STU FIIT
SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA
FACULTY OF INFORMATICS AND INFORMATION TECHNOLOGIES
VGG
Vision & Graphics Group
Institute of Computer Engineering and Applied Informatics

Základy tvorby interaktívnych aplikácií

Sieťové a webové aplikácie

Ing. Peter Kapec, PhD.

LS 2020-21

Obsah

- Statické web stránky
  - HTTP
- Dynamické Web aplikácie
  - CGI, PHP,...
- Aplikácie v prehliadači
- Klient-server aplikácie
- Základ pre moderné web aplikácie
  - Node.js

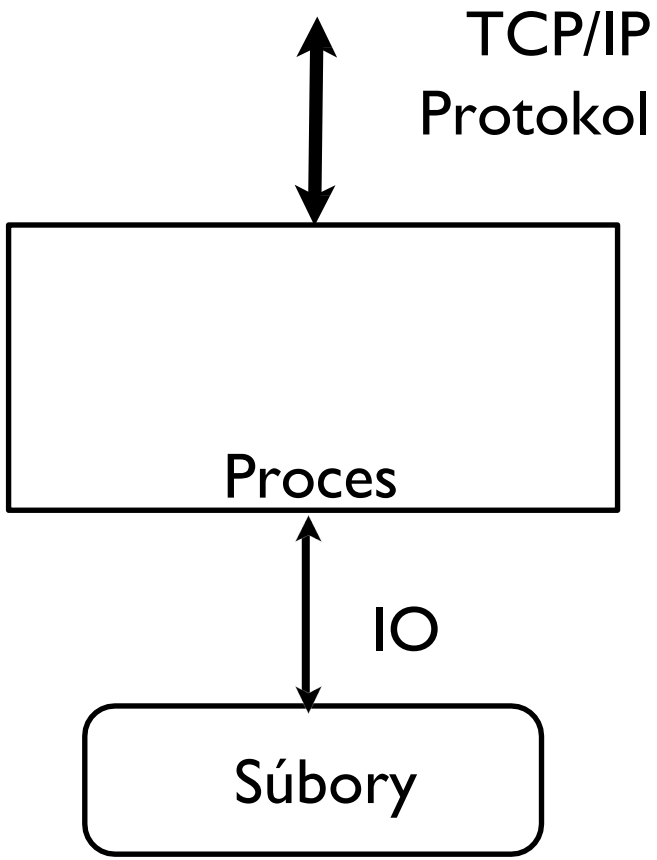
Web aplikácie

- Poznáme mnoho druhov “Web” aplikácií
- Typické sú najme klient-server aplikácie
- Modernejšie aplikácie využívajú asynchrónnu komunikáciu (AJAX, HTML5, Web 2.0)
- Aplikácie bežiace v prehliadači, napríklad Flash aplikácie či JavaScript hry
- ...

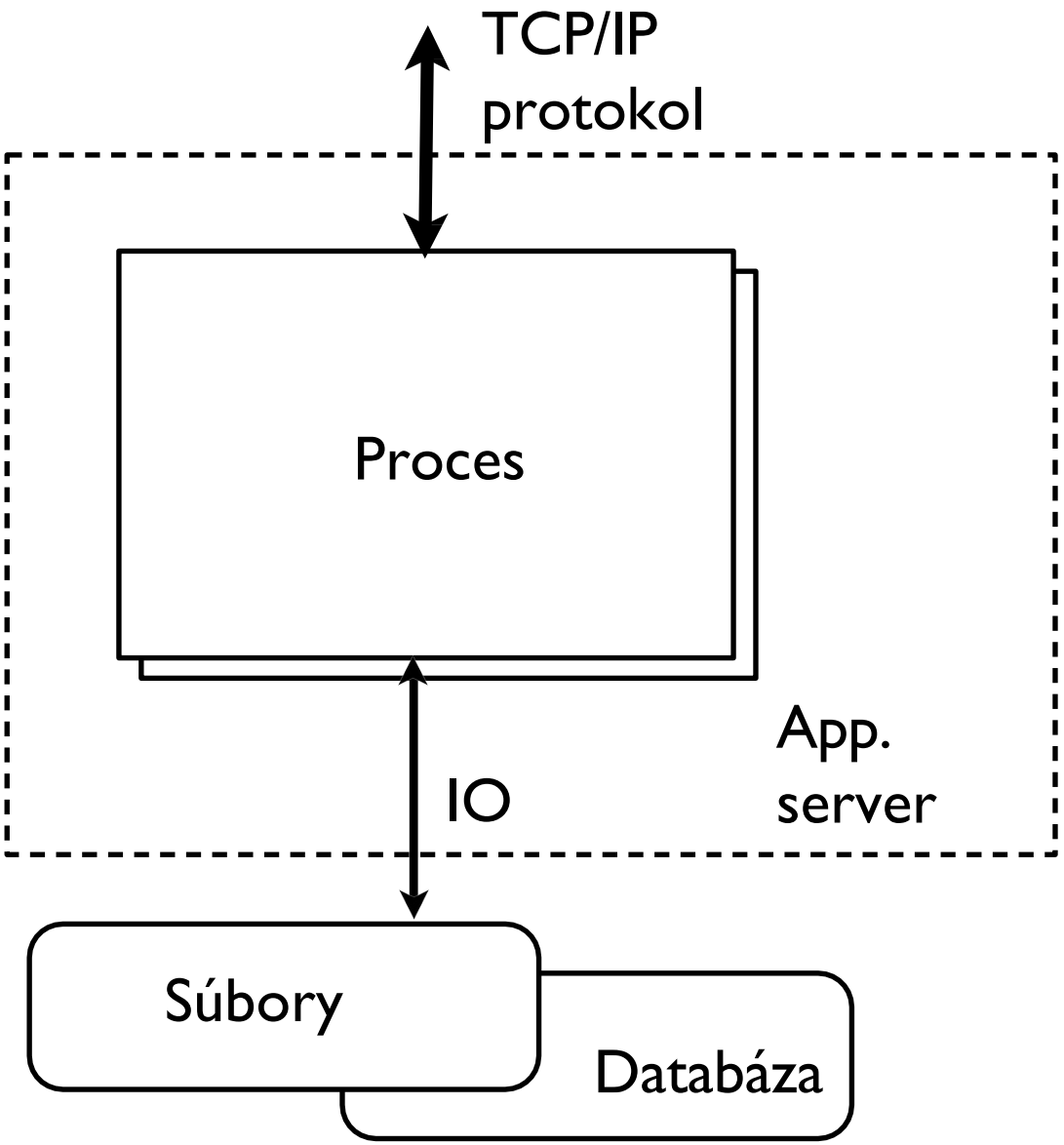
Technológie

- Čokoľvek schopné generovať text
  - CGI - Common Gateway Interface
  - Ruby, RubyOnRails
  - Python, Lua, Java ...
- LAMP - Linux Apache MySQL PHP
  - Typická kombinácia technológií pre bežné web aplikácie
- Node.js - Javascript aj na strane servera
- ...

Sieťová aplikácia

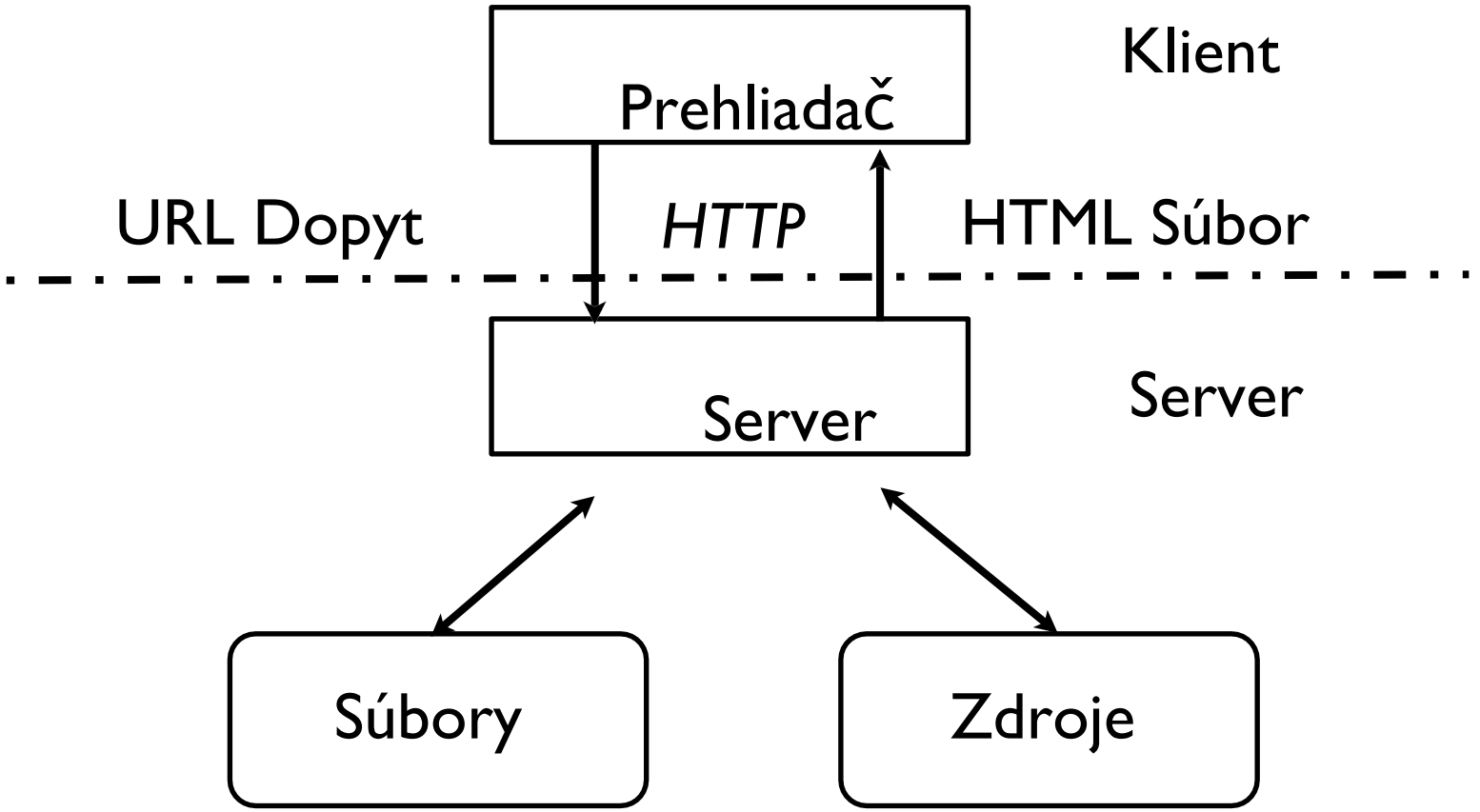


Aplikácie na serveri



Statické web stránky

Web Stránky


Prehliadač
Klient
URL Dopyt
HTTP
HTML Súbor
Server
Server
Súbory
Zdroje

HTTP

- Hyper Text Transfer Protokol
- Definuje pravidlá komunikácie prehliadača so serverom
- Dopyt
  - GET
  - POST
- Odozva
  - Stav
  - Typické odpovede
  - Dynamické HTML

HTTP

- Dopyt (Request)
  - Vyžiadanie stránky (pomocou URL)
  - Typicky HTML dokumenty, obrázky atd.
  - Obsahuje vstupy formulára
  - Obsahuje niektoré nastavenia prehliadača

HTTP

- Odpoveď (Response)
  - Súbor (HTML alebo akýkoľvek iný súbor)
  - Obsahuje vlastnosti dokumentu
  - **Môže smerovať na iné dokumenty**

Dopyt

- Syntax

```
METODA URI PROTOKOL
HLAVICKA1: HODNOTA
HLAVICKA2: HODNOTA
...
HLAVICKAn: HODNOTA

TELO
```

- Príklad

```
GET /index.html HTTP/1.1
host: altair:8000
```

Dopyt

- Komplikovanejší príklad

http://www2.fiit.stuba.sk/ HTTP/1.1
Accept:text/html,application/xhtml+xml,application/xml;q=0.9,*/* q=0.8
Accept-Charset:ISO-8859-1,
utf-8;q=0.7,*;q=0.3
Host:www2.fiit.stuba.sk
Accept-Encoding:gzip,deflate,sdch
Accept-Language:en US,en;q=0.8,sk;q=0.6
Cache-Control:max-age=0
Connection:keep-alive
Cookie:buxus_stu_fiit=i7k35od00tllvech3dfe6om664
User-Agent: Mozilla/5.0 (**Macintosh; Intel Mac OS X 10_8_3**) AppleWebKit/537.31 (KHTML, like Gecko) Chrome/26.0.1410.63 Safari/537.31

Hlavička Dopytu

- Popis klienta
  - *User-agent*: Prehliadač a operačný systém
  - *Accept*: Aký druh dokumentu prehliadač preferuje
  - *Accept-language*: Preferovaný jazyk používateľa
  - ...

Hlavička Dopytu

- Popis dopytu
  - *host*: Meno servera, dobré ak je na stroji viacero virtuálnych serverov
  - *Proxy-Connection*: keep-alive, hovorí serveru aby spojenie neuzavrel a čakal na dalšiu komunikáciu
  - *Keep-Alive*: 300 ako dlho bude spojenie aktívne
  - *Referer*: Odkiaľ klient prišiel, ktorá stránka generovala dopyt

Zaslanie informácie na server

- Formulár v HTML

```
<form method="POST"  action="http://localhost/test.php">
    <input type="text" name="text1">
    <input type="hidden" name="text2" value="80">
    <input type="submit" value="OK">
</form>
<form method="GET" action="http://localhost/test.php">
    <input type="text" name="text1">
    <input type="hidden" name="text2" value="80">
    <input type="submit" value="OK">
</form>
```

Dva typy dopytu

- Metóda GET
  - Na získanie (en. Getting) stránky
  - URL vpísané ako adresa v prehliadači
  - Linky v iných dokumentoch
  - Môže zaslať obmedzené množstvo informácií
  - Stránky možno pred-generovať (cache)

Dva typy dopytu

- Metóda POST
  - Na zaslanie informácie (en. Posting)
  - Môže obsahovať veľké množstvo dát
  - Nie je možné použiť cache
  - Neostávajú v histórii
  - Nie je možné vytvoriť odkaz

Metóda GET

- Príklad
  - Dáta sú súčasťou URL (URLEncoded)
  - Nemá obsah

GET /test.php?**text1=This+is+a+test&text2=80** HTTP/1.0 Connection: Keep-Alive
User-Agent: Mozilla/4.77 [en] (X11; U; Linux 2.2.18 i686) Host: localhost:45678
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
Accept-Encoding: gzip
Accept-Language: fr-FR, fr-CH, en, de-DE
Accept-Charset: iso-8859-1,*,utf-8

URL a kódovanie

- URL - Uniform Resource Locator
- Obmedzená znaková sada, nutné dáta zakódovať
  - • + = medzera
  - %xx = Hex(xx)
- Prehliadač automaticky dáta z formulára zakóduje
- Možno nastavit premennú viac krát:
  - ..test.html?chx1=1&chx1=2

URL a kódovanie

- Odkazy v stránke
  - Odkaz môže obsahovať GET dáta
  - Cez URL je možne preniesť ľubovolné parametre v HTML
  - <a href="example.php?name=john">

Metóda POST

- Príklad

```
POST /test.php HTTP 1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.77 [en] (X11; U; Linux 2.2.18 i686)
Host: localhost:45678
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
Accept-Encoding: gzip
Accept-Language: fr-FR, fr-CH, en, de-DE
Accept-Charset: iso-8859-1,*,utf-8
Content-type: application/x-www-form-urlencoded
Content-length: 29

text1=This+is+a+test&text2=80
```

Metóda POST

- Hodnoty formulára sú prenesené v tele dopytu
- Content-type: popisuje druh obsahu
- Content-length: popisuje veľkosť obsahu

Odozva

- Smeruje zo servera ku klientovi
  - Ide o odpoveď na dopyt
- Obsahuje stav
  - Dokument je Ok, Neexistuje (404) ...
- Samotný dokument
  - Obrázok gif/jpeg ..., HTML, čokoľvek
- Hlavička obsahuje meta-dáta
  - Dátum vytvorenia, jazyk, atd.

Odozva

- Formát je podobný dopytu
- Prvý riadok informuje o stave a protokole
  - HTTP/1.1 200 OK
  - HTTP/1.1 404 Not Found
  - Riadok obsahuje i slovný popis stavu
  - Hlavička má rovnaký formát ako dopyt
  - Za hlavičkou nasledujú dáta

Kód odozvy

- Informačné 1xx
- Úspech 2xx
- Presmerovanie 3xx
- Chyba Klienta 4xx
- Chyba Servera 5xx

Typická komunikácia

- Dopyt

GET / HTTP/1.1
host: www.wikipedia.org

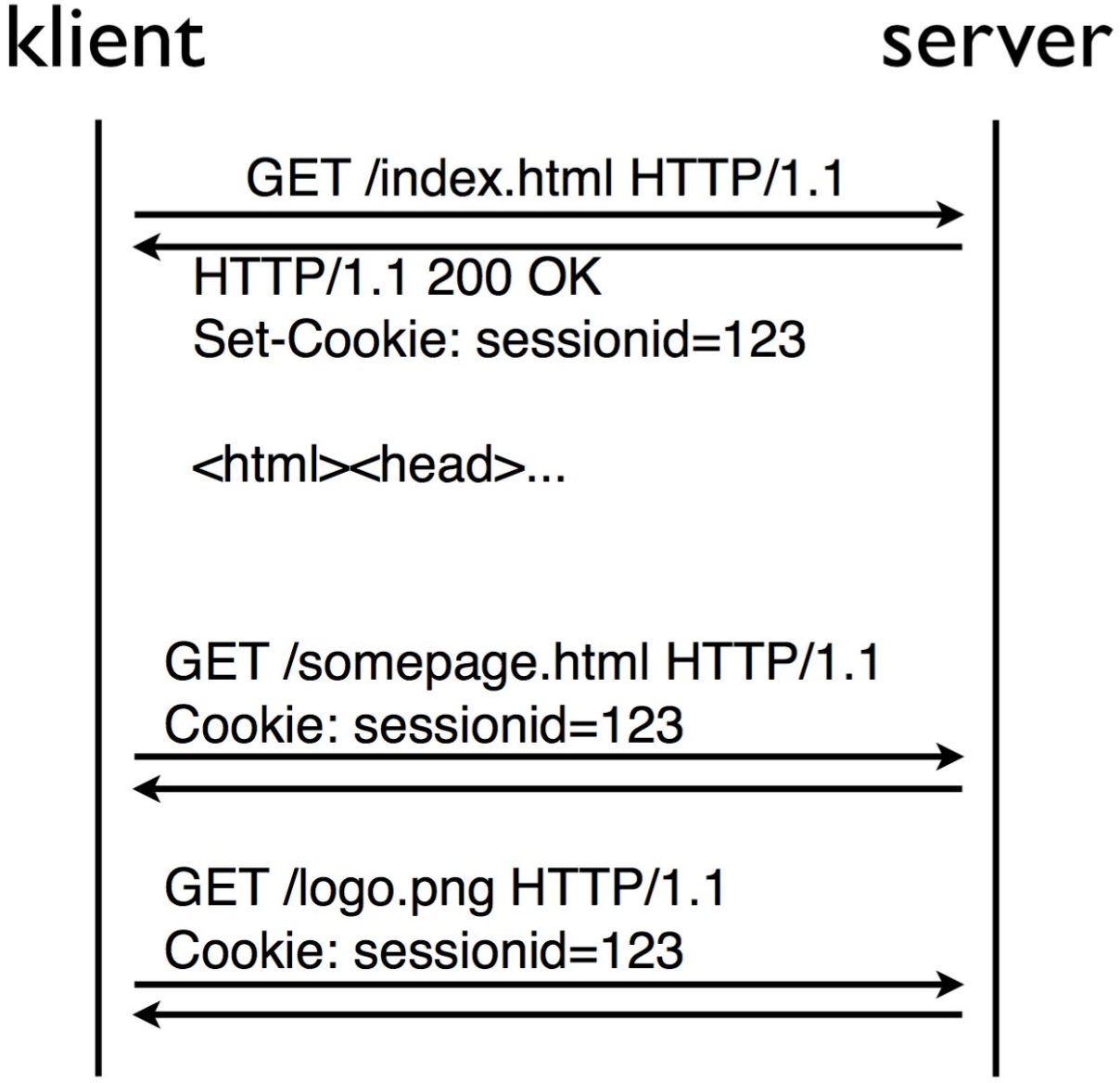
Odozva

HTTP/1.0 200 OK
Date: Wed, 10 Apr 2013 23:56:39 GMT
Server: Apache
X-Content-Type-Options: nosniff
Ing. Peter Kapec, PhD.
Last-Modified: Wed, 10 Apr 2013 22:21:25 GMT Vary:
Accept-Encoding
Content-Length: 48239
Content-Type: text/html; charset=utf-8
Age: 1
Cache-Control: s-maxage=3600, must-revalidate, max-age=0
ZS 2012-13
X-Cache: HIT from sq63.wikimedia.org X-Cache-Lookup: HIT from sq63.wikimedia.org:3128 X-Cache: MISS from sq37.wikimedia.org X-Cache-Lookup: HIT from sq37.wikimedia.org:80
Connection: close
<!DOCTYPE html>
<html lang="mul" dir="ltr">
<head>
<!-- Sysops: Please do not edit the main template directly; update /temp and synchronise. --> <meta charset="utf-8">
<title>Wikipedia</title>

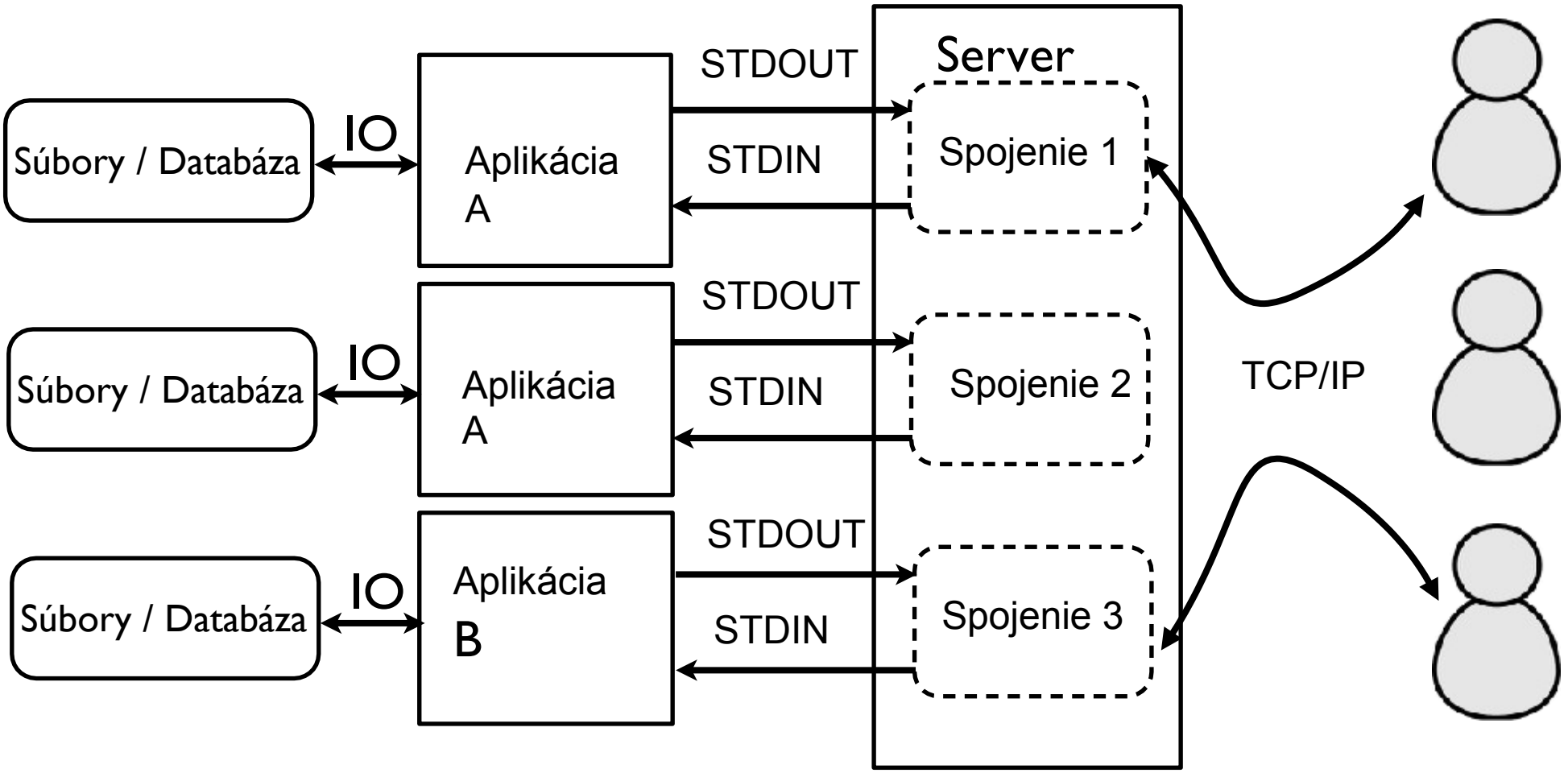
Cookies

- Princíp
  - Úložný priestor pre malé množstvo informácií
  - Zaslané raz so servera klientovi
  - Klient posiela informácie späť v každom dopyte
- Použitie
  - Zväčša už len na uloženie identifikátora spojenia

Cookies


klient
server
GET /index.html HTTP/1.1
HTTP/1.1 200 OK
Set-Cookie: sessionid=123
<html><head>...
GET /somepage.html HTTP/1.1
Cookie: sessionid=123
GET /logo.png HTTP/1.1
Cookie: sessionid=123

Obsluha viacerých klientov



Dynamické web stránky

Dynamické Stránky

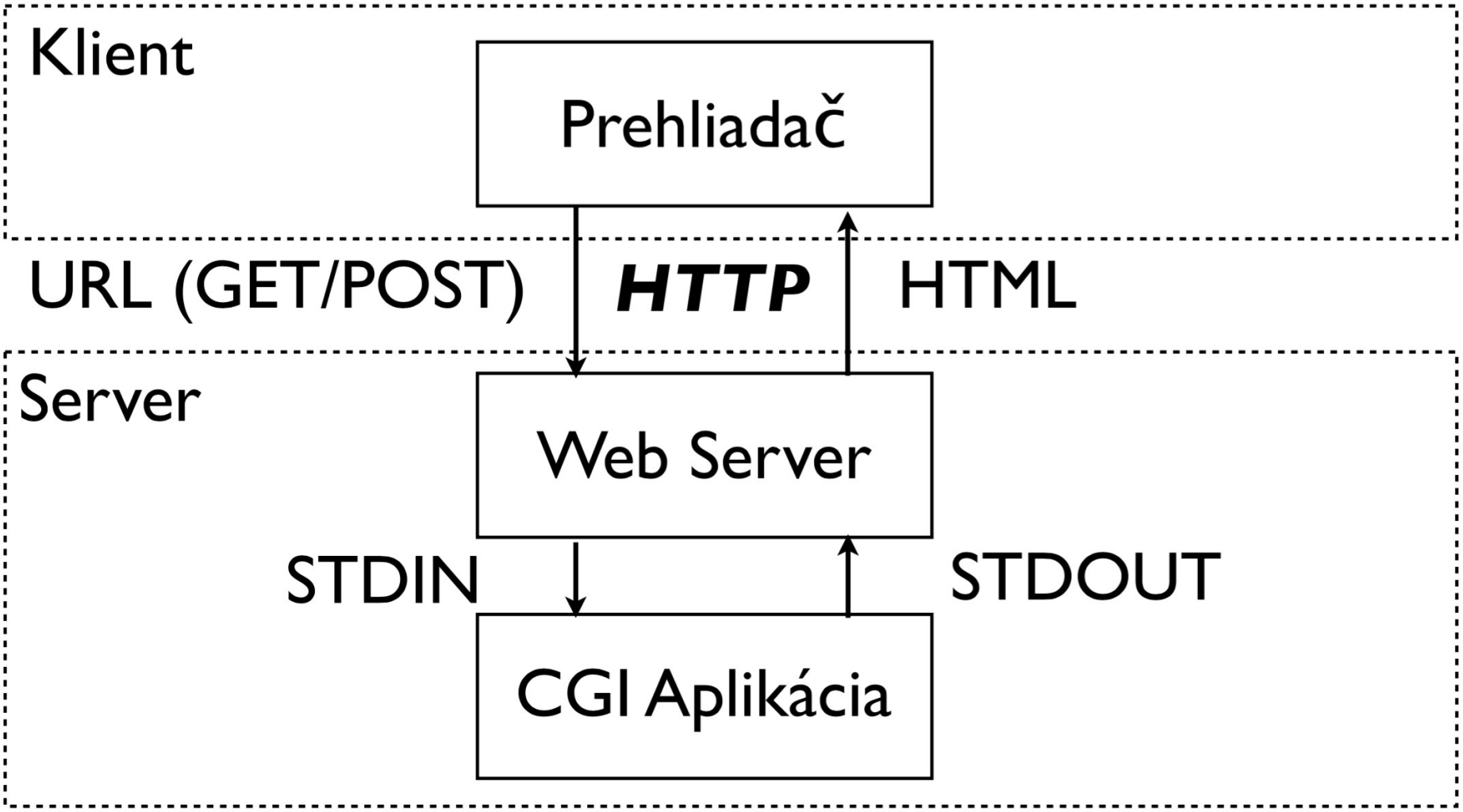
- Interakcia prebieha pomocou HTML dokumentov
- HTML je dynamicky generované pre daného používateľa
- Generované stránky obsahujú odkazy na ďalšie generované dokumenty
- Prehliadač zvyčajne nevykonáva žiadnu logiku, len zobrazuje generované stránky a načítava nové
- Typické napr. LAMP (Linux Apache MySQL PHP)

Web server + CGI

- CGI – Common Gateway Interface
- Jednoduchý spôsob tvorby stránok
- Spustiteľná aplikácia generuje text, ktorý web server posiela ďalej klientom
- Text možno generovať na základe dopytov, ktoré sa prejavia nastavením systémových premenných
- Modernejšia implementácia je FastCGI

CGI aplikácie

-



CGI aplikácia

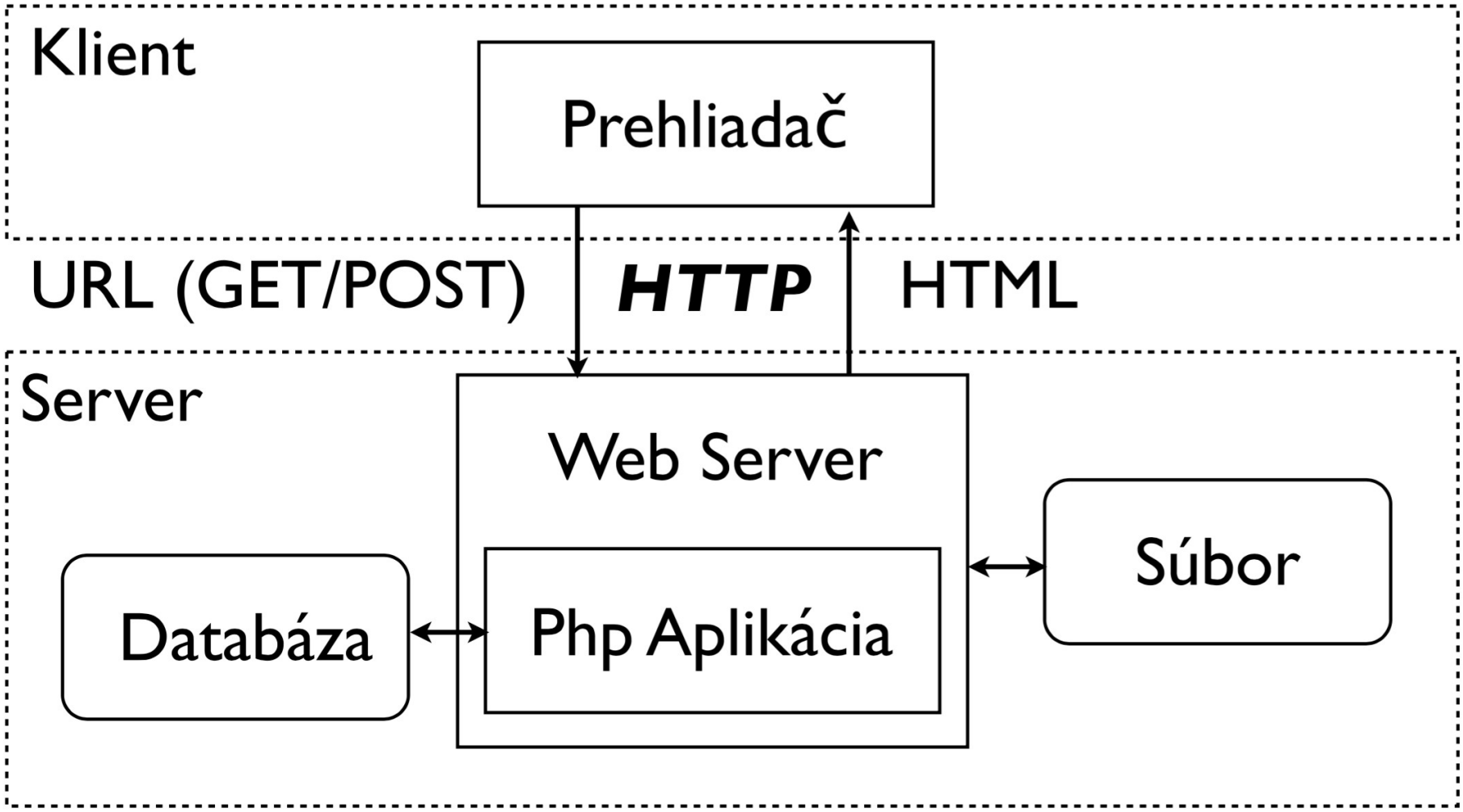
- Na generovanie možno použiť i shell skript

```
1 #!/bin/bash
2
3 echo "Content-type:text/html"
4 echo
5
6 echo '<html> <head> <title> CGI script </title> </head> <body>'
7
8 argument=`echo "$QUERY_STRING" | sed "s|q=||"`
9
10 echo "QUERY_STRING is: <b> $QUERY_STRING </b> <br>"
11 echo "Actual argument is: <b> $argument     </b> <br>"
12
```

CGI aplikácia

- Aplikačný server komunikuje s klientmi cez sieť
- Vstup a výstup aplikácie cez STDIN/STDOUT
- Aplikačný server je typicky HTTP server (Apache)
- Výstup aplikácií je obvykle HTML
- Architektonicky identické s aplikáciami príkazového riadku
- Častokrát sa používajú dynamické jaz. na generáciu obsahu stránok (PHP, Perl, Python, Ruby ...)

PHP aplikácie

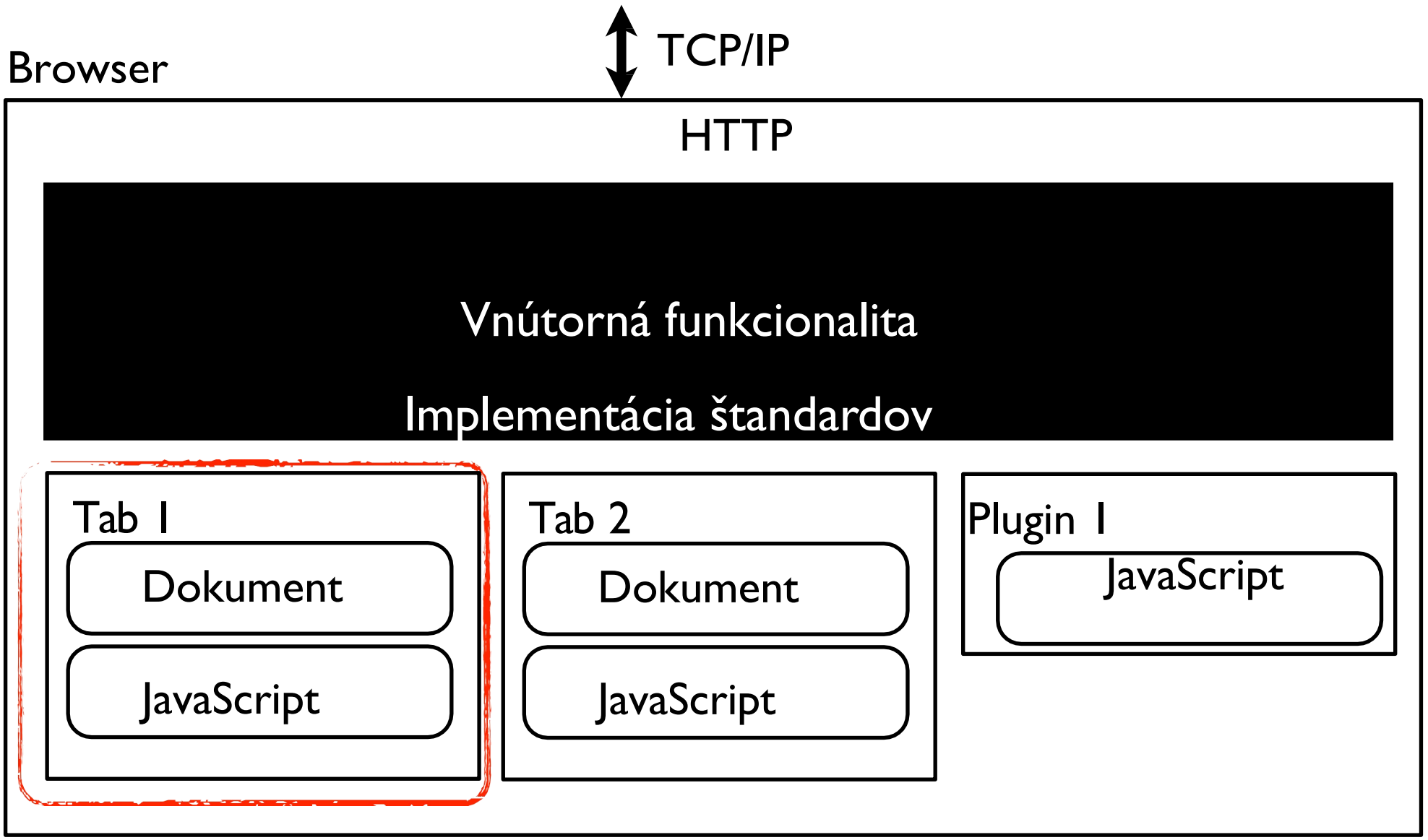


Web aplikácia v prehliadači

Aplikácie v prehliadači

- Namiesto operačného systému aplikácia komunikuje s prehliadačom
- Namiesto zobrazenia a priamej manipulácie s obrazom modifikujeme dokument, ktorý zobrazuje prehliadač
- Rovnako ako v grafických prostrediach sa komunikuje generovaním správ
- Jazyk je zvyčajne JavaScript

Typický prehliadač



Klient-server web aplikácie

Klient-Server Aplikácie

- Situáciu si možno predstaviť ako komunikáciou dvoch aplikácií
- Klientská aplikácia má za úlohu sprostredkovať informáciu zo servera používateľom. napr. JavaScript HTML stránka v prehliadači
- Server aplikácia má za úlohu poskytnúť dáta veľkému množstvu klientov. napr. XML alebo JSON poskytované cez HTTP server

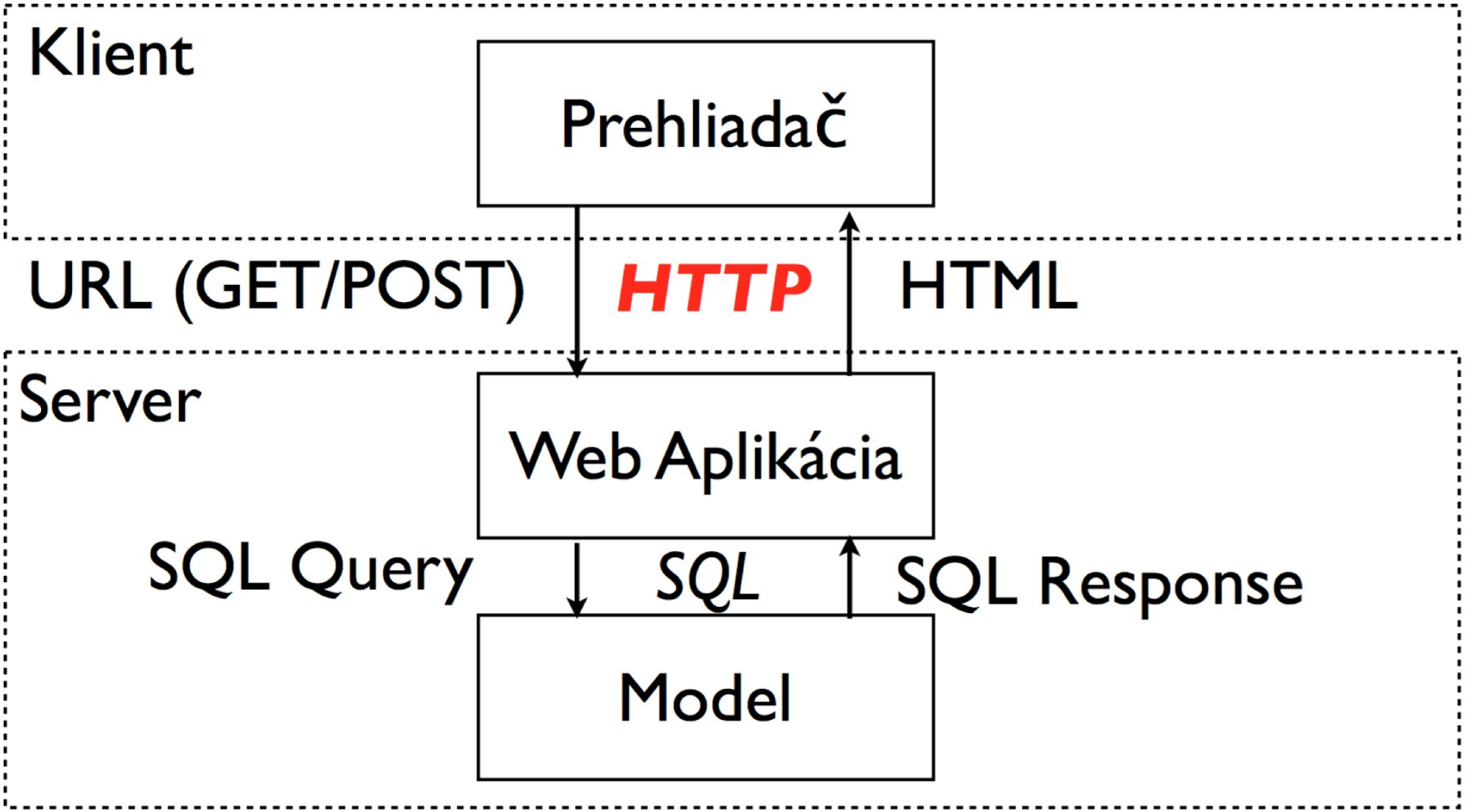
Klient-server aplikácie

- Typická architektúra, kde prehliadač vystupuje ako *View* pre aplikáciu bežiacu na strane servera
- Dáta sa do prehliadača šíria vo forme generovaného HTML, ktorý vytvorí nový *View*
- Stránky generuje *Presenter,* ktorý predstavuje jadro a logiku aplikácie
- Dáta sa ukladajú zväčša do databáz (napr. SQL)

Klient-server aplikácie

- O kontakt s používateľom sa stará prehliadač, na strane klienta nebeží žiadna dodatočná logika
- Používateľské rozhranie možno ľahko vymeniť keďže nie je priamo previazané s dátami aplikácie
- Používateľove akcie majú za následok vygenerovanie nového rozhrania na strane klienta
- Generovanie novej stránky sa realizuje cez **HTTP** protokol cez ktorý prenášame HTML dokumenty

Klient-server aplikácie



Web aplikácie

- Dokumenty, ktoré používateľ vidí, sú generované na strane servera aplikáciou
- Dokument obsahuje URL odkazy, ktoré ovládajú akcie aplikácie
- Aplikácia typicky spracúva informácie z POST a GET dopytov
- Na základe týchto dopytov vytvára ďalšie stránky

Jednoduchý JavaScript HTTP server

- implementovaný pomocou *http* knižnice
- na každú požiadavku odpovie “Hello World”

```
var http = require("http");

function onRequest(request, response) {
  console.log("New Request:");
  response.writeHead(404, {
    "Content-Type": "text/plain"
  });
  response.write("Hello World");
  response.end();
}

http.createServer(onRequest).listen(8888);

console.log("Server started");
```

JavaScript HTTP server

- dynamicky generuje odpovede a stránky podľa toho, akú URL si klient vyžiada
  - implementované spracovaním vo funkcii *onRequest()*
  - *req* - prijatá http požiadavka
  - *res* - odpoveď

```
var http = require('http');

function onRequest(req, res) {
...
}

http.createServer(onRequest).listen(8888);

console.log("Server started");
```

- cez *switch* spracujeme *req.url*
  - */* - vygeneruje HTML stránku s formulárom

```
function onRequest(req, res) {
  // set up some routes
  switch (req.url) {
    case '/':
      // show the user a simple form
      console.log("[200] " + req.method + " to " + req.url);
      res.writeHead(200, "OK", {
        'Content-Type': 'text/html'
      });
      res.write('<html><head><title>Hello Noder!</title></head><body>');
      res.write('<h1>Welcome Noder, who are you?</h1>');
      res.write('<form enctype="application/x-www-form-urlencoded"
                 action="/formhandler" method="POST">');
      res.write('Name: <input type="text" name="username" value="John Doe" /><br />');
      res.write('Age: <input type="text" name="userage" value="99" /><br />');
      res.write('<input type="submit" />');
      res.write('</form></body></html');
      res.end();
      break;
```

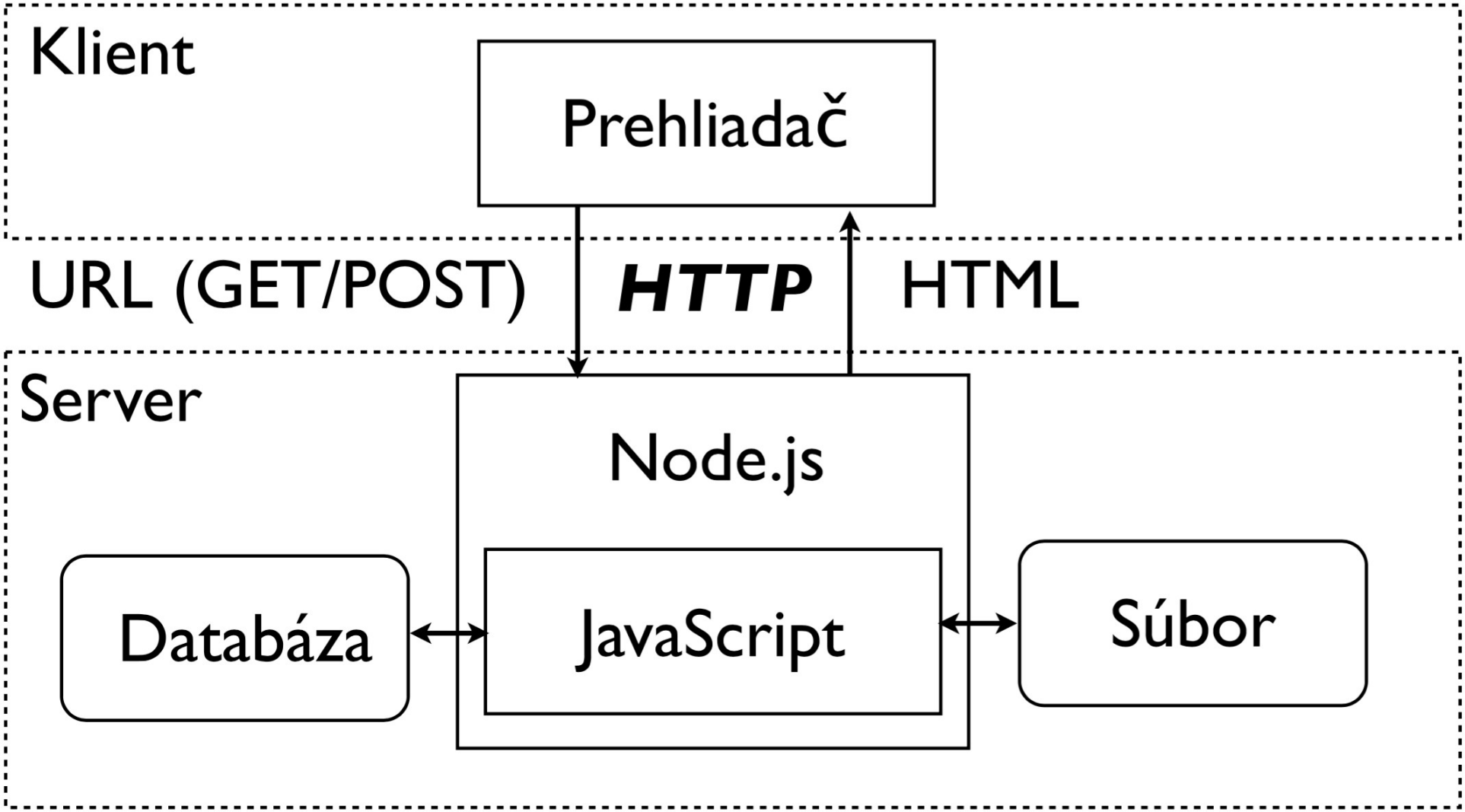
  - */formhandler* - správu typu POST spracuje, pre inú vráti chybu

```
  case '/formhandler':
    if (req.method == 'POST') {
      console.log("[200] " + req.method + " to " + req.url);

      req.on('data', function (chunk) {
        console.log("Received body data:");
        console.log(chunk.toString());
      });

      req.on('end', function () {
        // empty 200 OK response for now
        res.writeHead(200, "OK", {
          'Content-Type': 'text/html'
        });
        res.end();
      });

    } else {
      console.log("[405] " + req.method + " to " + req.url);
      res.writeHead(405, "Method not supported", {
        'Content-Type': 'text/html'
      });
      res.end('<html><head><title>405 - Method not supported</title>
               </head><body><h1>Method not supported.</h1></body></html>');
    }
    break;
```

- pre všetky ostatné URL požiadavky vygeneruje štandardnú 404 chybovú stránku

```
    default:
      res.writeHead(404, "Not found", {
        'Content-Type': 'text/html'
      });
      res.end('<html><head><title>404 - Not found</title>
              </head><body><h1>Not found.</h1></body></html>');
      console.log("[404] " + req.method + " to " + req.url);
  } // switch end
} // function ed
```

Node.js aplikácia



Node.js aplikácia

- JavaScript bežiaci na strane servera
- Node.js priamo obsluhuje spojenia s klientom
- Nie je nutné komunikovať sprostredkovane cez web server
- Ujednotenie jazyka na strane servera a klienta
- Dobrý výkon pri obsluhe veľkého množstva klientov

Zhrnutie

- Klúčové poznatky z prednášky
  - Prehľad a porovnanie riešení pre tvorbu aplikácií v prostredí web-u
  - Komunikácia cez HTTP
  - Ukážka jednoduchého web servera

Nabudúce

- Moderné Web aplikácie
  - AJAX
  - XAML a SOAP
  - JSON a REST
- Ukážka implementácie komunikácie servera s klientom

Ďakujem za pozornosť