

Tīmekļa programmatūras īpašības

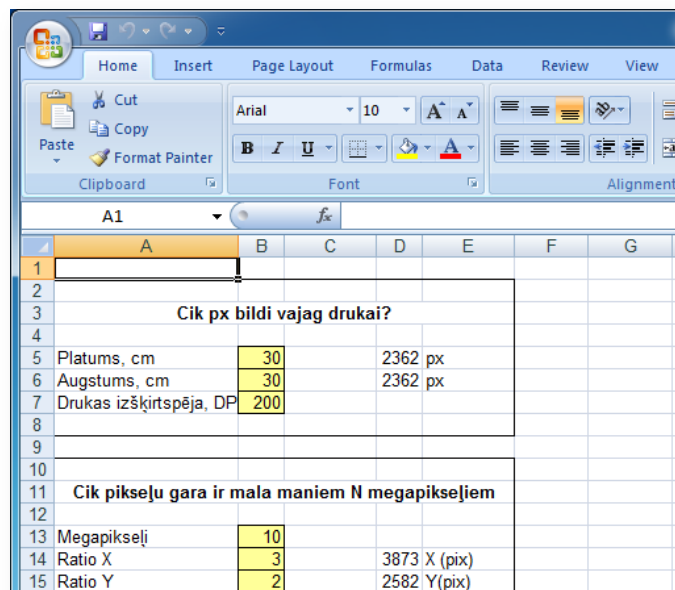
Krišs Rauhvargers
Tīmekļa tehnoloģijas II,
2011

Datu apstrādes sistēmas

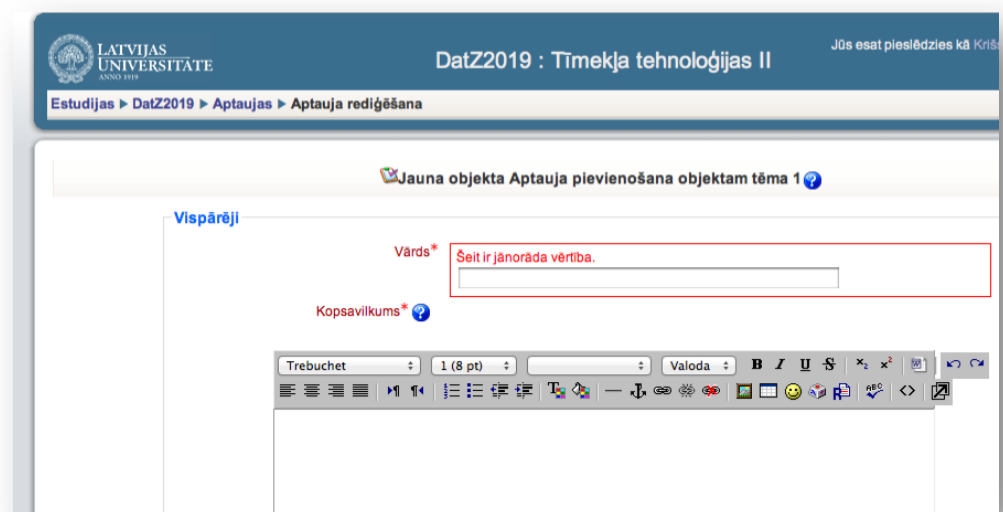
- Datu apstrādes sistēmām raksturīgas funkcijas
 - Datu glabāšana
 - Datu atlaides funkcionalitāte
 - Datu prezentācija lietotājam
 - Lietotāja veikto darbību apstrāde
 - Datu izmaiņu apstrāde

Klasiskā un tīmekļa programmatūra

Izpildāma “klasiskā”
lietojumprogramma



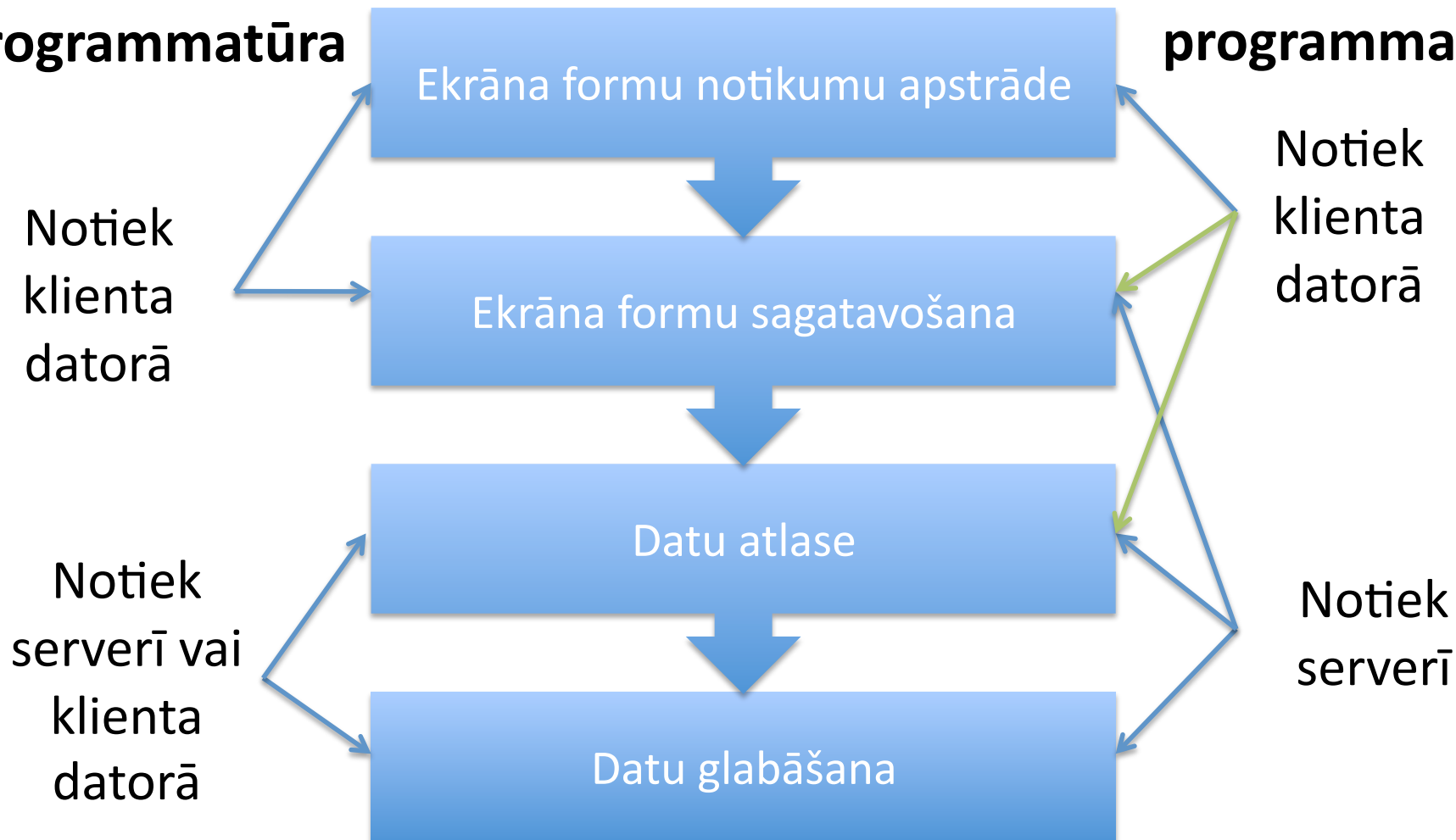
Tīmekļa
lietojumprogramma



Daudzslāņu programmatūra

**Klasiskā
programmatūra**

**Tīmekļa
programmatūra**



1. problēma: Sagatavotība darbam

- Tīmekļa programmatūrā klienta datorā ir vienīgi pārlūkprogramma
 - Klasiskā programmatūrā: uz klienta datora ir instalēts viss nepieciešamais
- Viss darbam nepieciešamais jāielādē no servera
- Kešatmiņa ļauj atkārtotas ielādes padarīt ātrākas

2. problēma: tīkla ātrums (lēnums)

- Slāņu izvietojums
 - Klasiskā programmatūrā slāņi ir *tuvu*
 - izmaiņas prezentācijas slānī viegli transportēt līdz datu glabātuvei
 - Tīmekļa programmatūrā lietotāja starp ekrāna formām un pārējiem slāņiem ir internets

Datu pārraide starp slāņiem

- Datu apstrādes loģika parasti ir uz servera
- Izmaiņas prezentācijas slānī izraisa datu izmaiņu pieprasījumus
- Katra lasīšana ir atsevišķs pieprasījums datu slānim
- Jauna pieprasījuma rezultāts ir jauns HTML dokuments, kas jāpārzīmē uz ekrāna
- Mūsdienu risinājumi
 - Prezentācijas sagatavošanu veikt pie klienta
 - Glabāt datus pie klienta

Ātruma uzlabojumi: AJAX

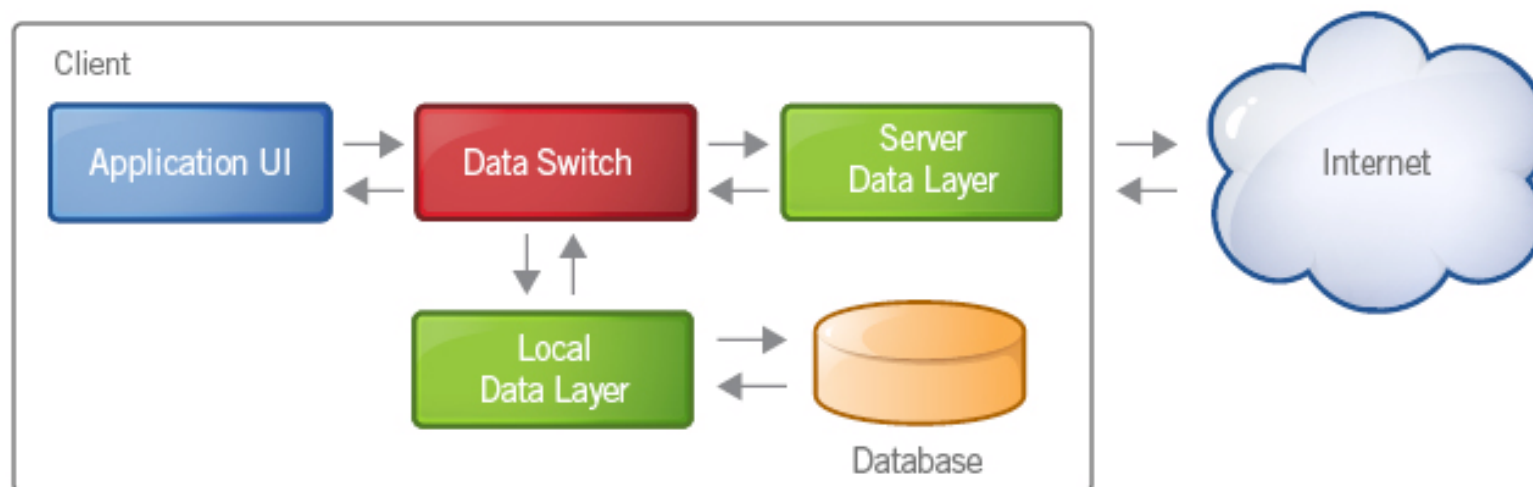
- AJAX ļauj pēc lapas ielādes nosūtīt datus uz serveri vai nolasīt papildus datus no servera
- Izmaiņas prezentācijas slānī veic ar JavaScript

Ātruma uzlabojumi: lieku datu nesūtīšana

- HTTP HEAD metode
 - Datra to pašu, ko GET, bet nesūta atpakaļ saturu
- HTTP ETAG papildinājums
 - Katrai resursa versijai piešķir tagu
 - Nākamajos pieprasījumos lietotājs informē, kura versija jau glabājas kešā
 - Ja klientam ir jaunākā versija, serveris atbild ar "304 Not modified"

Ātruma uzlabojumi: klienta datu glabātuves

- Ieviešot nelielu lokālu datu glabātuvi klienta datorā, iespējams samazināt nepieciešamo datu apmaiņu
 - Lapa jāveido dinamiski, piemēram, ar JavaScript, kas rūpējas par lapas izskata maiņu
 - Tikai brīdī, kad izmaiņas pilnībā pabeigtas, tās nosūta uz serveri
- Realizācijas: Google gears, Web SQL Database



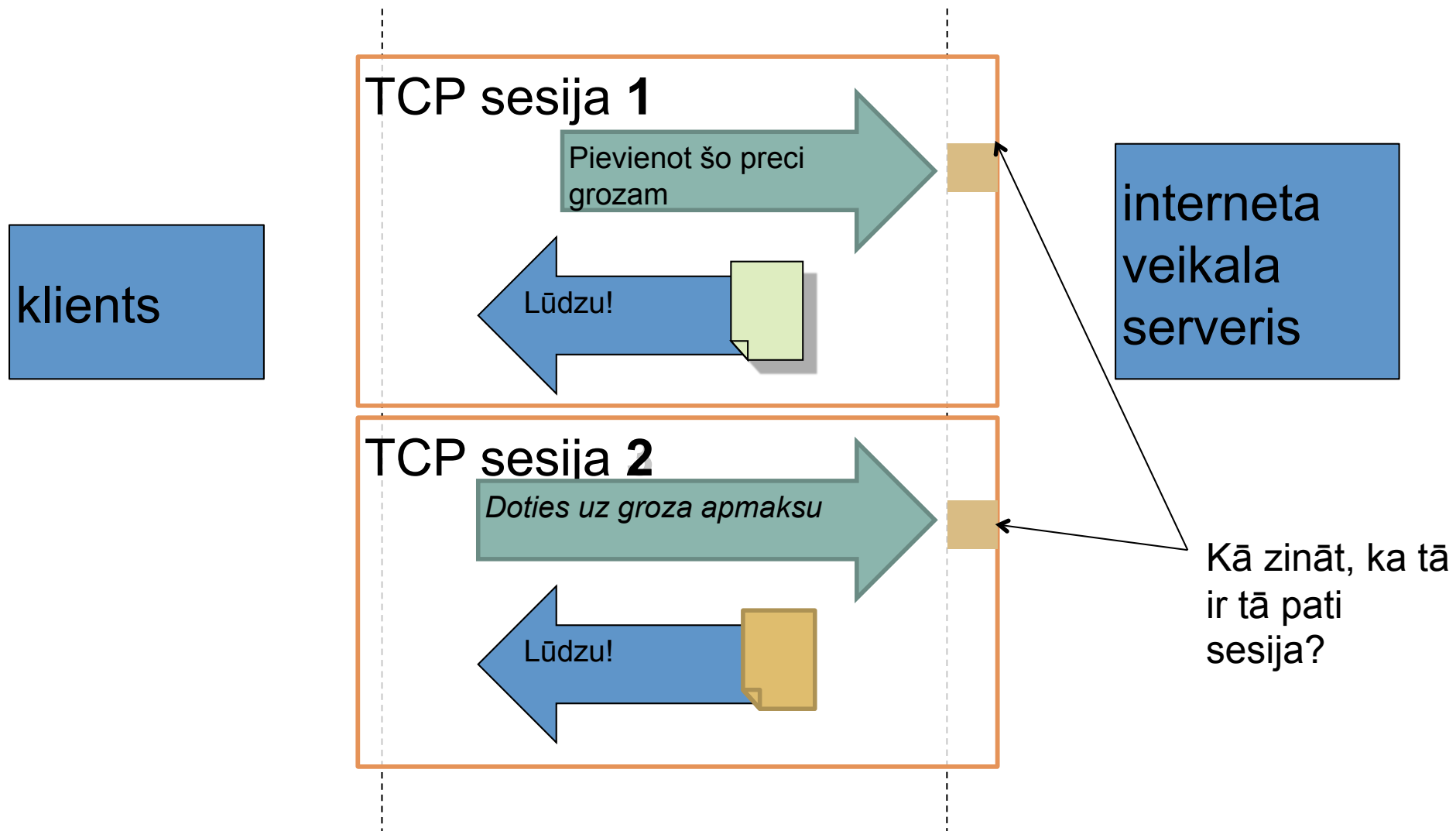
Google Gears uzbūve

<http://code.google.com/apis/gears/architecture.html>

3. problēma: lietojumprogrammas stāvokļa saglabāšana

- Katrs pieprasījums uz serveri ir neatkarīgs no citiem pieprasījumiem
- Personalizācijai, autentifikācijai u.c. nepieciešams *sesijas (seansa)* jēdziens
 - sesija sākas, kad lietotājs sūta pirmo pieprasījumu serverim
 - beidzas, kad lietotājs aizver pārlūkprogrammas logu
- Klienta pusē tas ir zināms, serverī - ne

Lietojuma scenārijs kā vaicājumu virkne



Secīga pieprasījuma atpazīšanas mehānismi (1)

- Identifikācija servera pusē pēc
 - IP adreses
 - Tīmekļa pārlūkprogrammas versijas
 - *Problēma*: vienai IP adresei var atbilst vairāki datori

Secīga pieprasījuma atpazīšanas mehānismi (2)

- *Viltīgie scenāriji*
 - Katram lietotājam sava, pielāgota adrese
 - atklātā veidā
 - `http://example.com/johnb`
 - paslēpti
 - `http://example.com/?x=1a2s3d4f5g`
 - *Problēma*: citiem pārsūtītas adreses
 - Formas dati
 - `<form method="POST">`
`<input type="hidden" name="user" value="john" />`
`</form>`
 - *Problēma*: nedrīkst izmantot ``

Risinājums - *cookies*

- 1997. gads “HTTP State Management Mechanism”
 - <http://www.ietf.org/rfc/rfc2109.txt>
- “Cookie” jeb sīkdatne ir datu kopums, ko serveris drīkst noglabāt pie klienta un palūgt klientam “atpakaļ”, kad nepieciešams
 - parasti tas tiek saglabāts atsevišķā nelielā failā
 - strikti ierobežojumi, kurš un kā drīkst piekļūt sīkdatnēm

Sīkdatnes

- Dodot klientam sīkdatni, serveris norāda
 - nosaukumu
 - saturu
 - derīguma termiņu
 - ja nav: derīgs līdz šīs sesijas beigām
 - derīguma apgabalu (domēnvārds, taka)
- Saņemot sīkdatni, klients
 - saglabā to pie sevis
 - pievieno katram nākamajam pieprasījumam
 - dzēš, kad beidzies derīguma termiņš

Sīkdatņu īpatnības

- Sīkdatne ir izvietota klienta datorā
 - Klients var iztīrīt sīkdatņu glabātuvi
 - Klients var mainīt sīkdatņu vērtības
- Likums #1: Neuzticēties nekādiem datiem, kas saņemti no lietotāja!
- Nedrīkst glabāt
 - Tiešā veidā izmantojamus faktus:
 - Lietotāja vārdu, paroli (vai pat paroles hešu)
 - Lietotāja ID
 - Faktu 'lietotājs ir autentificēts'
- Drīkst glabāt
 - Lietotājam tiešā veidā nederīgu informāciju
 - Informāciju, ko lietotājam nav izdevīgi mainīt

Sesijas identificēšana

- Lietotāja secīgu pieprasījumu atpazīšanai var lietot šādu scenāriju:
 - Pirmajā pieprasījumā serveris ģenerē sesijas identifikatoru (*sess/D*)
 - Izsniedz klientam *sess/D*, izmantojo sīkdatņu mehānismu
 - Katrā pieprasījumā klients nosūta savu *sess/D* sīkdatni
- Servera pusē saglabā datus, kas attiecas uz šo sesiju (t.s. "sesijas mainīgos", piemēram, iepirkumu groza saturs)

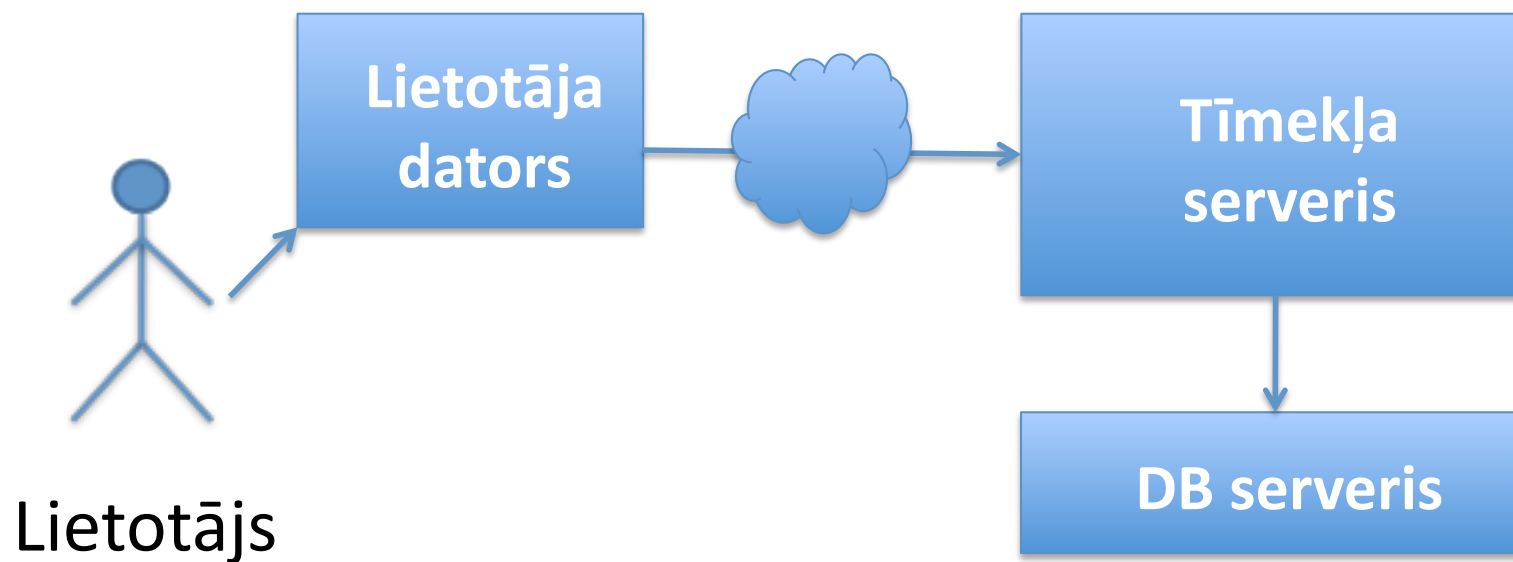
HTTP sesiju īpatnības

- Par katru HTTP sesiju ir zināms, kad tā sākusies,
- .. bet var nebūt ziņu, kad tā beidzas
 - prātīgi lietotāji nospiež “log out”
 - pārējie nē
- Tikai sesijas beigās serveris drīkst no “sesiju glabātuves” izmest tam zināmo sesijas identifikatoru un ar to saistītos datus.
 - HTTP serveri parasti izmanto “timeout” mehānismu, lai atpazītu nelietotās sesijas

Izturīgs sīkdatņu risinājums: EverCookie

- *EverCookie*
 - <http://samy.pl/evercookie/>
- Saglabā sīkdatnes informāciju
 - Parastajās sīkdatnēs
 - Flash sīkdatņu mehānisms
 - HTML5 "local storage", "Session storage"

4. problēma: drošība



Klienta drošība

- Pakalpojuma sniedzējs nav klientam pazīstams: uzticības problēma
- Sensitīvu datu pārsūtīšana internetā
- Sensitīvu datu nodošana pakalpojuma sniedzējam
- Pakalpojuma izmantošana no sveša datora

Pakalpojuma sniedzēja drošība

- Tīmekļa serveris pieejams jebkuram interneta lietotājam
- Jebkurš sistēmas lietotājs var kļūt par *Neo*
- Atvērtā koda programmatūra prasa papildu uzmanību

5. problēma: autentifikācija

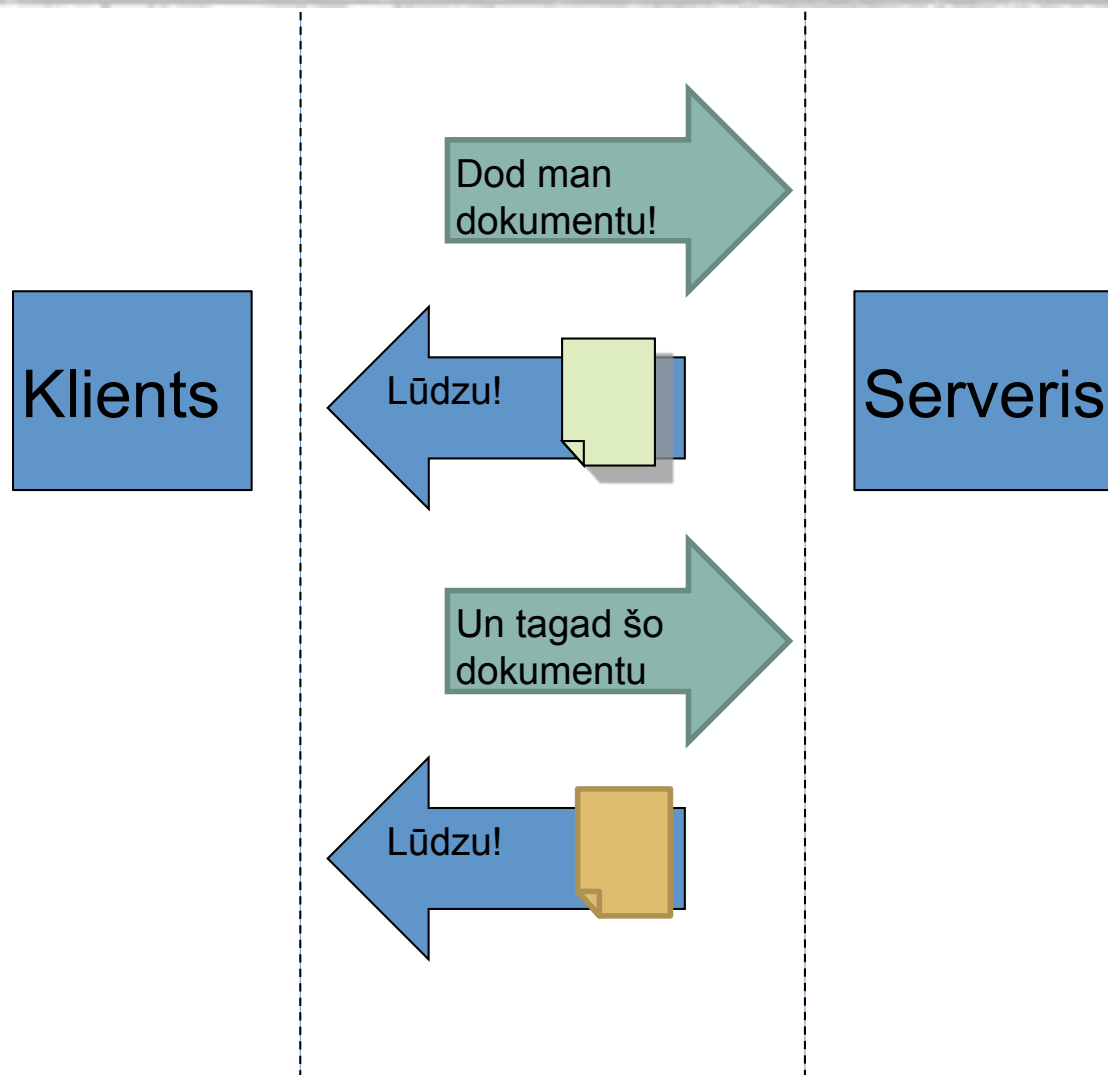
- Klasiski izpildāmā programmatūrā
 - Var uzskatīt, ka lietotājs, kurš pašlaik pieteicies datorā (*logged in*), arī darbina programmatūru
- Tīmekļa programmatūrā
 - URL adresi, kurā programma pieejama, var atvērt jebkurš
- Lietotāju *autentifikācija* ir spēja sistēmiski noteikt, vai lietotājs, kurš pieprasa piekļuvi resursam, ir tas, kuru domājam.

Autenticēta sesija

- Klients sūta paroli/lietotāja vārdu
- Serveris autenticē (salīdzina ar datubāzē esošo informāciju)
- Serveris
 - Izsniedz lietotājam sesijas ID
 - Lokālajā glabātnē saglabā informāciju par klienta autentifikācijas stāvokli
 - "ir autenticējies"
 - "ir administrators", u.c.

6. problēma: vienvirziena HTTP protokols

- HTTP labi der atsevišķu dokumentu nolasīšanai
- Katrs pieprasījums ir "pabeigts"



Klienta iniciatīva HTTP protokolā

- HTTP paredz, ka visa iniciatīva ir no klienta puses
- Pēc dokumenta nosūtīšanas serveris nevar informēt klientu par jaunumiem

Tipiska tīmekļa servera realizācija

```
While (true) {  
    Gaida pieprasījumu  
    Apstrādā pieprasījumu (izpilda CGI, PHP)  
    Sūta atbildi  
}
```

- Pēc atbildes nosūtīšanas parasti CGI process pārtrauc darbu

Apkārtceļi

- Klienta puses veikts "polling" – ik pēc neliela intervāla (ar AJAX) ielādē no servera jaunumus
 - Gmail chat
 - Twitter jaunākie tvīti

Uzlabojumi mūsdienās

- HTML5 paredz "tīmekļa ligzdas" (*web sockets*)
 - Klients pieslēdzas pie servera un savienojums paliek atvērts līdz viena no pusēm beidz "sarunu"
 - Serveris var sazināties ar klientu jebkurā brīdī
 - Jābūt gan specifiskai pārlūkprogrammai, gan serverim (non-IE)


7. problēma: datu sūtīšana uz serveri

- Sākotnēji(HTTP 0.9, <http://www.w3.org/Protocols/HTTP/AsImplemented.html>) paredz tikai GET pieprasījumus
 - `http://example.com/doc.htm`
 - `http://example.com/doc.cgi?a=b&c=d`
- HTTP 1.0 ievieš POST protokolu
 - Klients sūta lapas pieprasījumu un pēc tā – datu masīvu
 - HTML tiek papildināts "Web Forms" – iespēju aizpildīt šo datu masīvu

8. problēma – tīmekļa formas

- RFC 1866 (1995. gada novembris, <http://tools.ietf.org/html/rfc1866>) papildina HTML standartu ar FORM, INPUT elementiem
- RFC 1867 (<http://tools.ietf.org/html/rfc1867>) definē un failu augšuplādes iespēju, izmantojot *input type="file"*

Tīmekļa formu elementu trūkumi



Teksta un paroles ievade
Faila augšuplāde
Ķekškaste
Radiopoga
Izkrītošā izvēlne
Poga

Vairāk nav!

- Trūkst:
 - Kalendāra kontroles
 - *Slaidera*
 - Progresā rādītāja
 - Validatoru

Formu uzlabojumi mūsdienās

- HTML5 paredz
 - URL, e-pasta, datuma, laika, skaitļu ievadi, Intervālu ievadi, krāsu izvēlni
 - iespēju norādīt regulāro izteiksmi kā validatoru

9. problēma: grafika

- Galvenais attēlu veids tīmeklī ir rastra attēli
- Nav ērta veida, kā dinamiski ģenerēt attēlus
- HTML5 - CANVAS elements, ļauj klienta pusē "zīmēt"
 - Piemēram
 - grafiki ar jqPlot: <http://www.jqplot.com/tests/>
 - vektorgrafika ar paper.js: <http://paperjs.org/>