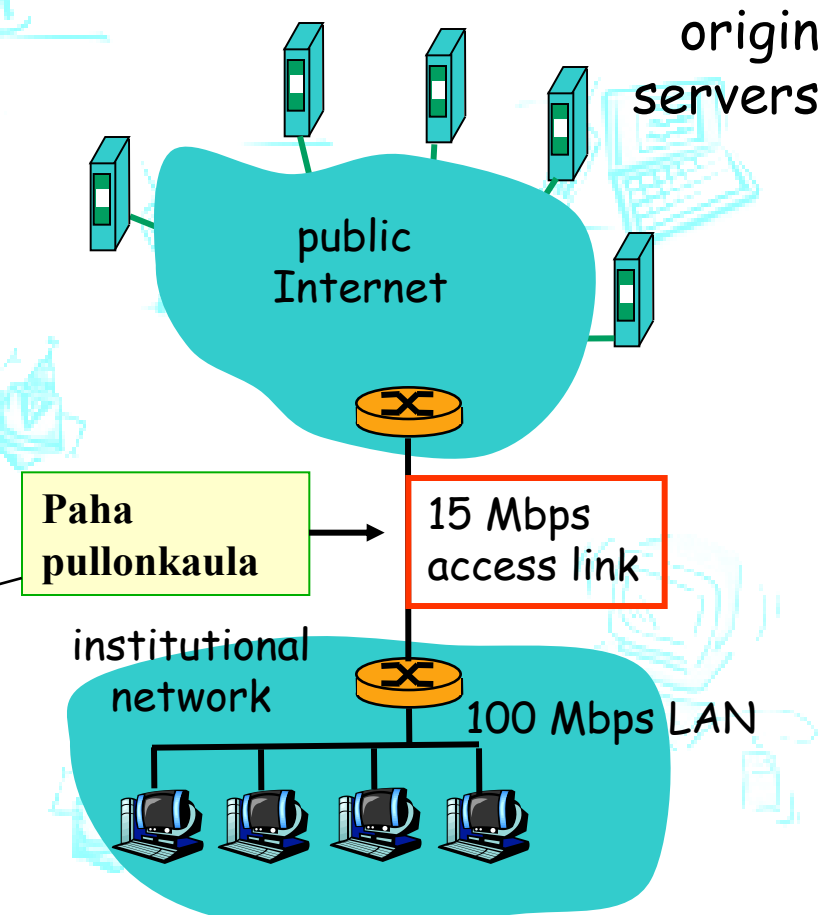


[Myös asiakaskone voi ylläpitää välimuistia!]

# Yhteyslinkin kapasiteetti ei riitä

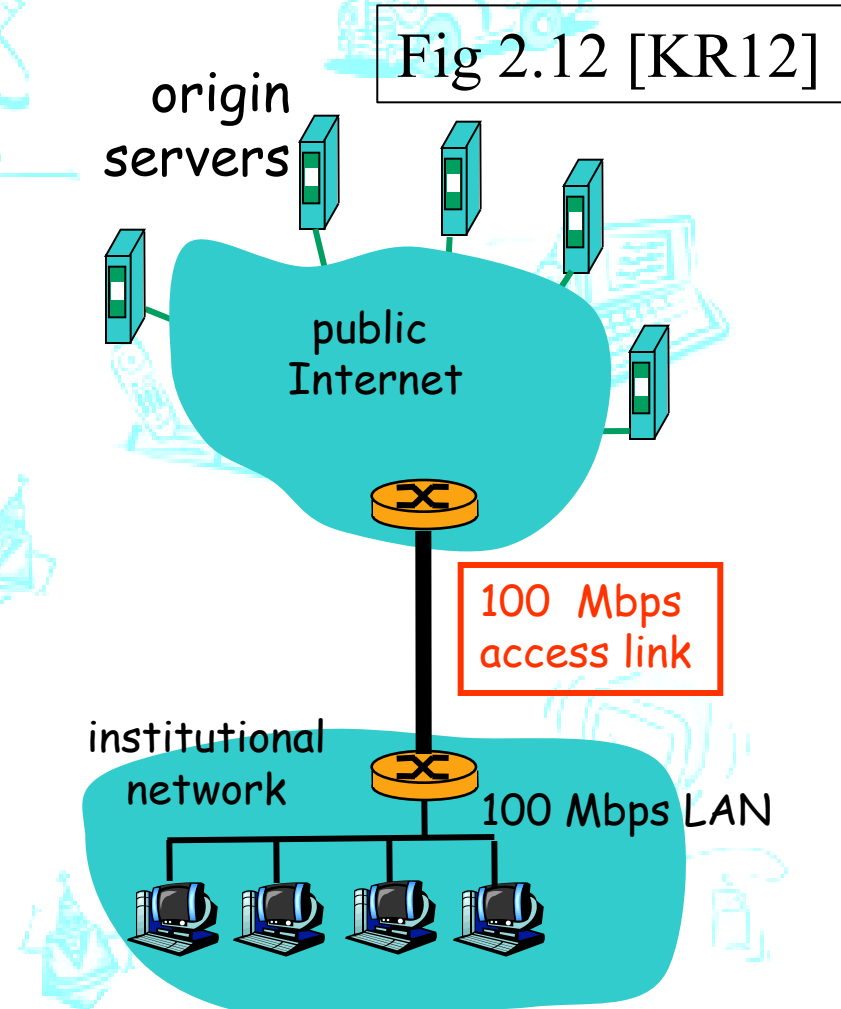
Fig 2.12 [KR12]

- Oletetaan
  - Haettavan objektin koko on 1 Mb
  - 15 pyyntöä/sek => 15 Mbps
  - Viive Internetin reitittimeltä palvelimelle ja takaisin = 2 sec
- Tällöin
  - paikallisverkon käyttöaste = 15%
  - ei ruuhkautunut => siirtoaika muutamia kymmeniä ms
  - Reititinlinkin käyttöaste = 100%
  - Saantiaika = Internet delay + Access delay + LAN delay = 2 sec + mins + msec



# Yhteyslinkin kapasiteetti ei riitä -> Nopeampi yhteys

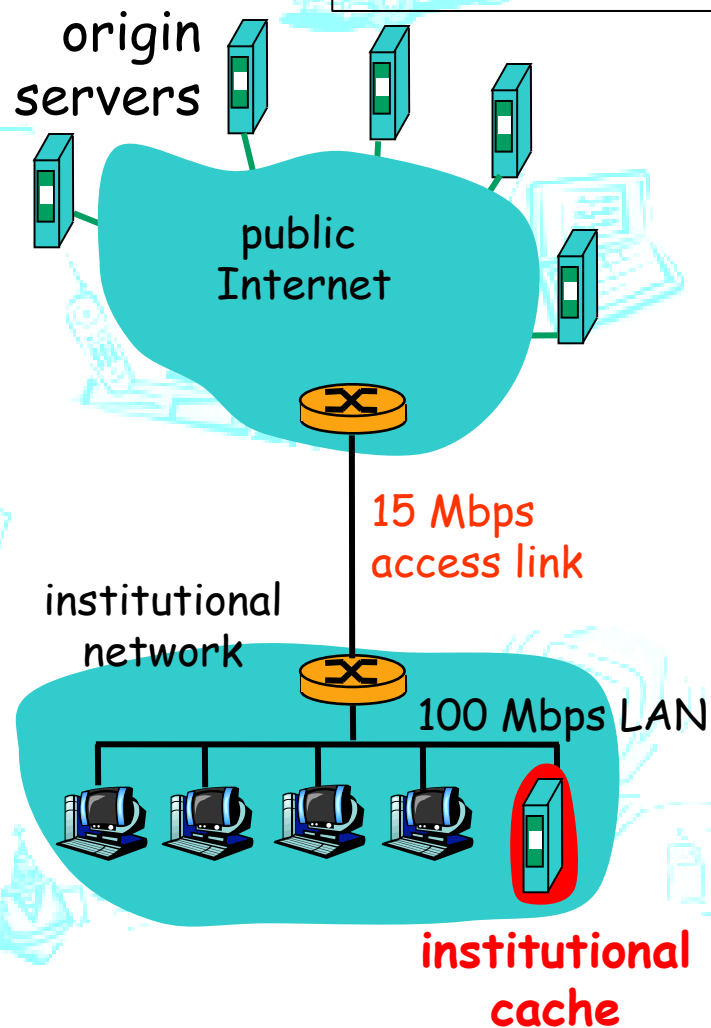
- Parannus?
  - Hankitaan nopeampi yhteys, esim. 100 Mbps
- Tällöin
  - Paikallisverkon käyttöaste = 15%
  - Reititinlinkin käyttöaste = 15%
  - Saantiaika = Internet delay + Access delay + LAN delay = 2 sec + msec + msec
- Mitähän nopeampi linkki maksaa?
  - Voi olla kallis ratkaisu!



# Yhteyslinkin kapasiteetti ei riitä -> Proxy-palvelin

Fig 2.13 [KR12]

- Parannus? Asennetaan proxy-palvelin
- Oletetaan,
  - osumatodennäköisyys (hit rate)= 0,4.  
(tyypillisesti välillä 0,2-0,7)
- Tällöin 40% pyynnöistä löytyy heti läheltä
  - Reititinlinkin käyttöaste putoaa 60%:iin
  - ei jonotusviipeitä, saantiaika ~10 ms
- 60% pyynnöistä palvelimelta saakka
  - Saantiaika = Internet delay + Access delay + LAN delay  
=  $0.6 * (2 + 0,01) \text{ sec} + 0,4 * 0,01 \text{ sec}$   
= 1,2 secs



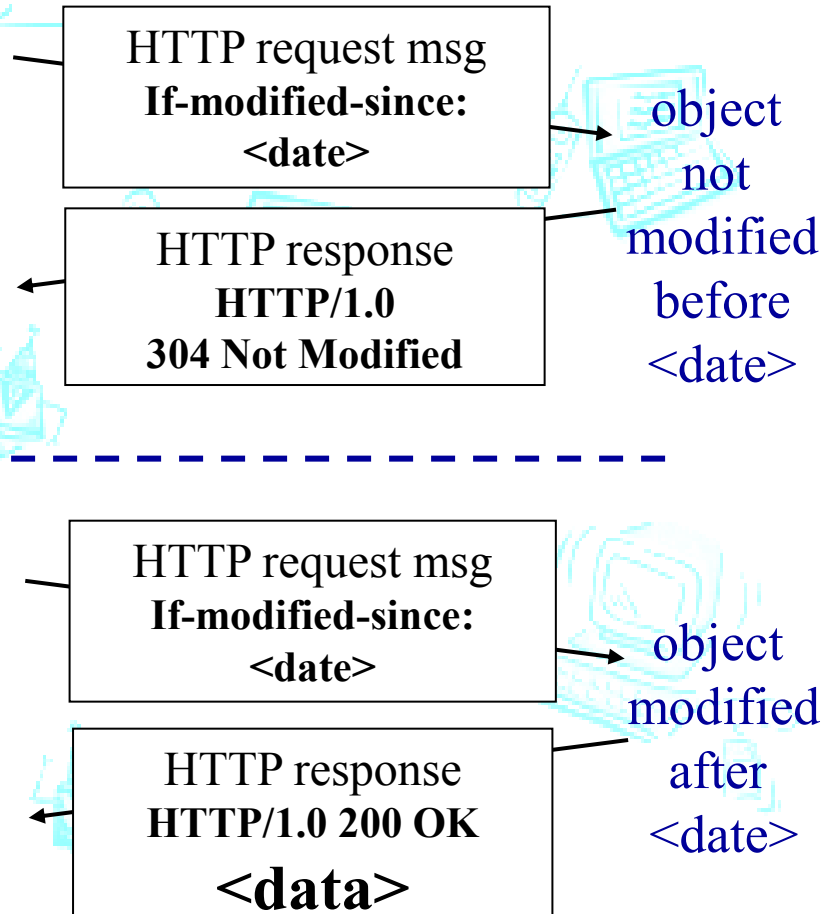
# Conditional GET

- *Tavoite:* Palvelin lähettää objektin uudelleen vain jos välimuistin kopio on vanhentunut (stale)
  - vain vähän siirrettävää tietoa
  - pienempi linkin käyttöaste
- *välimuisti:* kerro tallennetun kopion aikaleima osana HTTP GET pyyntöä  
`If-modified-since: <date>`
- *palvelin:* HTTP vastauksessa ei oliota, jos kopio on ajantasalla:  
`HTTP/1.0 304 Not Modified`

client



server



# Muita URL-osoitteita

- `file:///C:/webs/html/mottle.gif`  
Avaa paikallinen tiedosto ( asiakkaan tiedostojärjestelmässä)  
Selain ei generoi HTTP - pyyntöä, KJ huolehtii
- `ftp://usc.edu/pubs/myfile.doc`  
Hae tiedosto ftp-protokollaa käyttäen
- `news:hy.opiskelu.tht.tili`  
Avaa uutistenlukuohjelman käyttöliittymä ja muodosta yhteys uutispalvelimeen
- `mailto:oskari.olematon@cs.helsinki.fi`  
Avaa postiohjelman käyttöliittymä, välitä sähköposti postipalvelimelle
- `mms:video.avi`  
Avaa multimediasoitin  
Nouda MultiMedia Streaming - protokollaa käyttäen



# XSS-hyökkäys (ei kuulu kurssiin)

- XSS: Cross-Site Scripting
- Hyökkääjä voi ujuttaa koodiaan web-sivulle.  
Esimerkiksi:

```
<html>
```

```
<title>Sign My Guestbook!</title>
```

```
<body>
```

```
Sign my guestbook!
```

```
<form action="sign.php" method="POST">
```

```
<input type="text" name="name">
```

```
<input type="text" name="message" size="40">
```

```
<input type="submit" value="Submit"> </form></body>
```

# XSS(2) (ei kurssiin)

- Kun käyttäjä kirjoittaa kommentin, sivu ohjaa kommentin POST-muuttujan avulla sivulle sign.php.
- Nyt pahantahtoinen käyttäjä voikin kirjoittaa puhtaan tekstikommentin sijasta vaikkapa javascriptiä.
- Jos tämän jälkeen toinen käyttäjä vierailee sivulla, hän laukaisee javascriptin.
- Riippuu javascript-koodista, mihin ongelmiin käyttäjä joutuu.
- Tästä hyökkäyksestä on muita, hienostuneempia versioita.

# XSS(3) (ei kurssiin)

- Paras torjunta on huolellinen syötteiden tarkistus ja sanitointi.
- Eräs varotoimi on estää javascriptien suoritus asiakkaan puolella.
- Hyökkääjät ovat kehittäneet ovelia tapoja kiertää esim. sanitointia. Eräs tapa on käyttää URL-koodia. Jokaisella merkillä on oma URL-koodinsa ja selaimet tuntevat tämän koodiston.
- Toinen hyökkäystapa on pilkkoa koodi pieniin palasiin, jotka sitten yhdistetään myöhemmin.

# CSRF-hyökkäys (ei kurssiin)

- CSRF = Cross-Site Request Forgery
- Tarkastellaan yksinkertaista esimerkkiä, jossa käyttäjä puuhailee verkkopankissa tutkimassa tilejään.
- Samaan aikaan, kun pankkiyhteys on auki, hän innostuu surfailemaan sivulle [www.evilsite.com](http://www.evilsite.com), jossa sijaitsee Javascript-koodi:

```
<script>
```

```
document.location="http://www.naivebank.com/
```

```
TransferFunds.php?amount=10000&fromID=1234&toID=5678";
```

```
</script>
```

# CSRF(2) (ei kurssiin)

- Kun käyttäjän selain kohtaa tämän koodin, selain ohjautuu käyttäjän pankkiin ja erityisesti sivulle, joka yrittää siirtää rahaa käyttäjän tililtä.
- Hyökkäys onnistuu, jos käyttäjä on aikaisemmin todentunut pankille.
- Esimerkki on tietenkin epärealistinen, sillä mikään pankki ei siirrä rahoja varmistamatta siirtoa ensin käyttäjältä.
- Toiseksi pankkisivulta ei ilmeisesti voi poistua ilman, että istunto katkeaa.
- Toisaalta monissa sovelluksissa ei ole tällaisia varmistuksia.

# CSRF(3) (ei kurssiin)

- Vältä avaamasta tuntemattomia linkkejä sähköpostissa tai www-sivuilla.
- Henkilökohtaiset tiedot tulisi aina välittää HTTPS:n kautta.
- Tutki URL:t ja ole huolellinen varmenteiden kanssa.
- Sulje selaimen ActiveX- ja Java-toiminnot ja salli Javascript vasta selaimen huomautuksen jälkeen.
- Kirjaudu ulos palveluista kun olet lopettanut palvelun käyttämisen.



# VERKKO-OHJELMOINNISTA

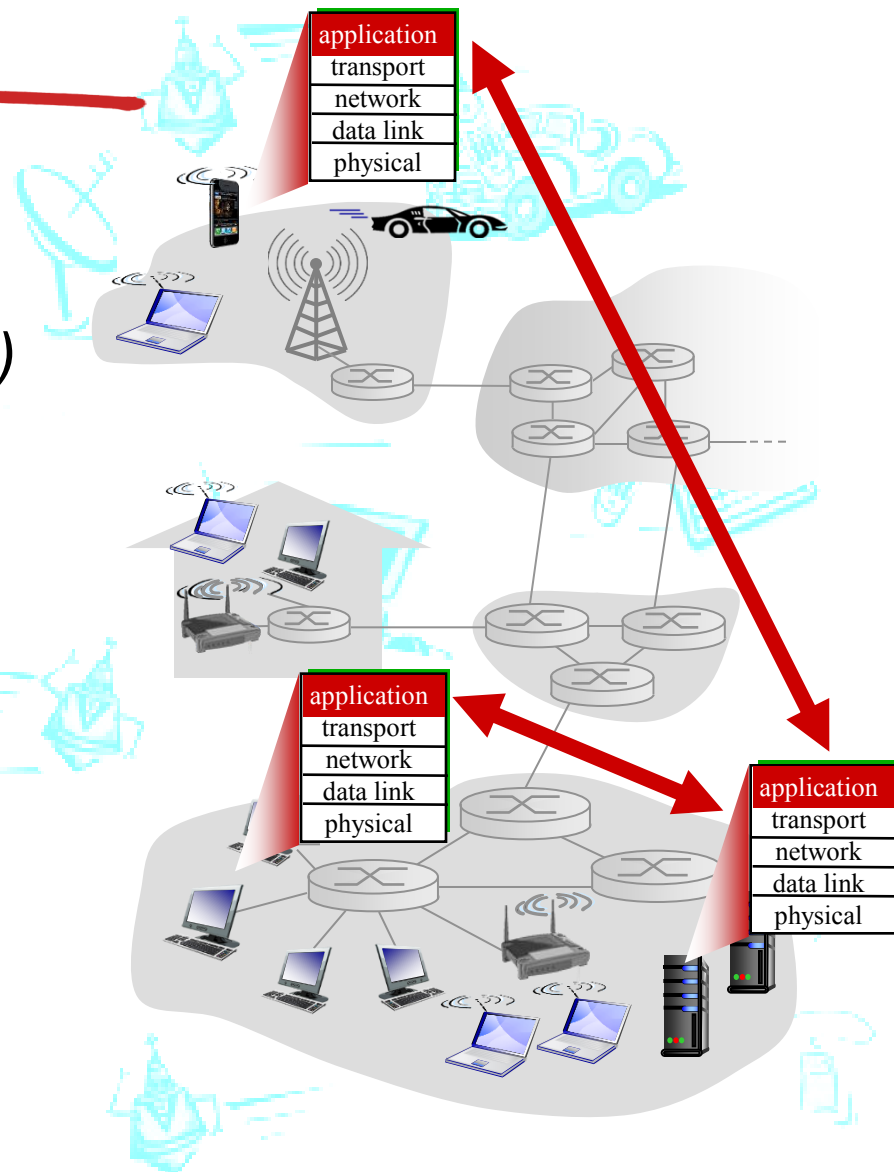
# Verkkosovellus

Kirjoita vain sovellusohjelma:

- Jota suoritetaan päätelaitteissa (*end systems*)
- Joka kommunikoi verkon yli
- Esim. www-palvelu (palvelinohjelma ja selain)

Älä kirjoiteta sovellustason ohjelmaa verkon ytimen laitteille:

- Ne eivät suorita sovellusohjelmia
- Niiden toiminta ei muutu sovellusten muuttuessa





# Pistoke – pikakertaus alkuosan kalvoista

- Sovellus luo pistokkeen (porttinumero KJ:ltä, jos sovellus ei muuta kerro)
  - Yksi pistoke per porttinumero
  - Palvelimella on pysyvä (standardi)portti ja kutakin asiakasyhteyttä varten luotu tilapäinen portti (yhteysportti)
  - Asiakasohjelmalle tilapäinen KJ:n valitsema
- Kaksisuuntainen (full duplex)
  - Samaan pistokkeeseen sekä kirjoitetaan että siitä luetaan
- Lähetys (send) - Kirjoita pistokkeeseen
- Vastaanotto (receive) - Lue pistokkeesta
- Alunperin Berkeley UNIXin (BSD) mukana
- HUOM: Rajapinta sovelluksen (sovelluskerros) ja kuljetuspalvelun(kuljetuskerros) välissä!

# TCP-kuljetuspalvelu

- Yhteyspyyntö palvelun porttiin
- Palvelija luo yhteyttä varten uuden portin
  - Voi palvella useita yhteyspyyntöjä
- Tavallisesti palvelija luo yhteyttä varten myös oman prosessin
- **Lue /kirjoita tavuja**

Welcoming socket = vastaanottopistoke?

Connection socket = yhteyspistoke?

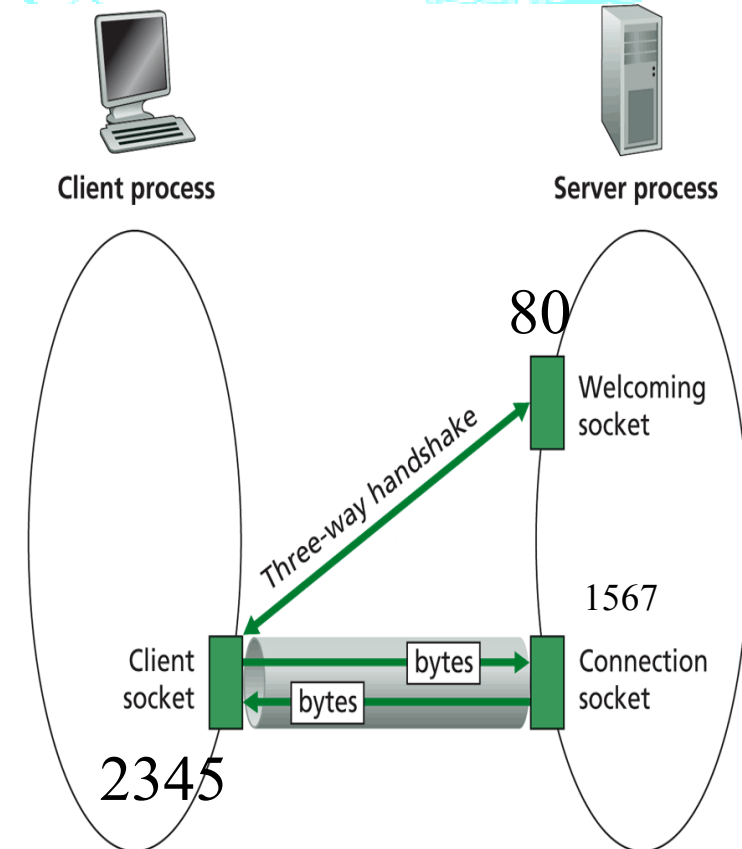
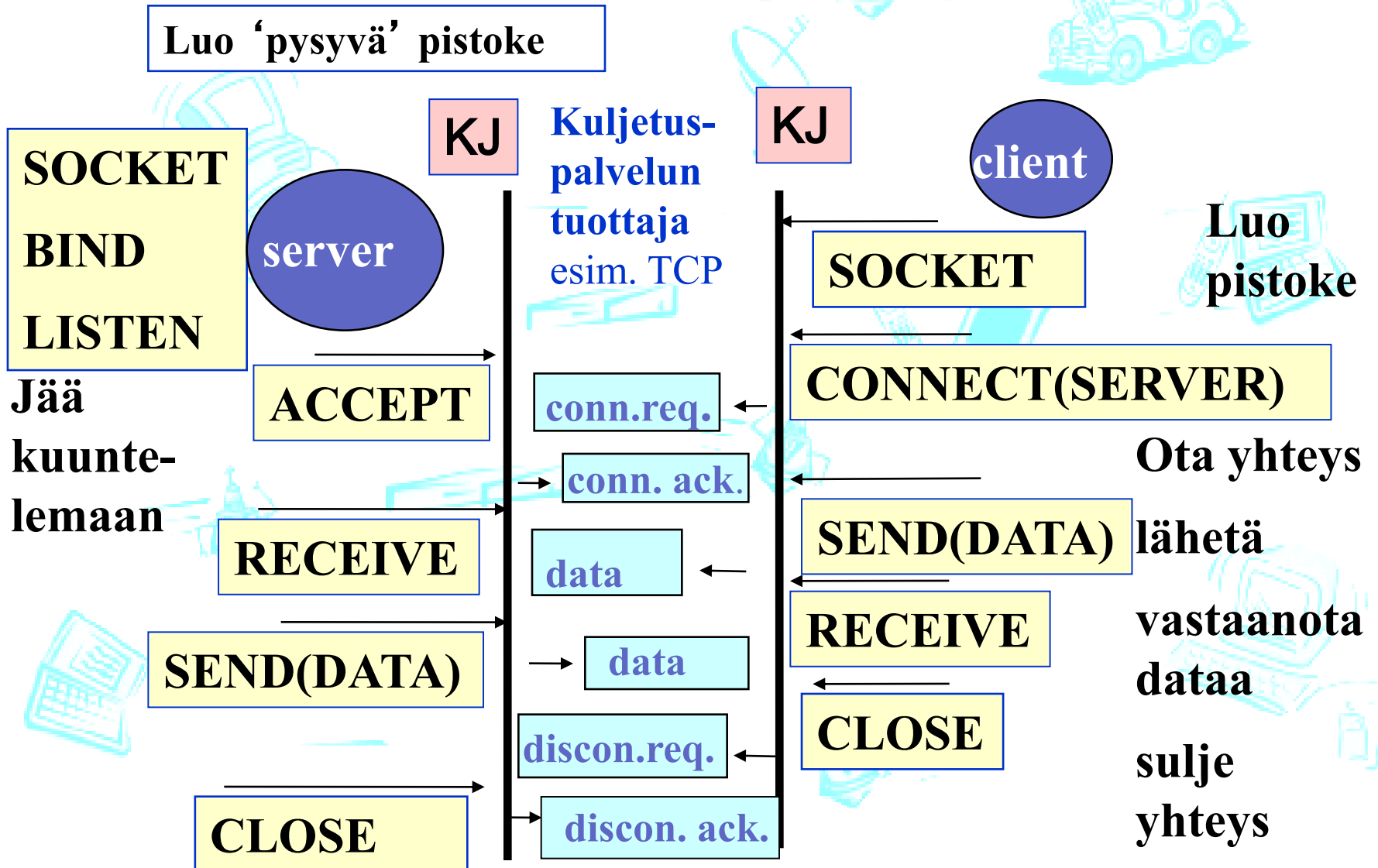
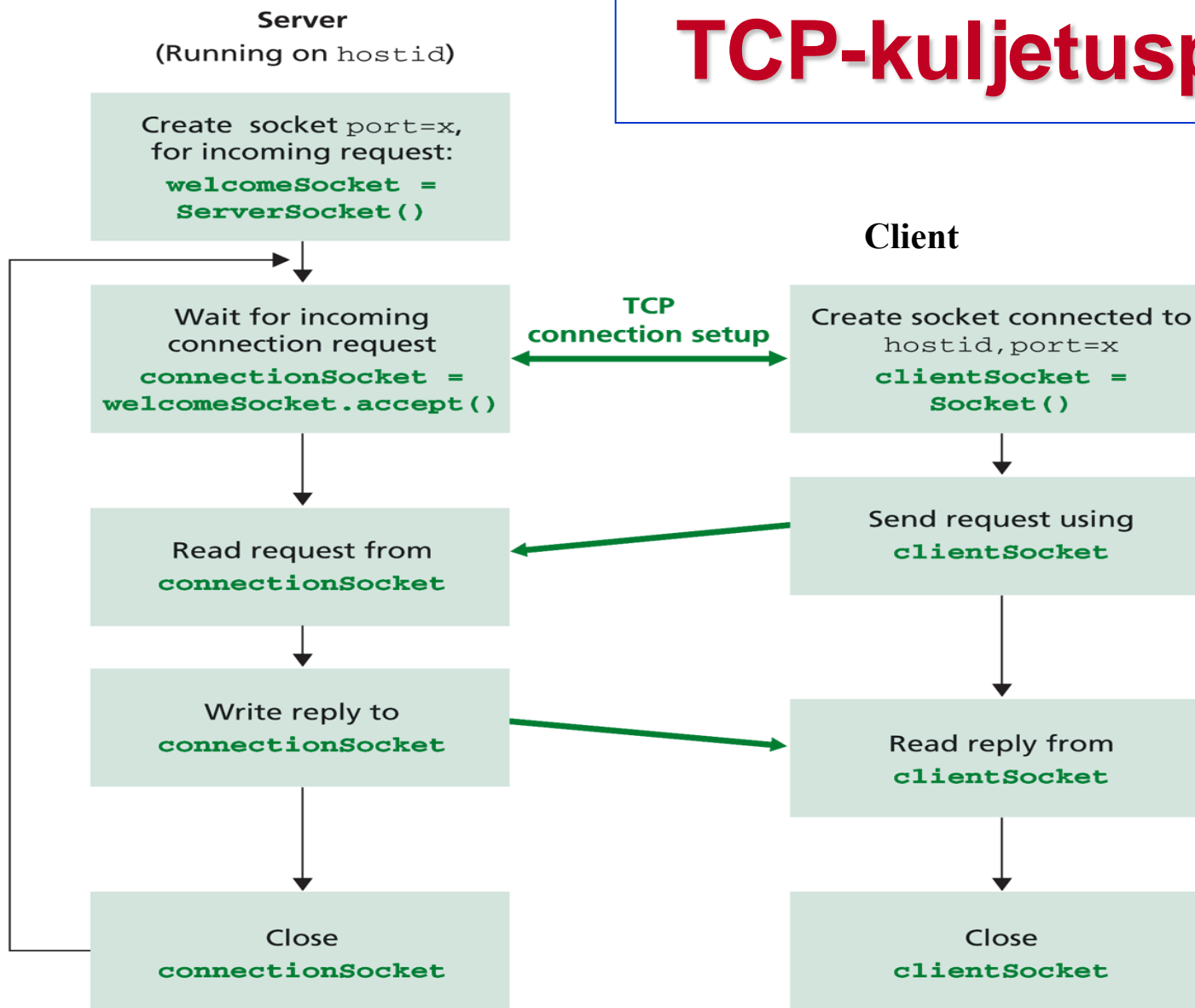


Figure 2.29: Client socket, welcoming socket, and connection socket

# TCP-kuljetuspalvelu



# TCP-kuljetuspalvelu



**Figure 2.28** ♦ The client-server application, using connection-oriented transport services

# Esimerkkisovellus: TCP-asiakas

Pythonin pistoke-kirjasto

## *Python TCPClient*

- from socket import \*
- serverName = 'servername'
- serverPort = 12000
- clientSocket = socket(AF\_INET, SOCK\_STREAM)
- clientSocket.connect((serverName,serverPort))
- sentence = raw\_input('Input lowercase sentence:')
- clientSocket.send(sentence)
- modifiedSentence = clientSocket.recv(1024)
- print 'From Server:', modifiedSentence
- clientSocket.close()

Luo TCP-pistoke palvelimen servername porttiin 12000

Lue käyttäjän näppäilemä syöte

Lähetä viesti palvelimelle. Lähetyksessä ei enää tietoa vastaanottajasta.

Sulje pistoke

# Esimerkkisovellus: TCP-palvelija

## *Python TCP Server*

Luo TCP vastaanottopistoke

Palvelin aloittaa saapuvien  
TCP-pyyntöjen odottamisen

Ikuinen silmukka

Palvelin odottaa `accept()`-  
kutsussa saapuvia pyyntöjä,  
kutsu palauttaa uuden  
pistokkeen kullekin yhteydelle

Lukee tavuja pistokkeesta  
(ei osoitetta kuten UDP)

Sulkee vain yhteyden pistokkeen  
(ei vastaanottopistoketta)

- `from socket import *`
- `serverPort = 12000`
- `serverSocket = socket(AF_INET, SOCK_STREAM)`
- `serverSocket.bind(('', serverPort))`
- `serverSocket.listen(1)`
- `print 'The server is ready to receive'`
- `while 1:`
  - `connectionSocket, addr = serverSocket.accept()`
  - `sentence = connectionSocket.recv(1024)`
  - `capitalizedSentence = sentence.upper()`
  - `connectionSocket.send(capitalizedSentence)`
  - `connectionSocket.close()`

# Vastaava TCP-asiakas Javalla

```
import java.io.*; import java.net.*;
class TCPClient {
    public static void main(String argv[]) throws Exception {
        String sentence;
        String modifiedSentence;
        BufferedReader inFromUser =
            new BufferedReader(new InputStreamReader(System.in));
        Socket clientSocket = new Socket("hostname", 12000);
        DataOutputStream outToServer =
            new DataOutputStream(clientSocket.getOutputStream());
        BufferedReader inFromServer =
            new BufferedReader(new InputStreamReader(
                clientSocket.getInputStream()));
        sentence = inFromUser.readLine();
        outToServer.writeBytes(sentence + '\n');
        modifiedSentence = inFromServer.readLine();
        System.out.println("FROM SERVER: " + modifiedSentence);
        clientSocket.close();
    }
}
```

yhteyspyyntö

Sulkee myös TCP-  
yhteyden

# Vastaava TCP-palvelija Javalla

```
import java.io.*; import java.net.*;
class TCPServer {
    public static void main(String argv[]) throws Exception {
        String clientSentence;
        String capitalizedSentence;
        ServerSocket welcomeSocket = new ServerSocket(12000);
        while(true) {
            Socket connectionSocket = welcomeSocket.accept();
            BufferedReader inFromClient = new BufferedReader(
                new InputStreamReader(connectionSocket.getInputStream()));
            DataOutputStream outToClient =
                new DataOutputStream(connectionSocket.getOutputStream());
            clientSentence = inFromClient.readLine();
            capitalizedSentence = clientSentence.toUpperCase() + '\n';
            outToClient.writeBytes(capitalizedSentence);
        }
    }
}
```

Yhteyspistokkeen  
luonti

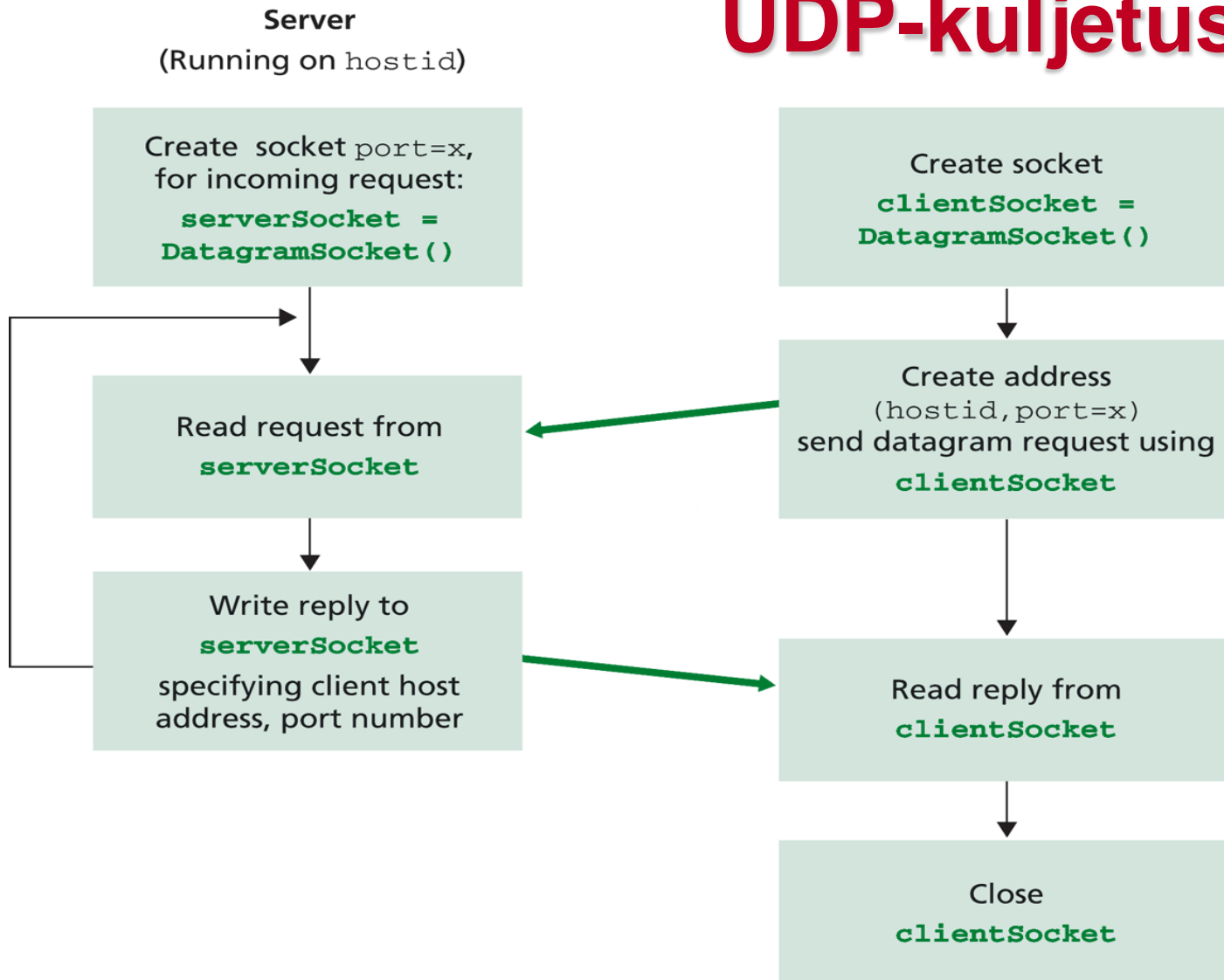
Palvelijan toiminta  
pyynnön ja  
vastauksen välissä



# UDP-kuljetuspalvelu

- Ei kättelyä, yhteydenmuodostusta /purkua
- Ei-luotettava
- Sovellusprosessi lukee ja kirjoittaa kokonaisia yksittäisiä sanomia
- Lähettäjä kertoo KJ:lle sanoman lisäksi kohteen IP-osoitteen ja portin
  - POSIX: `send(sockfd, msg[], msg_len, flags, addr[], addr_len)`
- Vastaanottaja saa KJ:ltä mahdollista vastausta varten lähettäjän IP-osoitteen ja portin
  - POSIX: `recvfrom(sockfd, msg[], msg_len, flags, addr[], *addr_len)`

# UDP-kuljetuspalvelu



**Yhteydetön**  
**Epä-**  
**luotettava**  
**Ei takaa**  
**sanomien**  
**järjestystä**

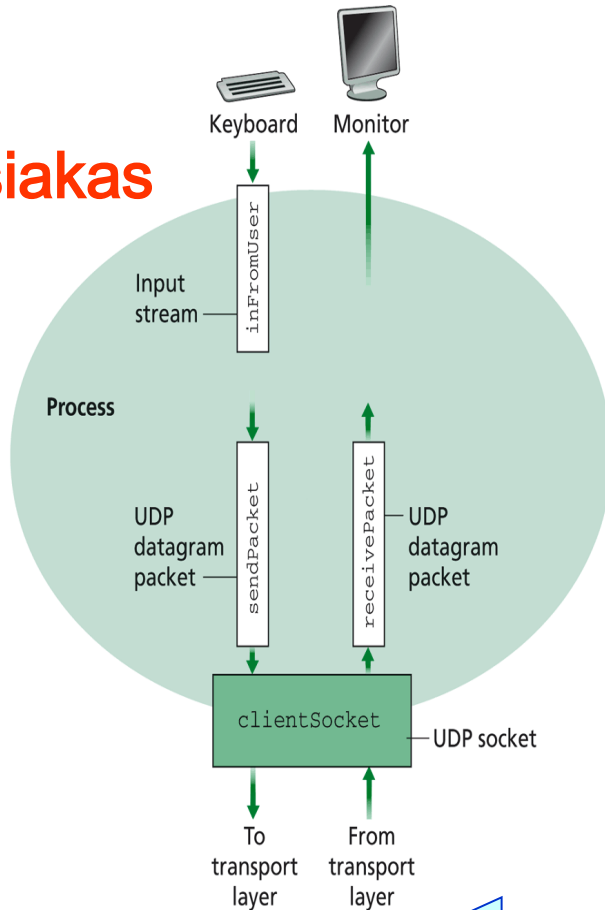


**Figure 2.2** ♦ The client-server application, using connectionless transport services

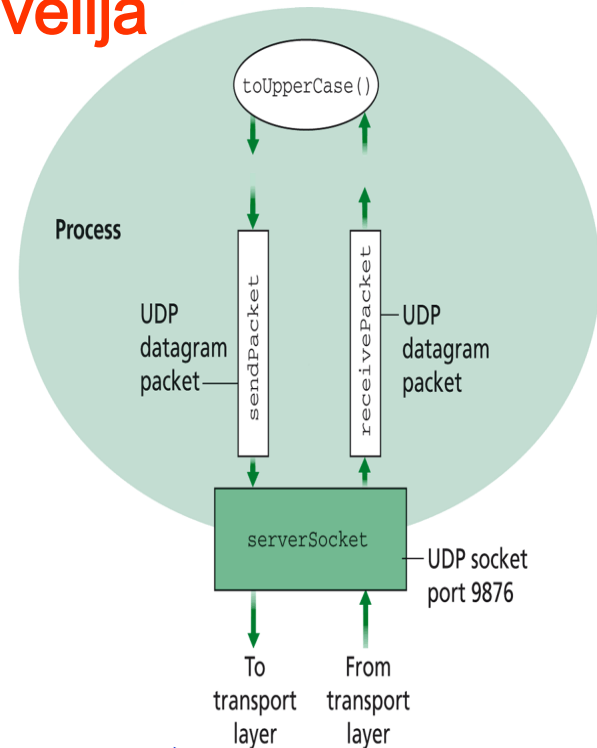
# UDP-esimerkki

Aiempi painos  
KuRo08: Fig 2.35 ja 2.36

Asiakas



Palvelija



Kuljetuspalvelu

# Esimerkkisovellus: UDP-asiakas

## *Python UDPClient*

```
from socket import *
serverName = 'hostname'
serverPort = 9876
clientSocket = socket(socket.AF_INET,
                       socket.SOCK_DGRAM)
message = raw_input('Input lowercase sentence:')
clientSocket.sendto(message,(serverName, serverPort))
modifiedMessage, serverAddress =
    clientSocket.recvfrom(2048)
print modifiedMessage
clientSocket.close()
```

Luo UDP-pistoke  
palvelimen  
hostname  
porttiin 9876

Liitä pistokkeeseen  
palvelimen nimi ja portti:  
lähetä merkkijono

Lue vastaus pistokkeesta

Tulosta vastaus;

Sulje pistoke

# Esimerkkisovellus: UDP-palvelija

## *Python UDPServer*

```
from socket import *
serverPort = 9876
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(("", serverPort))
print "The server is ready to receive"
while 1:
    message, clientAddress =
serverSocket.recvfrom(2048)
    modifiedMessage = message.upper()
    serverSocket.sendto(modifiedMessage,
clientAddress)
```

Luo UDP-pistoke

Sido pistoke paikalliseen  
porttiin numero 9876

Ikuinen silmukka

Lue sanoma UDP-  
pistokkeesta, sanomassa on  
mukana asiakkaan IP ja  
porttinumero

Lähetä muokattu  
merkkijono (takaisin)  
asiakkaalle

# Vastaava UDP-asiakas Javalla

```
import java.io.*; import java.net.*;
class UDPClient {
    public static void main(String args[ ]) throws Exception {
        BufferedReader inFromUser =
            new BufferedReader(new InputStreamReader(System.in));
        DatagramSocket clientSocket = new DatagramSocket();
        InetAddress IPAddress = InetAddress.getByName("hostname");
        byte[ ] sendData = new byte[1024];
        byte[ ] receiveData = new byte[1024];

        String sentence = inFromUser.readLine();
        sendData = sentence.getBytes();
        DatagramPacket sendPacket =
            new DatagramPacket(sendData, sendData.length, IPAddress, 9876);
        clientSocket.send(sendPacket);
        DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
        clientSocket.receive(receivePacket);
        String modifiedSentence = new String(receivePacket.getData());
        System.out.println("FROM SERVER:" + modifiedSentence);
        clientSocket.close();
    }
}
```

IP-osoitteen  
selvittäminen!

Vapauttaa pistokkeen

# Vastaava UDP-palvelija Javalla

```
import java.io.*; import java.net.*;
class UDPServer {
    public static void main(String args[]) throws Exception {
        DatagramSocket serverSocket = new DatagramSocket(9876);
        byte[] receiveData = new byte[1024];
        byte[] sendData = new byte[1024];
        while(true) {
            DatagramPacket receivePacket =
                new DatagramPacket(receiveData, receiveData.length);
            serverSocket.receive(receivePacket);
            String sentence = new String(receivePacket.getData());
            InetAddress IPAddress = receivePacket.getAddress();
            int port = receivePacket.getPort();
            String capitalizedSentence = sentence.toUpperCase();
            sendData = capitalizedSentence.getBytes();
            DatagramPacket sendPacket =
                new DatagramPacket(sendData, sendData.length, IPAddress, port);
            serverSocket.send(sendPacket);
        }
    }
}
```

**Pura paketti:  
data, IP-osoite  
ja portti**

**Lähetä muokattu data.**