INGI2145: CLOUD COMPUTING (Fall 2015)

Web Programming

19 November 2015

Lecture slides adapted from Upenn NETS212 by A. Haeberlen, Z. Ives
Reproduced with permission

© 2015 M. Canini

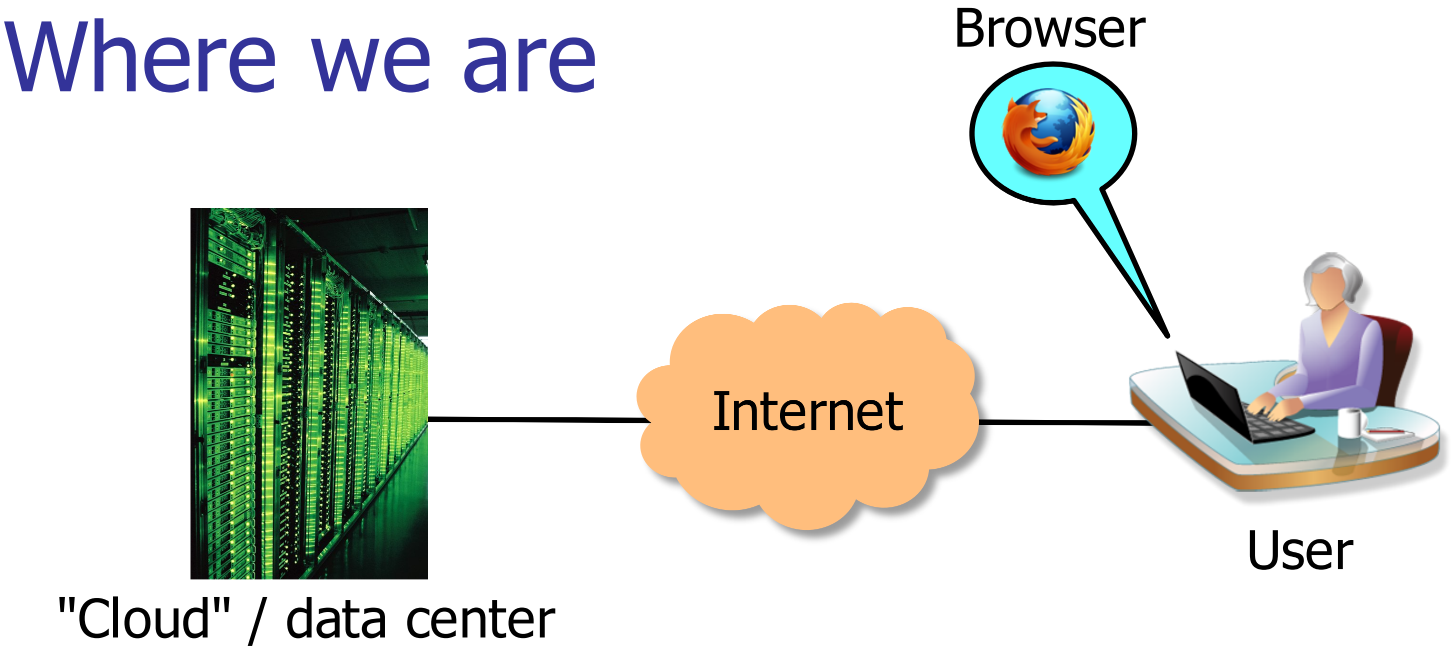
Announcements

- HW2 will soon be available
- It will be due by 17 Dec 2015

- Organized as a problem-oriented assignment
- Will work in groups; each of 4 students

- Goal: solving the challenges to scale a simplified micro-blogging Cloud application
  - Put together many aspects of Cloud computing saw in class

Where we are



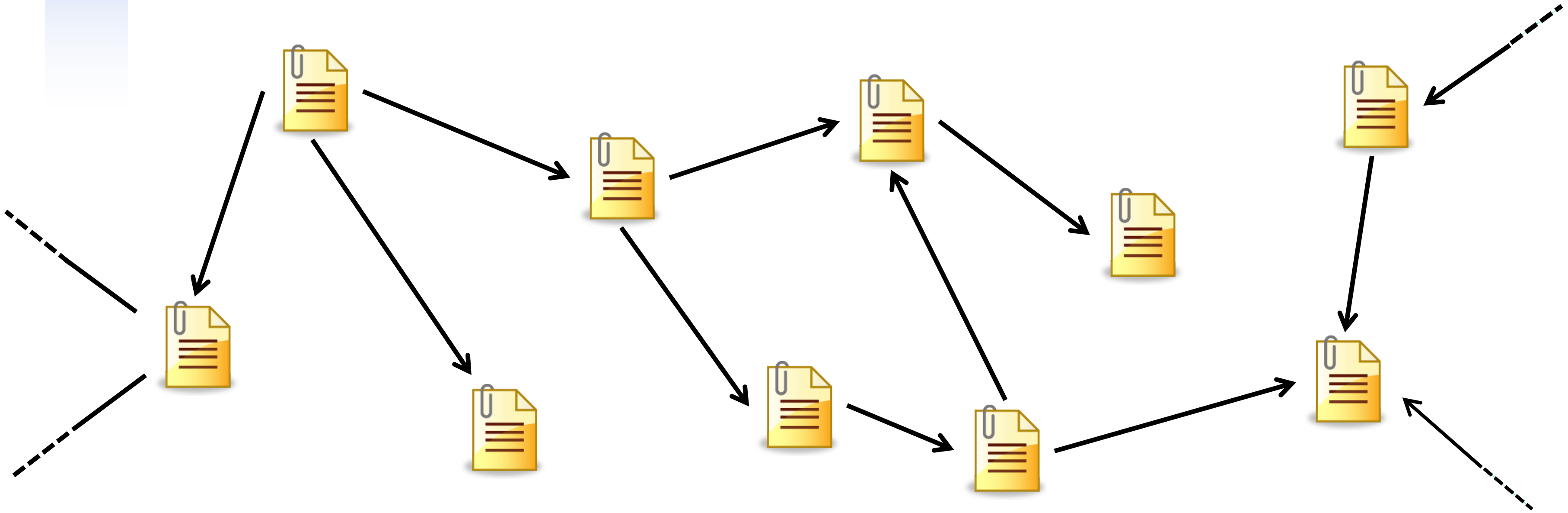
- So far: The 'backend'
  - Large-scale distributed system processes lots of data
  - Economic model, architecture, programming (MapReduce)
- Next: The 'frontend'
  - How to get the data to the users
  - How to build interactive web applications

© 2015 M. Canini

Goals for the next two lectures

- Architecture of the Web
  - Key concepts: Hyperlink, URI/URL, ...
  - Building blocks: HTML, HTTP, DNS, ...
- Servers (esp., web servers)
  - Client/server model
  - State, and where to keep it
  - Architecture: Threads, thread pools, events
- Web applications
  - Dynamic content; maintaining state
- Example use case at SoundCloud

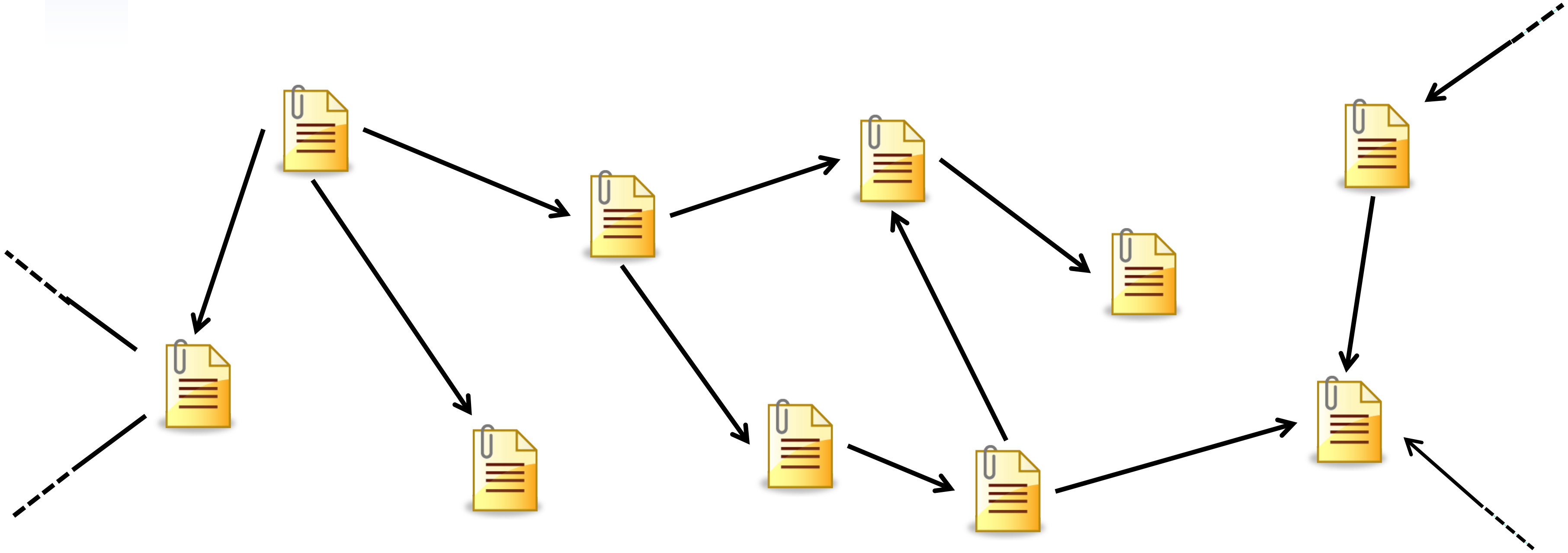
The World Wide Web (WWW)



- A service that runs on the Internet
  - Not identical to the Internet!
- A collection of interconnected documents and other resources
  - Not a hierarchy - any document can reference any other!

© 2015 M. Canini

Making the Web: Ingredients



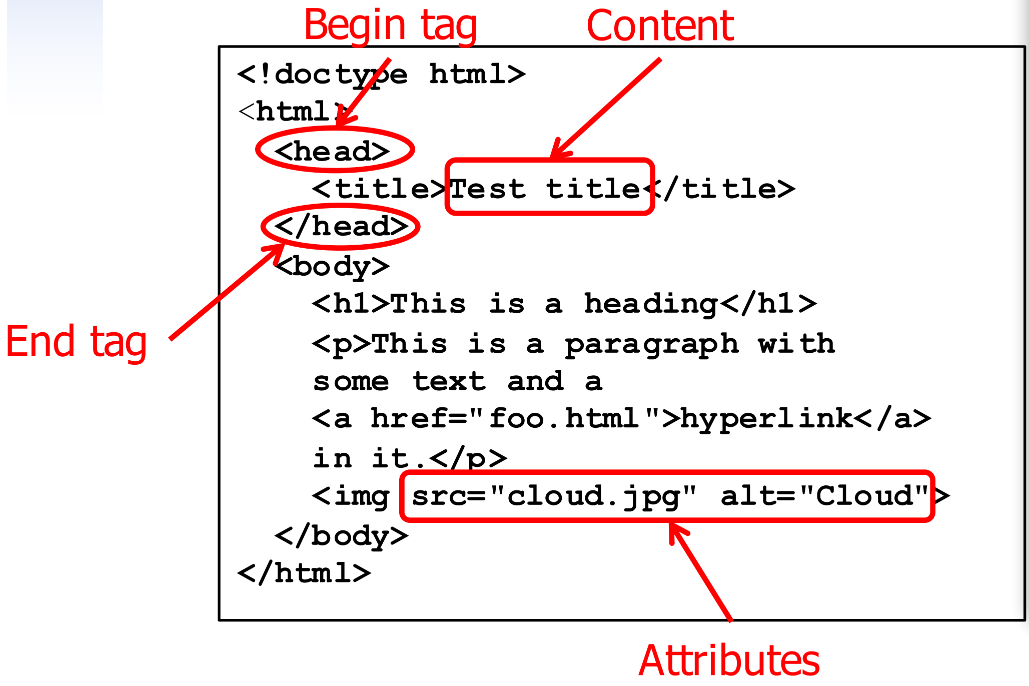
- What do we need to build the Web?

© 2015 M. Canini

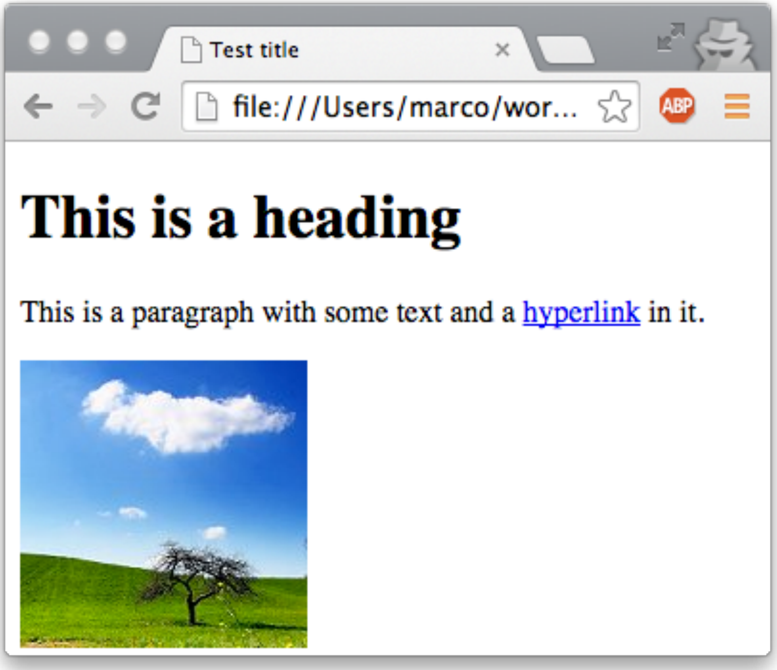
What do we need to make the Web work?

Formats for writing the documents	HTML
A program for displaying documents	Browser
Unique names for the documents	URIs, URLs
A way to find documents	DNS
A system for delivering documents	
Architecture	Client/server model
Efficient implementation	Threads; event-driven prog.
A protocol for transferring documents	HTTP
A way to make content dynamic	
Programming model	Scripting; servlets
Keeping state	Cookies

HTML



```
<!doctype html>
<html>
  <head>
    <title>Test title</title>
  </head>
  <body>
    <h1>This is a heading</h1>
    <p>This is a paragraph with
    some text and a
    <a href="foo.html">hyperlink</a>
    in it.</p>
    <img src="cloud.jpg" alt="Cloud">
  </body>
</html>
```



This is a heading

This is a paragraph with some text and a hyperlink in it.

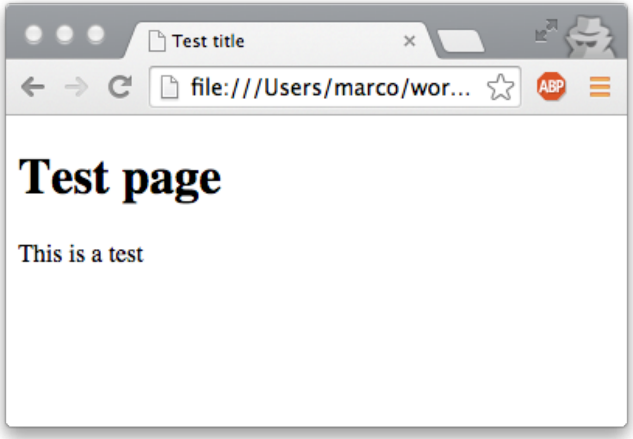
- General structure: Elements, tags, content
  - Most elements have a begin tag and an end tag (denoted by /)
  - Tags may have attributes, e.g., <img> has URL of the image
  - Hierarchy of elements

© 2015 M. Canini

HTML: Presentation and representation

```
<!doctype html>
<html>
  <head>
    <title>Test title</title>
  </head>
  <body>
    <h1>Test page</h1>
    <p>This is a test</p>
  </body>
</html>
```

Representation
(bits and bytes in the document)



Test page

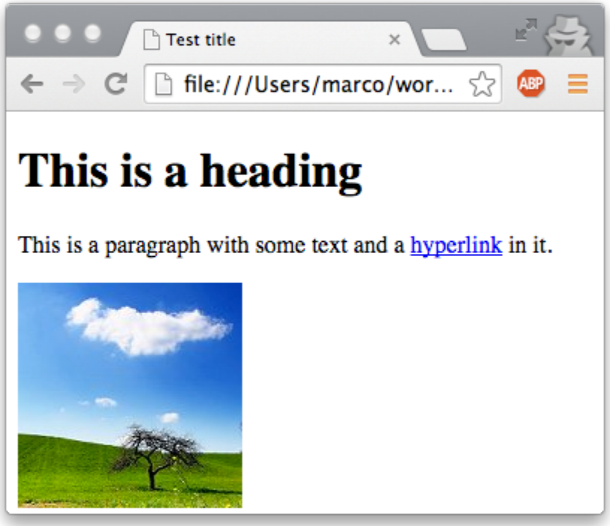
This is a test

Presentation
(document displayed by the browser)

- Original idea: Separate the two
  - Document representation would only describe the structure and the semantic content
  - Browser would take care of the visual layout
- Pros and cons of this approach?
  - Do you think people are following it today?

© 2015 M. Canini

Is basic HTML rendering enough?



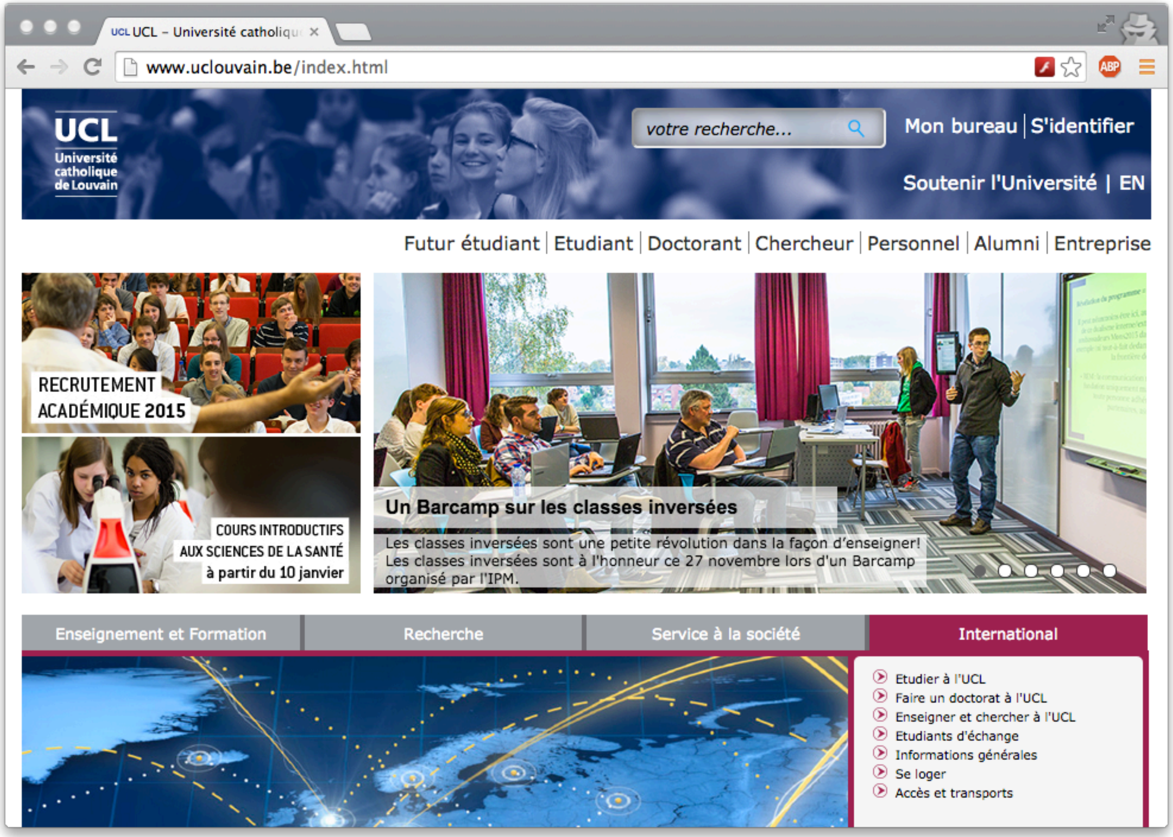
Test title

file:///Users/marco/wor…

This is a heading

This is a paragraph with some text and a hyperlink in it.

vs.



UCL UCL – Université catholiqu

www.uclouvain.be/index.html

UCL Université catholique de Louvain

votre recherche...

Mon bureau | S'identifier

Soutenir l'Université | EN

Futur étudiant | Etudiant | Doctorant | Chercheur | Personnel | Alumni | Entreprise

RECRUTEMENT ACADÉMIQUE 2015

COURS INTRODUCTIFS AUX SCIENCES DE LA SANTÉ à partir du 10 janvier

Un Barcamp sur les classes inversées

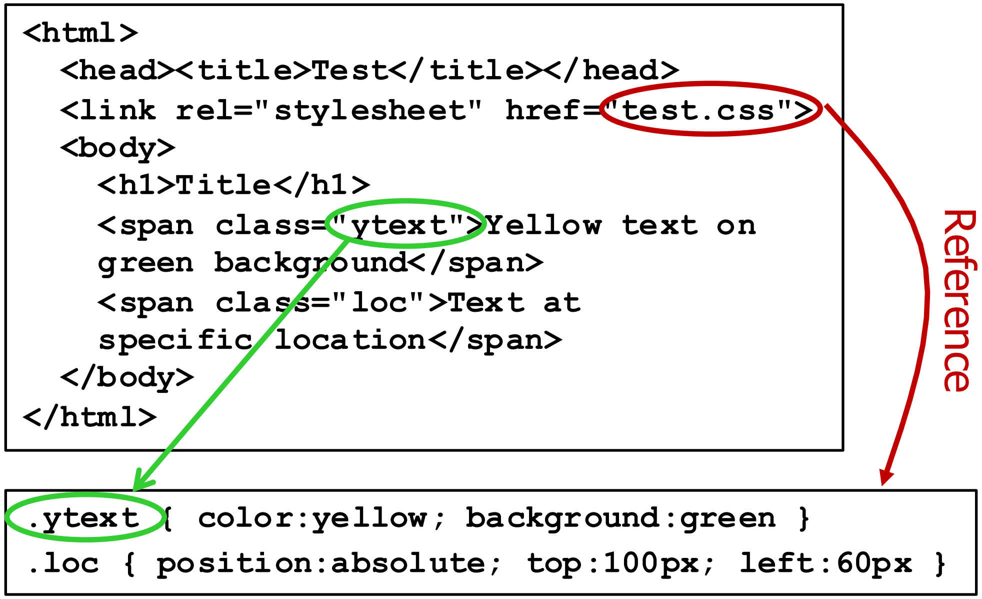
Les classes inversées sont une petite révolution dans la façon d'enseigner! Les classes inversées sont à l'honneur ce 27 novembre lors d'un Barcamp organisé par l'IPM.

Enseignement et Formation | Recherche | Service à la société | International

- Etudier à l'UCL
- Faire un doctorat à l'UCL
- Enseigner et chercher à l'UCL
- Etudiants d'échange
- Informations générales
- Se loger
- Accès et transports

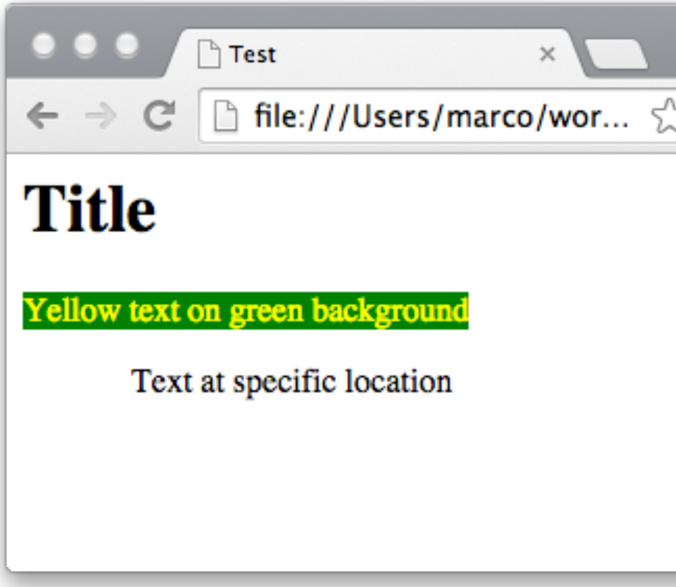
© 2015 M. Canini

Cascading Style Sheets



```
<html>
  <head><title>Test</title></head>
  <link rel="stylesheet" href="test.css">
  <body>
    <h1>Title</h1>
    <span class="ytext">Yellow text on
    green background</span>
    <span class="loc">Text at
    specific location</span>
  </body>
</html>
```

```
.ytext { color:yellow; background:green }
.loc { position:absolute; top:100px; left:60px }
```



Title

Yellow text on green background

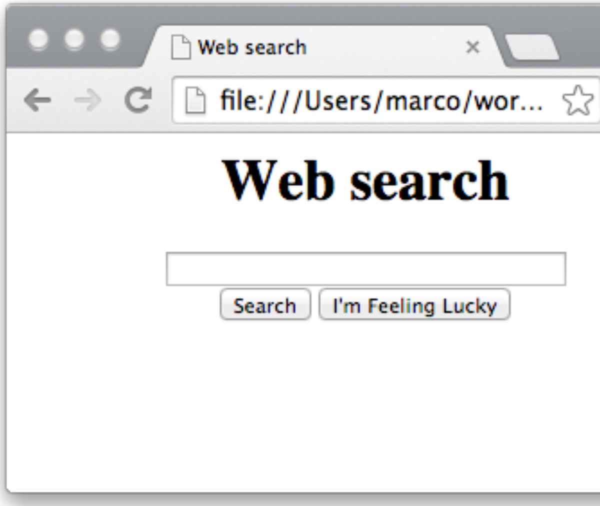
Text at specific location

- Idea: Separate content and formatting again
  - Formatting instructions are kept in a separate file that is linked from the document
  - Document is annotated with references to the formatting instructions, via class="xxx" attribute or special elements (`<span>...</span>`, `<div>...</div>`)

© 2015 M. Canini

Forms

```
<html>
  <head><title>Web search</title></head>
  <body>
    <center><h1>Web search</h1>
    <form action="search.html" method="post">
      <input type="text" size="40" name="term"><br>
      <input type="submit" value="Search">
      <input type="button" value="I'm Feeling Lucky">
    </form></center>
  </body>
</html>
```



- What if we want the user to input some data?
  - `<form>` element creates and input form in the document
  - Several input types available: Single line of text, multiline text, radio buttons, checkboxes, buttons, dropdown boxes...
  - Data is sent over the network once the form is submitted
  - One way to create interactive 'web applications'; more about this later

© 2015 M. Canini

What do we need to make the Web work?

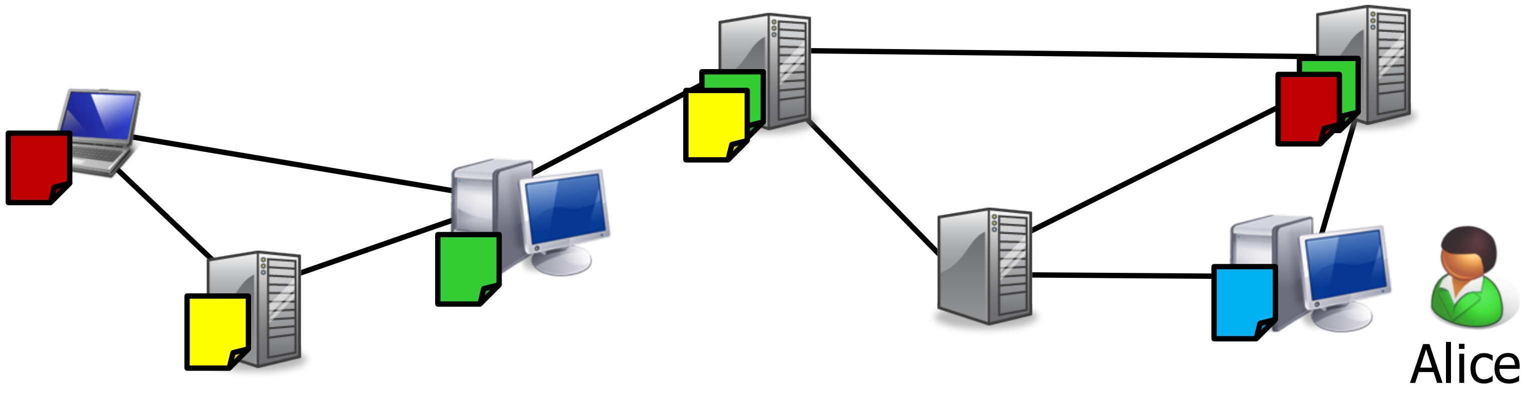
- Formats for writing the documents HTML
- A program for displaying documents Browser
- Unique names for the documents
- A way to find documents
- A system for delivering documents
  - Architecture
  - Efficient implementation

NEXT

- A protocol for transferring documents
- A way to make content dynamic
  - Programming model
  - Keeping state

© 2015 M. Canini

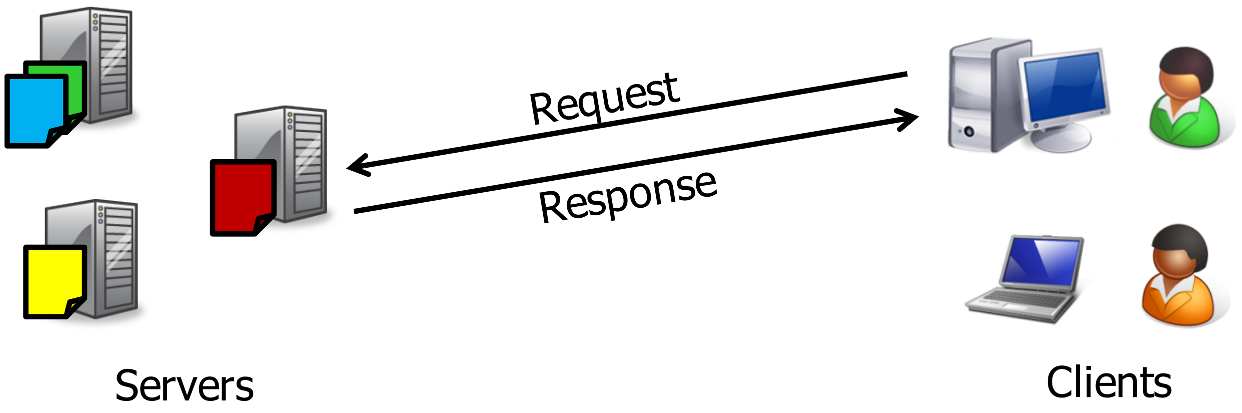
The peer-to-peer model



- How the Web could work (but doesn't):
  - Each machine locally stores some documents
  - If a machine needs a new document, it asks some other machines until it finds one that already has the document
  - No machine is special - they are all 'peers'
- Pros and cons of this approach?

© 2015 M. Canini

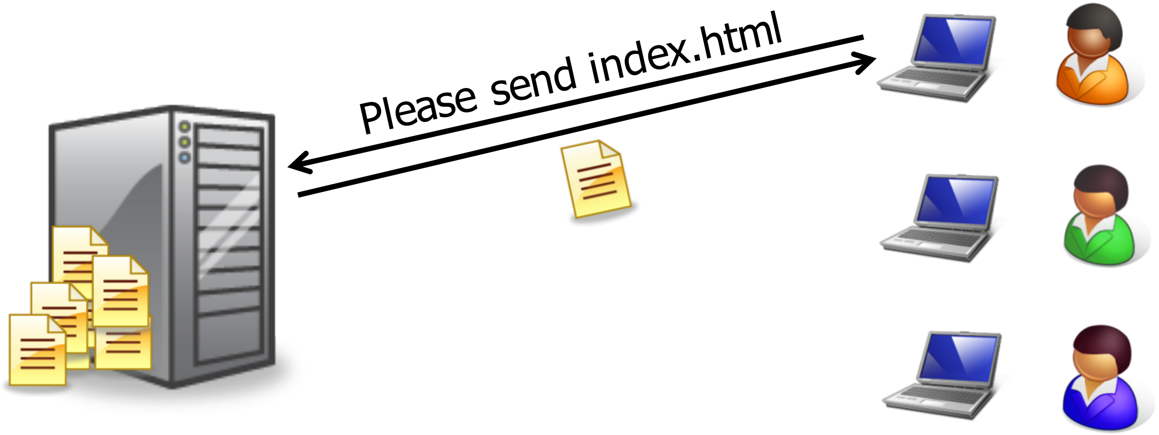
The client-server model



- How the Web actually works today:
  - Some machines (servers) offer documents
  - Other machines (clients) use documents
  - Clients can request documents from servers
  - Model is used for many other services, not just for the web
- Pros and cons of this approach?

© 2015 M. Canini

Servers



- Server: A machine that offers services to other machines
  - Examples: Mail server, file server, chat server, print server, terminal server, web server, name server, game server, ...
- Protocol often uses request/response pattern

© 2015 M. Canini

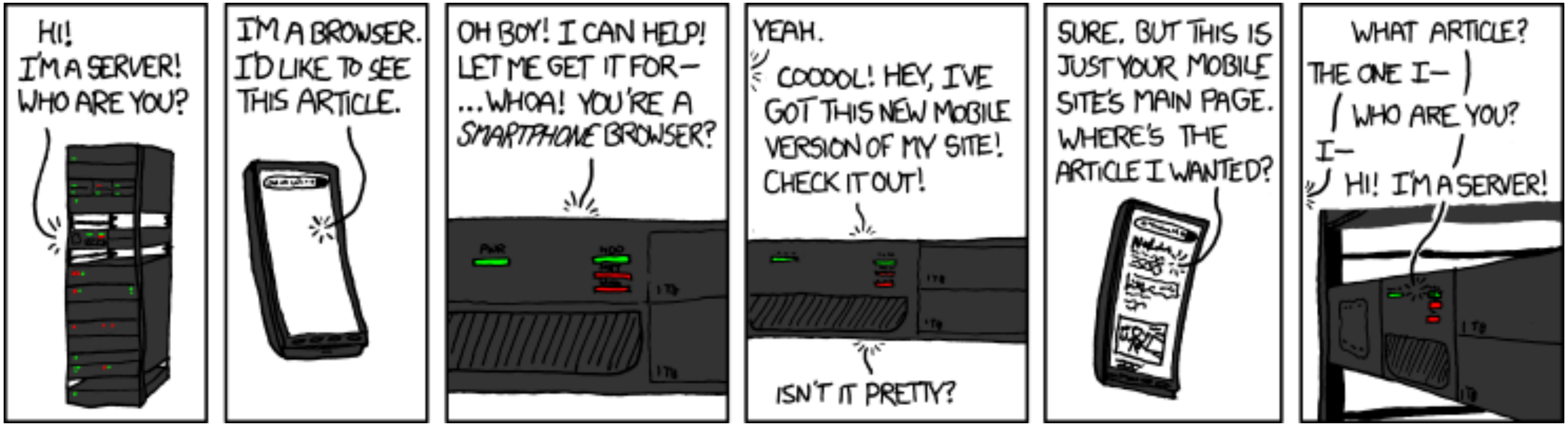
State, and where to keep it

- What if clients make multiple related requests?
  - Example: Open file, read data, read more data, close file
  - Need to remember some state between requests, e.g., which file was opened, or how much data has already been read
  - Who should keep this state: Client, server, or both?
  - If it is kept on the client, how does the server access it?
  - If it is on the server, how does the client reference it?

- If there is no state, or the client keeps all of it, we can build a stateless server
  - Server can forget everything about completed requests
  - Pros and cons of such a design?

© 2015 M. Canini

Server attention span


HI! I'M A SERVER! WHO ARE YOU?
I'M A BROWSER. I'D LIKE TO SEE THIS ARTICLE.
OH BOY! I CAN HELP! LET ME GET IT FOR— ...WHOA! YOU'RE A SMARTPHONE BROWSER?
PWR
HDD
1TB
1TB
YEAH.
COOOOL! HEY, I'VE GOT THIS NEW MOBILE VERSION OF MY SITE! CHECK IT OUT!
ISN'T IT PRETTY?
SURE. BUT THIS IS JUST YOUR MOBILE SITE'S MAIN PAGE. WHERE'S THE ARTICLE I WANTED?
WHAT ARTICLE?
THE ONE I—
WHO ARE YOU?
I—
HI! I'M A SERVER!
1TB
1TB

http://xkcd.com/869/

Recap: Client-server model

- Many possible system architectures
  - Examples: Client/server and peer-to-peer
  - Each has its own set of tradeoffs
  - Web uses client/server, but it could have been otherwise

- Client-server model
  - Functionality implemented by special machines
  - Request/response pattern

- State, and where to keep it
  - Could be on the client, on the server, or on both; pros/cons
  - Stateless servers

© 2015 M. Canini

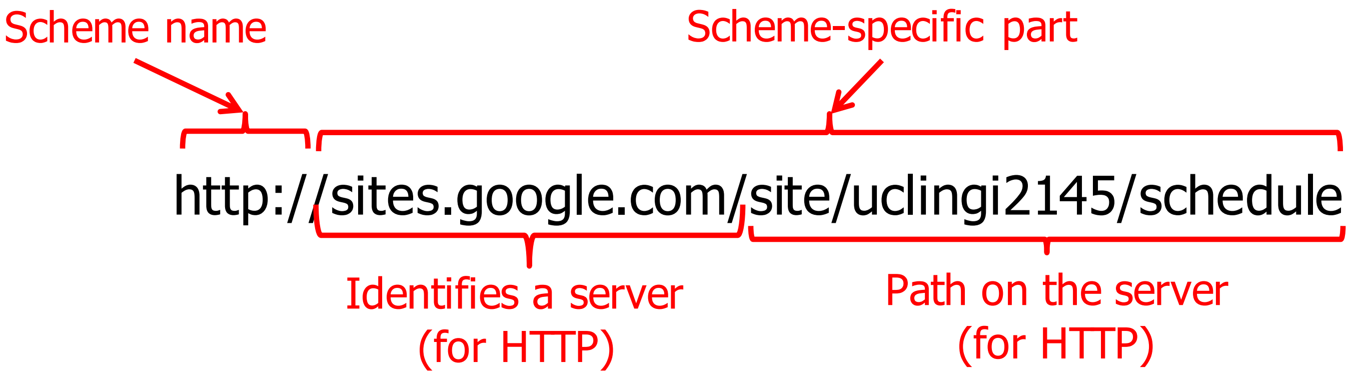
What do we need to make the Web work?

- Formats for writing the documents — HTML
- A program for displaying documents — Browser
- Unique names for the documents — URIs, URLs
- A way to find documents — DNS

NEXT

- A system for delivering documents
  - Architecture — Client/server model
  - Efficient implementation
- A protocol for transferring documents
- A way to make content dynamic
  - Programming model
  - Keeping state

URIs, URNs, and URLs



Scheme name

Scheme-specific part

http://sites.google.com/site/uclingi2145/schedule

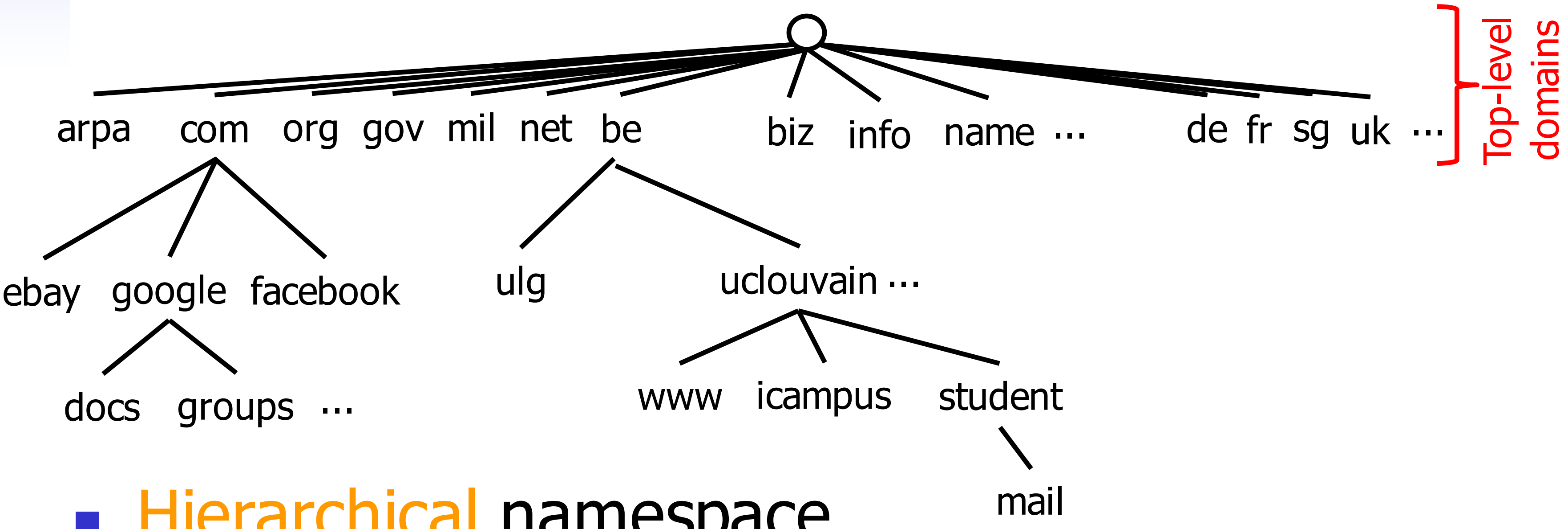
Identifies a server
(for HTTP)

Path on the server
(for HTTP)

- Uniform Resource Identifier (URI)
  - Comes in two forms: URN and URL
- Uniform Resource Name (URN)
  - Specifies *what* to find, independent of its location
  - Example: urn:isbn:1449311520 (for the course textbook)
- Uniform Resource Locator (URL)
  - Specifies *where* to find something
  - <scheme>://[user[:password]@]<server>[:port]/[path][/resource][?param1=value1&param2=value2&...]

© 2015 M. Canini

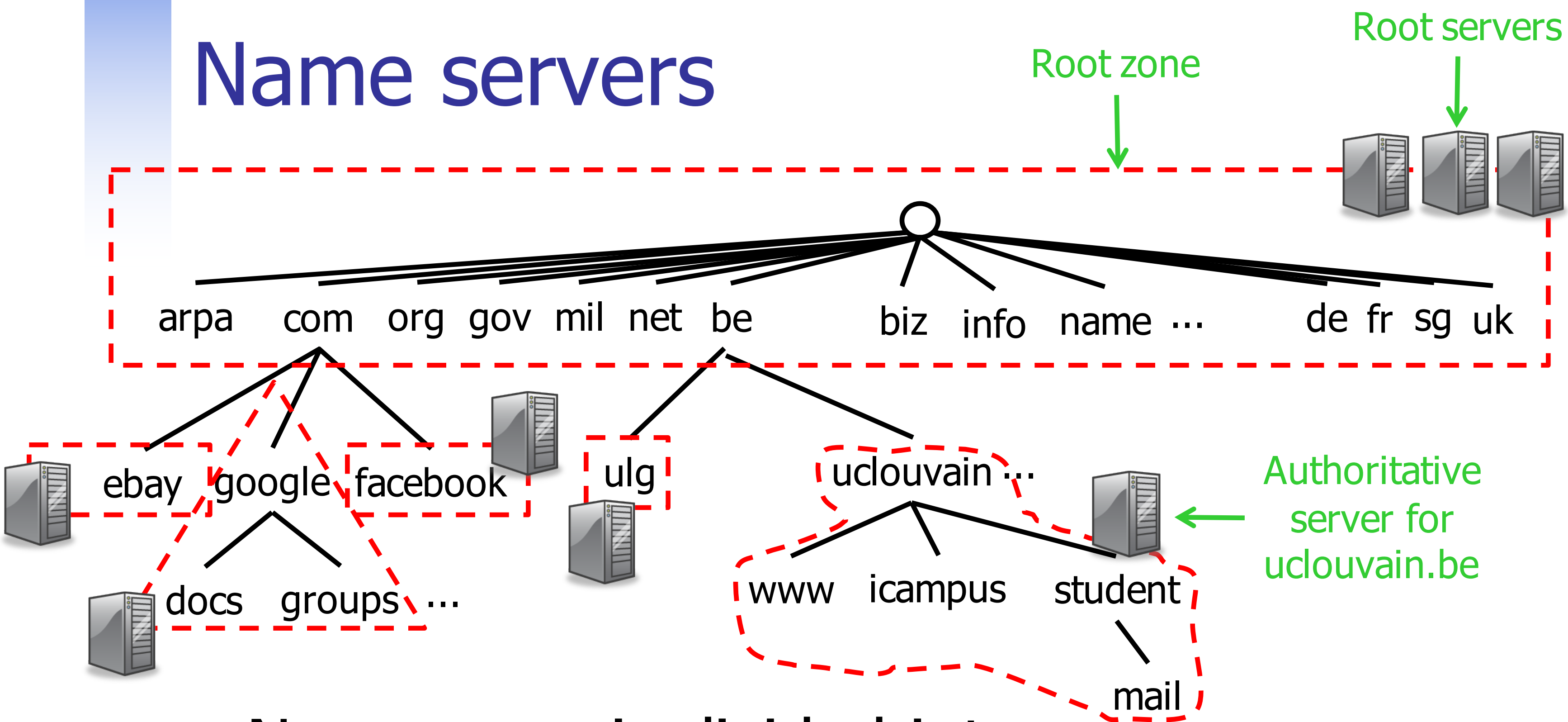
DNS namespace



- Hierarchical namespace
  - First level managed by the Internet Corporation for Assigned Names and Numbers (ICANN)
  - Authority over other levels is delegated
    - Second level generally managed by registrars
    - Further levels managed by organizations or individuals
  - Given a DNS name, how can we find the corresponding host?

© 2015 M. Canini

Name servers



- Namespace is divided into zones
  - TLDs belong to the root zone
- Each zone has an authoritative name server
  - Authoritative server knows, for each name in its zone, which machine corresponds to a given name, or which other name server is responsible

© 2015 M. Canini

What do we need to make the Web work?

- Formats for writing the documents HTML
- A program for displaying documents Browser
- Unique names for the documents URIs, URLs
- A way to find documents DNS
- A system for delivering documents
  - Architecture Client/server model
  - Efficient implementation
- A protocol for transferring documents HTTP
- A way to make content dynamic
  - Programming model
  - Keeping state

NEXT

The HTTP protocol

- How to communicate with a web server?
  - Use the HyperText Transfer Protocol (HTTP)

- A very simple protocol
  - First specified in 1990
  - Runs on top of TCP/IP
  - Default port 80 (unsecure), or 443 (secure, with SSL)
  - Originally stateless (HTTP/1.0), but current version (HTTP/1.1) added support for persistent connections
    - Why?

- Development continues (e.g., SPDY)
  - HTTP/2 specs published in May 2015
  - By Oct 2015, 1.9% of the top 10M websites supports HTTP/2

[W3Techs]

© 2015 M. Canini

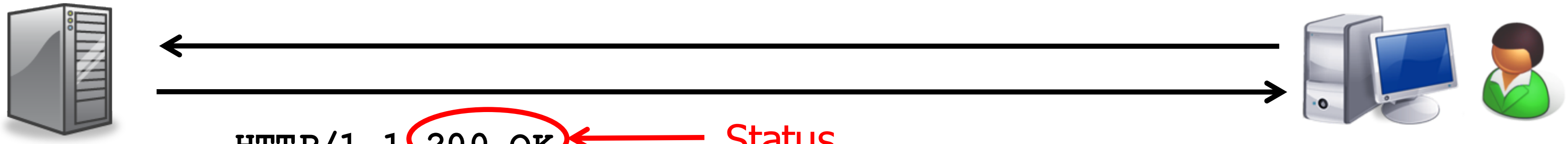
Example: A simple HTTP request

URI

Method

Headers

```
GET /marco.canini/ingi2145/test.html HTTP/1.1
Host: perso.uclouvain.be
Accept: */*
User-Agent: Mozilla/5.0 (...)
If-Modified-Since: Wed, 26 Nov 2014 14:12:37 GMT
```



Status

Headers

```
HTTP/1.1 200 OK
Date: Wed, 26 Nov 2014 14:18:19 GMT
Server: Apache/2.2.3 (CentOS)
Last-Modified: Wed, 26 Nov 2014 14:15:46 GMT
Content-Length: 103
Content-Type: text/html
```

Content (optional)

```
<html><head><title>Test document</title></head>
<body><h3>Test</h3><p>This is a test</p></body>
</html>
```

© 2015 M. Canini

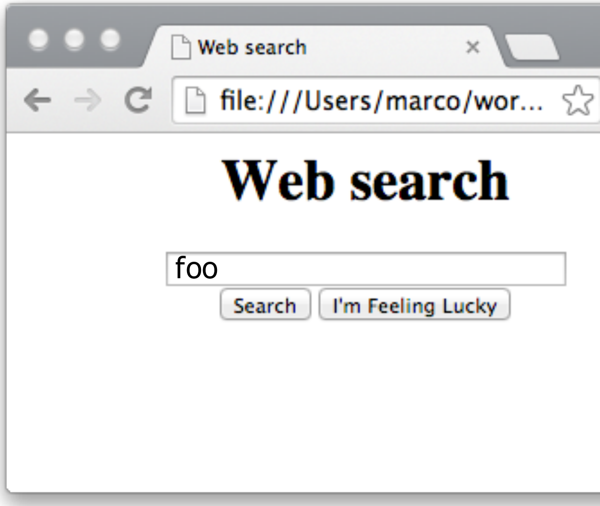
Common HTTP methods

- GET
  - Retrieve whatever information is identified by the URI
- HEAD
  - Like GET, but retrieves only metadata, not the actual object
- PUT
  - Store information under the specified URI
- DELETE
  - Delete the information specified by the URI
- POST
  - Adds new information to whatever is identified by the URI
  - Intended, e.g., for newsgroup posts; today, used mostly to implement dynamic content via forms

© 2015 M. Canini

Forms and GET/POST

```
<html>
  <head><title>Web search</title></head>
  <body>
    <center><h1>Web search</h1>
    <form action="search.html" method="      ">
      <input type="text" size="40" name="term"><br>
      <input type="submit" value="Search">
      <input type="button" value="I'm Feeling Lucky">
    </form></center>
  </body>
</html>
```



Web search

foo

- What happens when we hit 'Search'?

With method="get":

```
GET /search.html?term=foo HTTP/1.1
Accept: text/html
```

With method="post":

```
POST /search.html HTTP/1.1
Accept: text/html

term=foo
```

GET or POST?

- GET should be used for idempotent requests
  - Requests that are safe to re-execute without side-effects, such as making a purchase or committing an edit
  - Browser warns user when resending a POST, but not a GET
- GET has length restrictions
  - Data is put into the URL, whose length is restricted
- GET should only be used with text
  - POST works with arbitrary data (including binaries)

© 2015 M. Canini

Headers

```
GET /index.html HTTP/1.1
Host: www.uclouvain.be
Connection: keep-alive
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,
  image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_5)
  AppleWebKit/537.36 (KHTML, like Gecko) Chrome/38.0.2125.122 Safari/537.36
Referer: https://www.google.com/
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8

HTTP/1.1 200 OK
Date: Wed, 26 Nov 2014 14:28:13 GMT
Server: Apache/2.2.3 (CentOS)
Set-Cookie: JSESSIONID=62CC31...; Path=/cps
Set-Cookie: CPSSESSIONID_ucl=SID-E...; Path=/
Expires: Wed, 26 Nov 2014 14:28:13 GMT
Connection: Keep-Alive
Content-Type: text/html;charset=UTF-8

<html>...
```

- Both the request and the response can contain headers with additional information

Status codes

- Server sends back a status code to report how the request was processed

- Common status codes:
  - 200 OK
  - 301 Moved Permanently
  - 304 Not Modified
  - 401 Unauthorized
  - 403 Forbidden
  - 404 Not Found
  - 500 Internal Server Error

Recap: HTTP

- HTTP - a simple, stateless protocol
  - Server does not (need to) remember past requests

- Request and response contain headers
  - Used to exchange additional information, e.g., to request content in specific formats, or to exchange metadata

- Common HTTP methods:
  - GET - Retrieve a specific document
  - HEAD - Get metadata for a specific document
  - POST - 'Add' information to a document (used for forms)

© 2015 M. Canini

What do we need to make the Web work?

- Formats for writing the documents HTML
- A program for displaying documents Browser
- Unique names for the documents URIs, URLs
- A way to find documents DNS
- A system for delivering documents
  - Architecture Client/server model
  - Efficient implementation NEXT Threads; event-driven prog.
- A protocol for transferring documents HTTP
- A way to make content dynamic
  - Programming model
  - Keeping state

A simple web server

```
socket = listen(port 80);
while (true) {
  connection = accept(socket);
  request = connection.read();
  if (request == "GET <document>") {
    data = filesystem.read(document);
    connection.write("HTTP/1.1 200 OK");
    connection.write(data);
  } else {
    connection.write("HTTP/1.1 400 Bad request");
   }
  close(connection);
}
```

- Is this a good (web) server? If not, why not?

© 2015 M. Canini

The need for concurrency

- What if the server receives lots of requests?
  - Idea #1: Process them serially
  - Problem: Slow client can block everyone else
  - Idea #2: One server for each request
  - Problem: Wasteful

- Server needs to handle requests concurrently
  - Available resources are multiplexed between requests
- How do we do this?
  - Threads, thread pools
  - Events

© 2015 M. Canini

Refresher: Threads and processes

- Physical machine has some fixed number of processor cores - say, c
  - What if we need more than c threads of execution?
- Idea: Time-share cores
  - A single core works on one thread for a while, then context-switch to another one
  - Switching can be done cooperatively: Each thread yields the core to another thread when it has nothing to do
  - Switching can also be preemptive: Each thread gets to run for a fixed amount of time (quantum, typ. 10-100ms)
  - Pros and cons of preemption?
  - Difference between a thread and a process?

A simple thread-based web server

```
worker(connection) {
  request = connection.read();
  if (request == "GET <document>") {
    data = filesystem.read(document);
    connection.write("HTTP/1.1 200 OK");
    connection.write(data);
  } else {
    connection.write("HTTP/1.1 400 Bad req");
  }
  close(connection);
}
main() {
  socket = listen(port 80);
  while (true) {
    connection = accept(socket);
    (new Thread).run(worker, connection);
  }
}
```

Worker thread (one per connection)

Main loop (accepts new connections and launches new threads)

© 2015 M. Canini

Thread-based servers

- Relevant Java constructs:
  - Worker can be a subclass of Thread + implement run()
  - Worker w = new Worker(); w.start();
  - Alternative: Worker implements Runnable
  - Thread t = new Thread(w); t.start();

- What if the threads share some resources?
  - Potential race conditions + consistency issues
  - Can use synchronization and locking
  - Need to be careful about deadlock, livelock, starvation

© 2015 M. Canini

Thread pools

- Problem: Threads operations are expensive
    - Creating and destroying a thread takes time
    - Context switching between threads takes time
    - Making too many threads can exhaust available resources

- Idea: Keep a thread pool with a fixed number of worker threads
    - When a worker is done handling a request, it is assigned another one
    - When there are more concurrenct requests than workers, requests must wait in a queue until a worker is available
    - When there are few requests, some workers may be idle
    - How many threads should we put into the pool?

© 2015 M. Canini

Event-driven programming

- What benefits did the threads really give us?
  - If we have one core and run 1,000 threads on it, does the work really get done faster than with one thread?
- The server's work is driven by events, e.g.:
  - Incoming connection: Need to set up data structures
  - Request arrives: Need to parse + start reading document
  - Document read: Need to send back to the client
  - Entire document sent: Close connection, clean up structures
- Why not base server architecture on events?
  - All we need is a single thread!

An event-based web server

```
handleNewConnection(e) { startReading(e.connection); }
handleRequestRead(e) {
  if (e.request == "GET <document>") {
    issueFilesystemRead(document);
  } else {
    issueWrite(e.connection, "HTTP/1.1 400 Bad req");
  }
}
/* other handlers go here */

main() {
  EventQueue q;
  while (true) {
    e = q.getNextEvent();
    case e of {
      NewConnection: handleNewConnection(e);
      RequestArrived: handleRequestRead(e);
      FileReadCompleted: handleFileRead(e);
      AllDataWritten: handleDataWritten(e);
    }
  }
}
```

Event handlers (must not block)

Who puts events into the queue?

Dispatch loop (distributes events to handlers)

Continuations

- What if, in response to some event, we must perform a blocking system call?
  - Example: Request arrives; now we need to read the file from disk (blocking read() call) and send it back to the client
  - What would happen if we called read() in the event handler?
  - Solution: Write two event handlers:
    - Handler A parses the request and issues a non-blocking read (using a special system call)
    - Handler B is called when the read completes and sends data to client

- What if handler A has some state that handler B needs to know?
  - Must be saved explicitly in a continuation

Event-driven programming in Node

```
var queryByID = function(ID, callback) {
  dynamo.query({TableName:'students', KeyConditions: {“NOMA” : … ID …}},
    function (err, data) {
      if (err) {
        console.log("Error occurred!");
        callback(null);
      } else if (data.Items.length == u0) {
        console.log("No results!");
        callback(null);
      } else {
        console.log("Got data for student: " + ID);
        console.log("The data is: "+data);
        callback(data);
      }
    }
```

Must always call callback, even if the operation has failed!

Inner callback has access to the state of the outer function!

```
function complete_send(data) {
  if (data == null) {
    resp.write(“An error occurred”);
  } else {
    resp.write(“Student info: “ + render(data));
  }
  resp.end();
}
…
queryByID(someID, complete_send)
```

nt-driven

.g., network I/O), it must take
at function once the blocking

- Where is the continuation?

Pros and cons

- Thread-based server or event-driven one?
  - Event-driven servers typically have better performance
  - Event-driven servers do not need as much synchronization
    - Just a single thread - no concurrency!!! (on a single core)
    - However, may need some flags if events can share resources
  - Thread-based servers are typically easier to write+maintain

- What about scaling?
  - Thread-based can immediately take advantage of all cores
  - Event-driven servers require a bit of extra care

Scaling single-threaded server

- To take advantage of multiple cores:
  - Replicate the same flow of execution using many processes

- To scale out to multiple machines:
  - Deploy multiple instances on many machines
  - Use a reverse proxy to load balance requests across machines

Recap: Web servers

- Need to process requests concurrently
  - Otherwise, extremely difficult to achieve high throughput
- Common server architectures:
  - Single-threaded
  - Multithreaded
  - Thread pools
  - Event-driven
- Event-driven architecture:
  - Harder to program, but very efficient
- Most of this also applies to other kinds of servers, not just to web servers

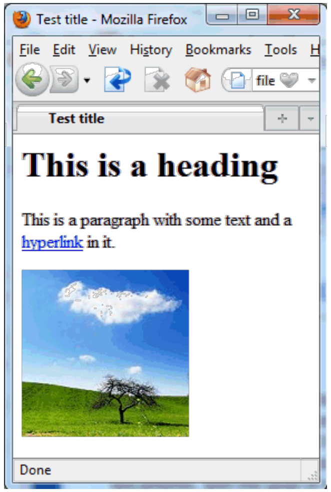
What do we need to make the Web work?

- Formats for writing the documents HTML
- A program for displaying documents Browser
- Unique names for the documents URIs, URLs
- A way to find documents DNS
- A system for delivering documents
  - Architecture Client/server model
  - Efficient implementation Threads; event-driven prog.
- A protocol for transferring documents HTTP
- A way to make content dynamic
  - Programming model
  - Keeping state

NEXT

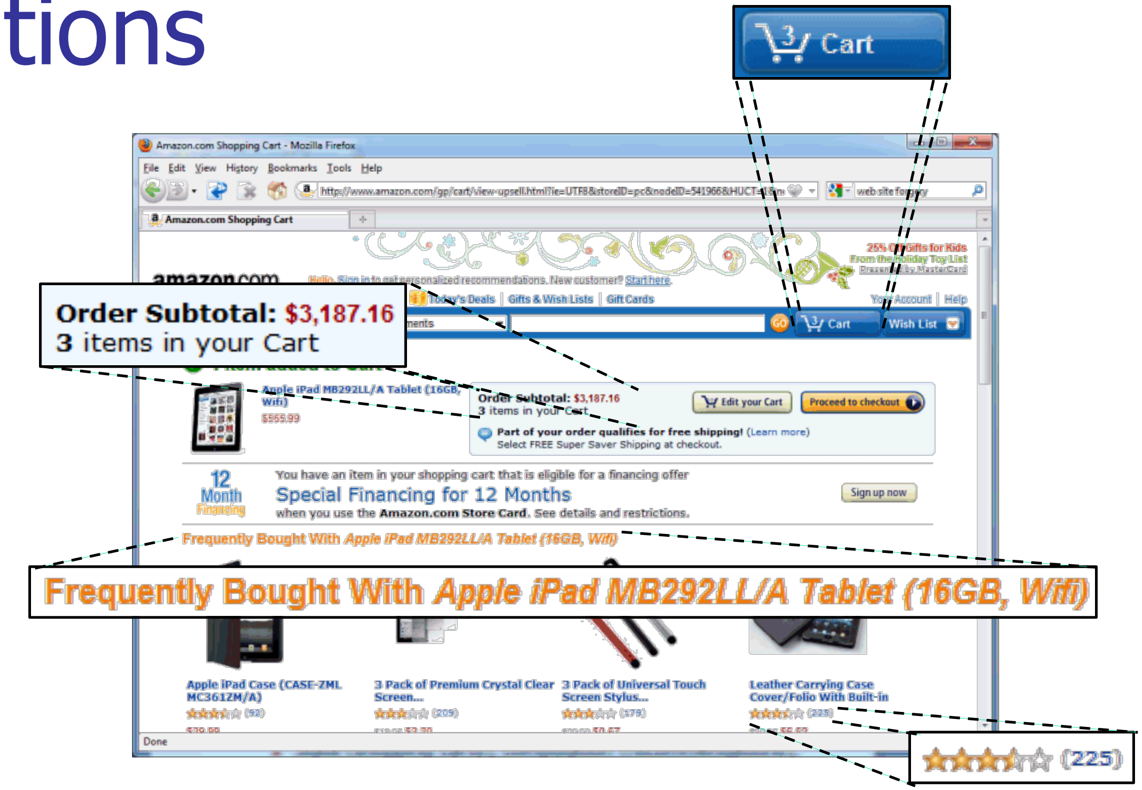
© 2015 M. Canini

Web applications


Test title - Mozilla Firefox
File Edit View History Bookmarks Tools H
file
Test title
Done

This is a heading

This is a paragraph with some text and a hyperlink in it.



Order Subtotal: $3,187.16
3 items in your Cart

Apple iPad MB292LL/A Tablet (16GB, Wifi)
$555.99

Part of your order qualifies for free shipping! (Learn more)
Select FREE Super Saver Shipping at checkout.

12 Month Financing

You have an item in your shopping cart that is eligible for a financing offer
Special Financing for 12 Months
when you use the Amazon.com Store Card. See details and restrictions.

Frequently Bought With Apple iPad MB292LL/A Tablet (16GB, Wifi)

Apple iPad Case (CASE-ZML MC361ZM/A)

3 Pack of Premium Crystal Clear Screen...

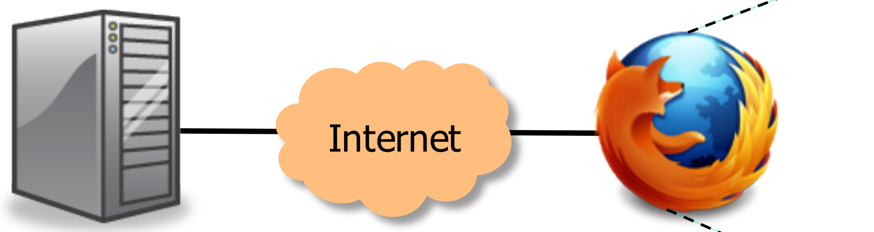
3 Pack of Universal Touch Screen Stylus...

Leather Carrying Case Cover/Folio With Built-in

★★★★☆ (225)

- So far: Writing and delivering static content
- But many web pages today are dynamic
  - State (shopping carts), computation (recommendations), rich I/O (videoconferencing), interactivity, ...

Client-side and server-side



Web server

Client
(web browser)

Amazon.com: Online Shopping for Electronics, Apparel, Computers, Books, DVDs & more - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://www.amazon.com/
amazon.com
Shop All Departments
Search All Departments
Cart
Wish List
Books
Movies, Music & Games
Digital Downloads
Kindle
Computers & Office
Electronics
Home, Garden & Pets
Grocery, Health & Beauty
Toys, Kids & Baby
Clothing, Shoes & Jewelry
Sports & Outdoors
Tools & Home Improvement
Automotive & Industrial
The All-New Kindle
Kindle 3G
$189
Kindle
$139
Order now
FREE Two-Day Shipping with Amazon Prime
IT ONLY TAKES 5,000 POINTS TO GET $50 CASH BACK.
What Other Customers Are Looking At Right Now
wishlist $100,000 Sweepstakes
If You Can Think It You Can Watch It
Find
Match case

User

- Where does the web application run?
  - Can run on the server, on the client, or have parts on both
    - Modern browsers are highly programmable and can run complex applications (example: client-side part of Google's Gmail)
    - Some believe the browser will be 'the new operating system'
  - Client-side technologies: JavaScript, Java applets, Flash, ...
  - Server-side technologies: CGI, PHP, Java servlets, Node.js, …

© 2015 M. Canini

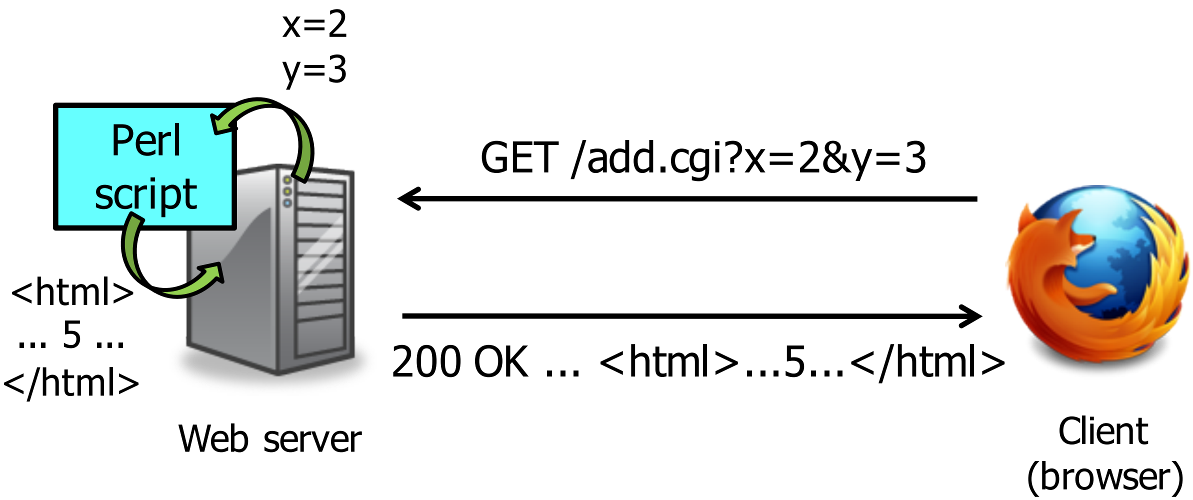
Dynamic content

- Web application technologies NEXT
  - Background: CGI
  - Java Servlets
- Node.js / Express / EJS
- AJAX
- Session management and cookies

Dynamic content

- How can we make content dynamic?
  - Web server needs to return different web pages, depending on how the user interacts with the web application
- Idea #1: Build web app into the web server
  - Why is this not a good idea?
- Idea #2: Loadable modules
  - Is this a good idea?
  - Pros and cons?

CGI



- Common Gateway Interface (CGI)
  - Idea: When dynamic content is requested, the web server runs an external program that produces the web page
  - Program is often written in a scripting language ('CGI script')
  - Perl is among the most popular choices

CGI

- A little more detail:
  1. Server receives HTTP request
     - Example: GET /cgi-bin/shoppingCart.pl?user=ahae&product=iPad
  2. Server decides, based on URL, which program to run
  3. Server prepares information for the program
     - Metadata goes into environment variables, e.g., QUERY_STRING, REMOTE_HOST, REMOTE_USER, SCRIPT_NAME, ...
     - User-submitted data (e.g., in a PUT or POST) goes into stdin
  4. Server launches the program as a separate process
  5. Program produces the web page and writes it to stdout
  6. Server reads the web page and returns it to the client

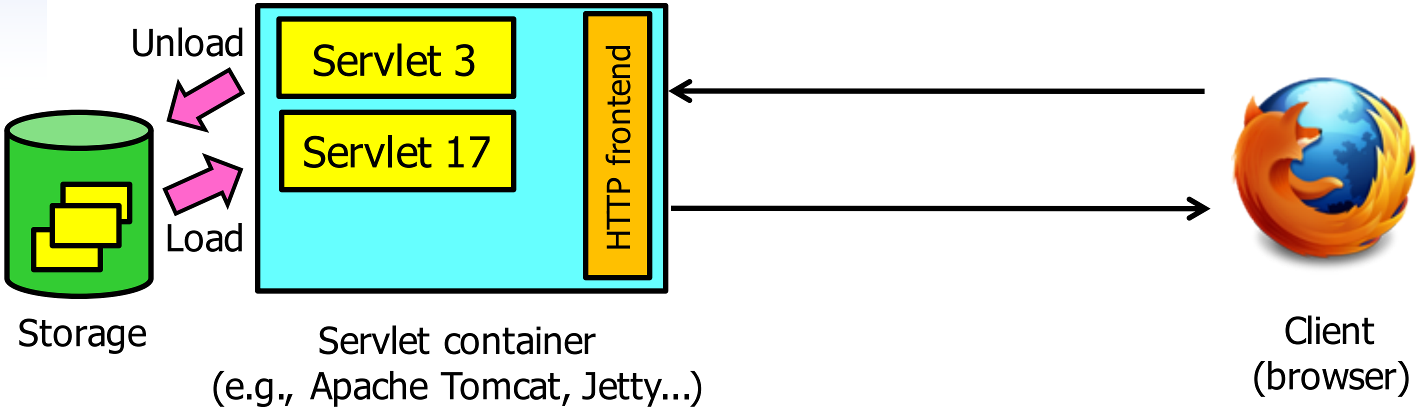
© 2015 M. Canini

Drawbacks of CGI

- Each invocation creates a new process
  - Time-consuming: Process creation can take much longer than the actual work
  - Inefficient: Many copies of the same code in memory
  - Cumbersome: Must store session state in the file system

- CGIs are native programs
  - Security risk: CGIs can do almost anything; difficult to run third-party CGIs; bugs (shell escapes! buffer overflows!)
  - Low portability: A CGI that runs on one web server may not necessarily run on another
  - However, this can also be an advantage (high speed)

What is a servlet?



- Servlet: A Java class that can respond to HTTP requests
  - Implements a specific method that is given the request from the client, and that is expected to produce a response
  - Servlets run in a special web server, the servlet container
    - Only one instance per servlet; each request is its own thread
  - Servlet container loads/unloads servlets, routes requests to servlets, handles interaction with client (HTTP protocol), ...

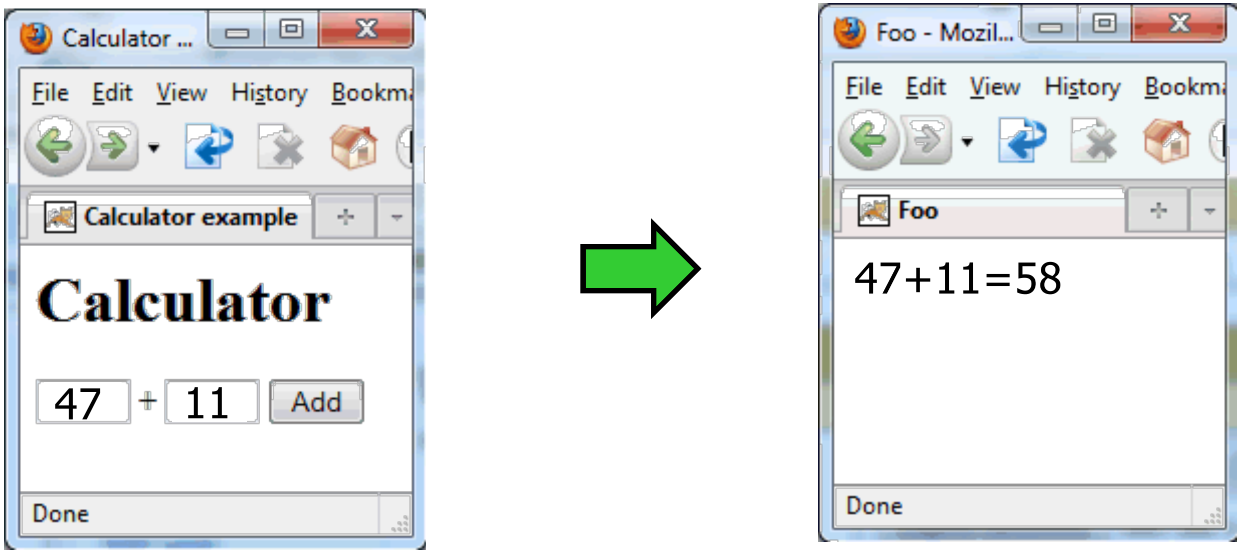
© 2015 M. Canini

Servlets vs CGI

	CGI	Servlets
Requests handled by	Processes (heavyweight)	Threads (lightweight)
Copies of the code in memory	Potentially many	One
Session state stored in	File system	Servlet container (HttpSession)
Security	Problematic	Handled by Java sandbox
Portability	Varies (many CGIs platform-specific)	Java

© 2015 M. Canini

A simple example



Calculator

47 + 11 Add

47+11=58

- Running example: A calculator web-app
  - User enters two integers into a HTML form and submits
    - Result: GET request to calculate?num1=47&num2=11
  - Web app adds them and displays the sum

© 2015 M. Canini

The Calculator servlet

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class CalculatorServlet extends HttpServlet {
  public void doGet(HttpServletRequest request, HttpServletResponse
response)
       throws java.io.IOException {
    int v1 = Integer.valueOf(request.getParameter("num1")).intValue();
    int v2 = Integer.valueOf(request.getParameter("num2")).intValue();
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<html><head><title>Hello</title></head>");
    out.println("<body>"+v1+"+"+v2+"="+(v1+v2)+"</body></html>");
} }
```

Numbers from the GET request become parameters

- Two easy steps to make a servlet:
  - Create a subclass of HttpServlet
  - Overload the doGet() method
    - Read input from HttpServletRequest , write output to HttpServletResponse
    - Do not use instance variables to store session state! (why?)

Dynamic content

- Web application technologies
  - Background: CGI ✓
  - Java Servlets ✓
- Node.js / Express / EJS NEXT
- AJAX
- Session management and cookies

© 2015 M. Canini

What is Node.js?

- A platform for JavaScript-based network apps
  - Based on Google's JavaScript engine from Chrome
  - Comes with a built-in HTTP server library
  - Lots of libraries and tools available; even has its own package manager (npm)
- Event-driven programming model
  - There is a single "thread", which must never block
  - If your program needs to wait for something (e.g., a response from some server you contacted), it must provide a callback function

© 2015 M. Canini

What is JavaScript?

- A widely-used programming language
  - Started out at Netscape in 1995
  - Widely used on the web; supported by every major browser
  - Also used in many other places: PDFs, certain games, ...
  - ... and now even on the server side (Node.js)!
- What is it like?
  - Dynamic typing, duck typing
  - Object-based, but associative arrays instead of 'classes'
  - Prototypes instead of inheritance
  - Supports run-time evaluation via eval()
  - First-class functions

© 2015 M. Canini

What is Express?

- Express is a minimal and flexible framework for writing web applications in Node.js
  - Built-in handling of HTTP requests
  - You can tell it to 'route' requests for certain URLs to a function you specify
    - Example: When /login is requested, call function handleLogin()
  - These functions are given objects that represent the request and the response, not unlike Servlets
  - Supports parameter handling, sessions, cookies, JSON parsing, and many other features
  - API reference: http://expressjs.com/api.html

```
var express = require('express');
var app = express();

app.get('/', function(req, res) {
  res.send('hello world');
});

app.listen(3000);
```

What is Embedded JS (EJS)?

```
app.get('/', function(req, res) {
  res.send('<html><head><title>'+
    'Lookup result</title></head>'+
    '<body><h1>Search result</h1>'+
    req.param('word')+' means '+
    +lookupWord(req.param('word')));
  );
});
```

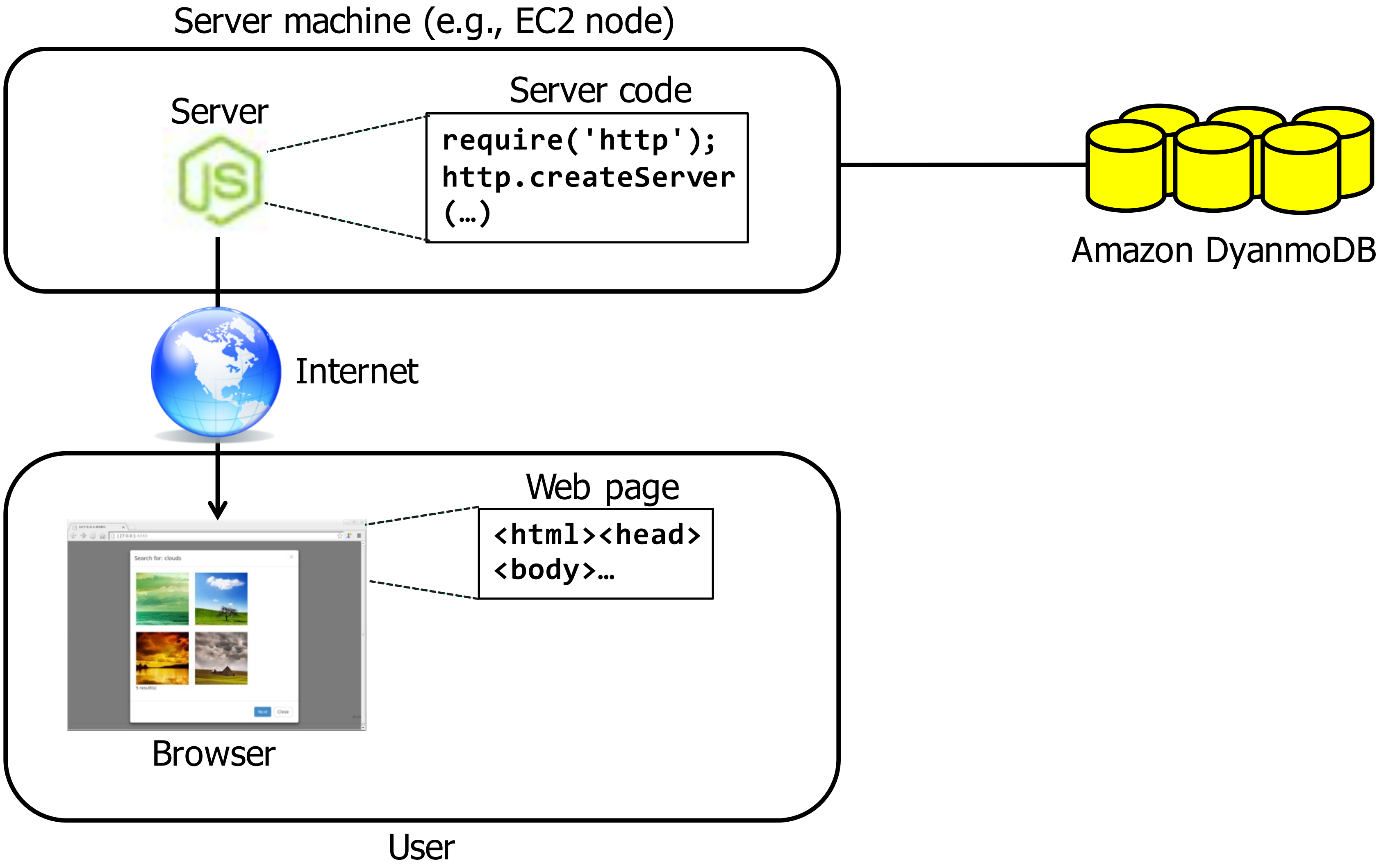
```
...
 w = req.param('word');
 res.render('results.ejs',
  {blank1:w, blank2:lookupWord(w)});
```

```
<html><head><title>Lookup result</title>
</head><body><h1>Search result</h1>
<% =blank1 %> means <% =blank2 %>
```

- We don't want HTML in our code!
- EJS allows you to write 'page templates'
  - You can have 'blanks' in certain places that can be filled in by your program at runtime
  - <% =value %> is replaced by variable 'value' from the array given to render()
  - <% someJavaScriptCode() %> is executed
    - Can do conditionals, loops, etc.

© 2015 M. Canini

How do the pieces fit together?



Server machine (e.g., EC2 node)

Server

Server code

```
require('http');
http.createServer
(…)
```

Amazon DyanmoDB

Internet

Web page

```
<html><head>
<body>…
```

Browser

User

© 2015 M. Canini

Dynamic content

- Web application technologies
  - Background: CGI ✓
  - Java Servlets ✓
- Node.js / Express / EJS ✓
- AJAX ← NEXT
- Session management and cookies

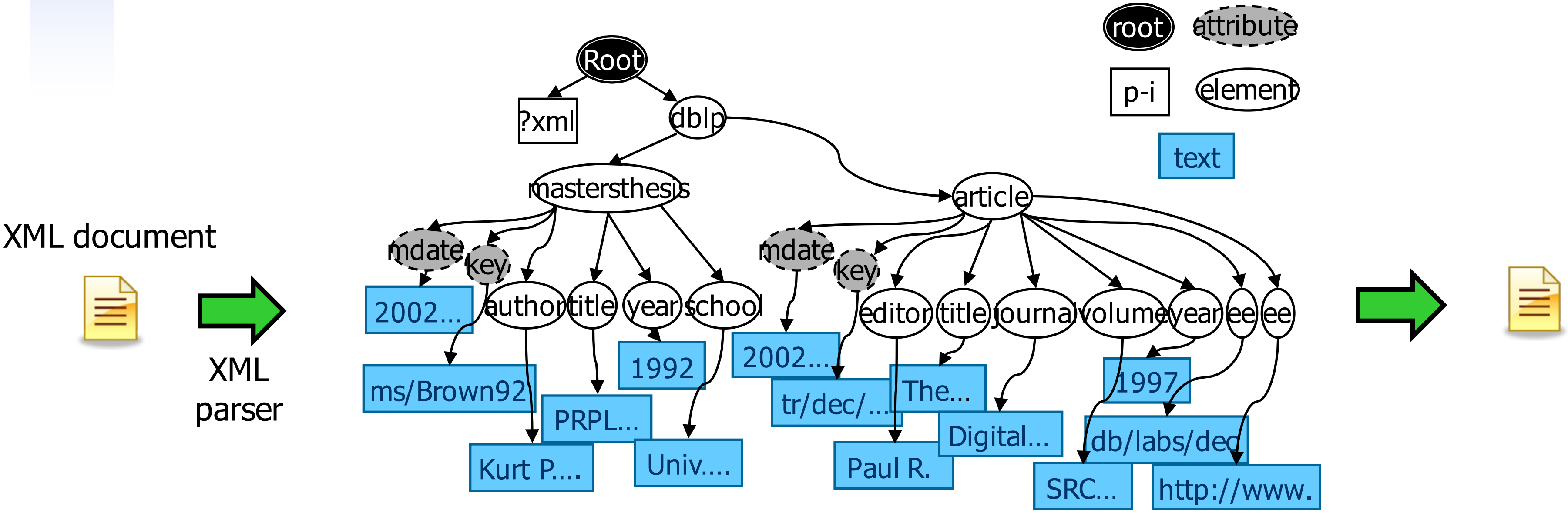
© 2015 M. Canini

What about the client side?

- Javascript was created as a script language for web browsers
  - Runs directly in the browser → can respond to user interactions quickly, without wide-area latencies
  - Can interact with web pages through the DOM
- Developed at Netscape (1995)
- Standardized as ECMAscript (1997)

© 2015 M. Canini

What is the Document Object Model?



- Document components represented by objects
  - Objects have methods like getFirstChild(), getNextSibling()...
    → can be used to traverse the tree
  - Can also modify the tree, and thus alter the document, via insertAfter(), etc.

Functions for accessing the DOM

- The HTML page itself is called 'document'
- To get information from the document:
  - var price = document.getElementById('price').value;
  - var allimages = document.getElementsByName('img');
  - var firstimg = document.getElementsByName('img')[0];
- To put information into the document:
  - Create new elements; replace, or append to, existing nodes

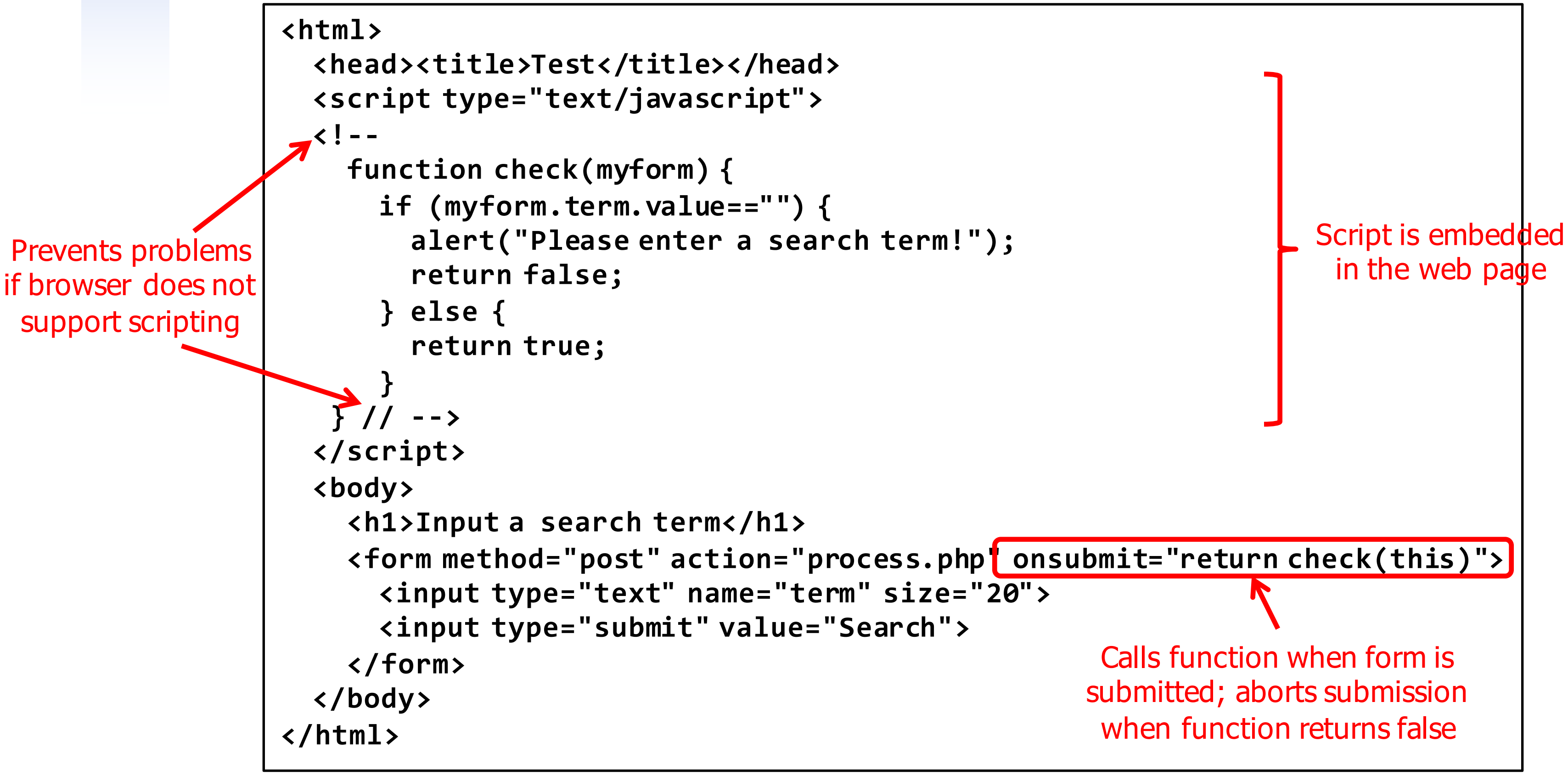
```
// Find thing to be replaced
var mainDiv = document.getElementById("main-page");
var orderForm = document.getElementById("target");
// Create replacement
var paragraph = document.createElement("p");
var text = document.createTextNode("Here is the new text.");
paragraph.appendChild(text);
// Do the replacement
mainDiv.replaceChild(paragraph, target);
```

Event handlers

- Client-side JS can react to various events
  - Examples: User clicks on an element or presses a key, user submits a form, user changes a selection in a form, page finishes loading, mouse moves over a certain element...
  - Web page can request that the browser call a certain JavaScript function when the event occurs

- Events can be requested from the web page:
  - <a href="foo.html" onClick="alert('You\'ve clicked!')">
- ... or directly from JavaScript:
  - theElement.onclick = functionName (DOM 0)
  - theElement.addEventListener(type, function, opt)

A simple client-side example



```
<html>
  <head><title>Test</title></head>
  <script type="text/javascript">
  <!--
    function check(myform) {
      if (myform.term.value=="") {
        alert("Please enter a search term!");
        return false;
      } else {
        return true;
      }
   } // -->
  </script>
  <body>
    <h1>Input a search term</h1>
    <form method="post" action="process.php" onsubmit="return check(this)">
      <input type="text" name="term" size="20">
      <input type="submit" value="Search">
    </form>
  </body>
</html>
```

Prevents problems if browser does not support scripting

Script is embedded in the web page

Calls function when form is submitted; aborts submission when function returns false

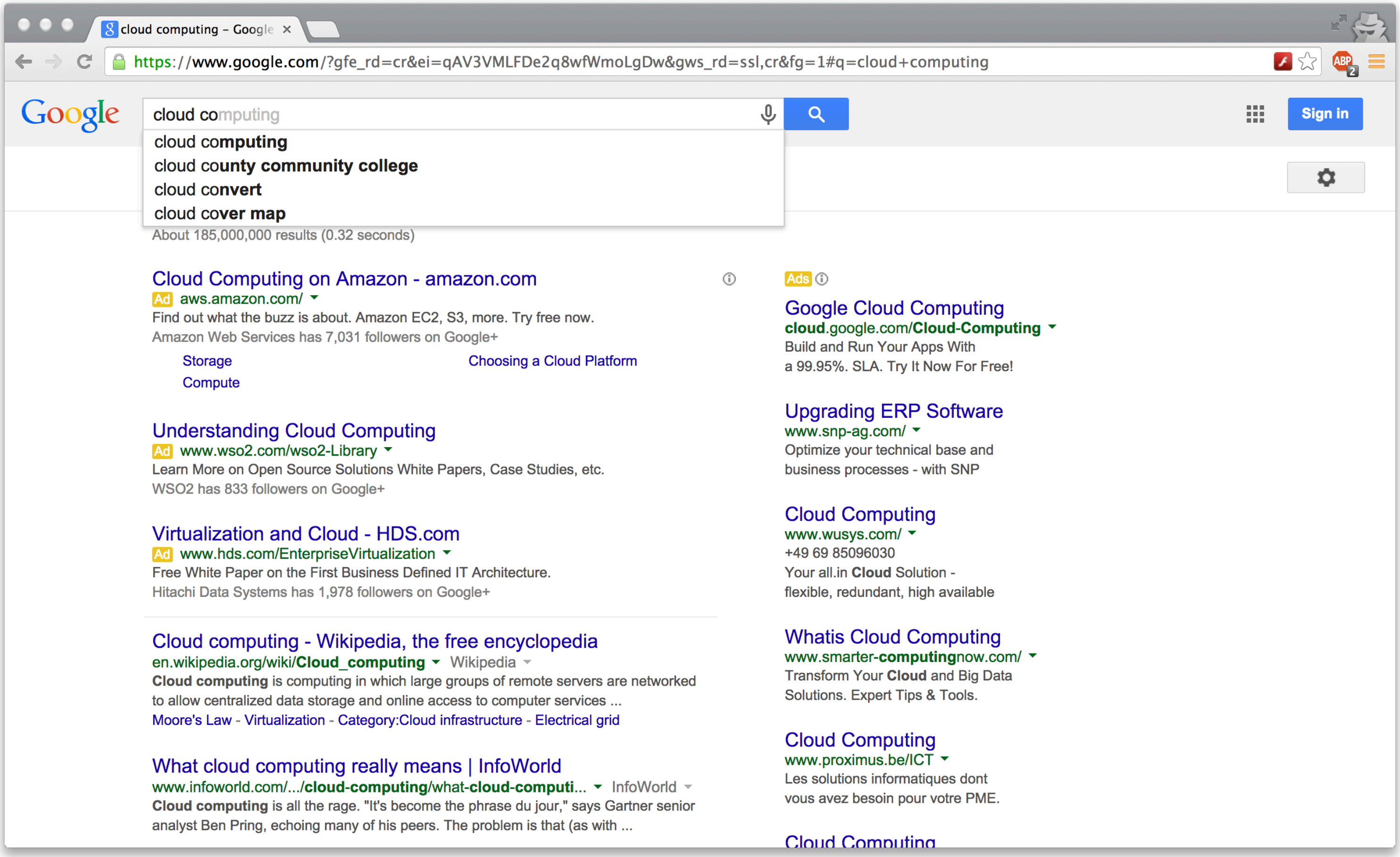
- Example: Form validation
  - Warns the user if no search term is specified

© 2015 M. Canini

Including JavaScript in HTML

- Option #1: Embed entirely in HTML document
  - As in previous example: Use <script>...</script>
  - To be safe, enclose script in HTML comments (to avoid confusing browsers that don't recognize JavaScript)
- Option #2: Attach as a separate file
  - <script type="text/javascript" src="myscript.js"> </script>
  - Some browsers need the space between <script>...</script> and don't load the script if this is omitted
- Remember that script is visible to the client!
  - Do NOT hardcode any passwords or include any secrets

Implementing search suggestions


cloud computing - Google
https://www.google.com/?gfe_rd=cr&ei=qAV3VMLFDe2q8wfWmoLgDw&gws_rd=ssl,cr&fg=1#q=cloud+computing
Google
Sign in

cloud computing

- cloud **computing**
- cloud **county community college**
- cloud **convert**
- cloud **cover map**

About 185,000,000 results (0.32 seconds)

Cloud Computing on Amazon - amazon.com
Ad aws.amazon.com/
Find out what the buzz is about. Amazon EC2, S3, more. Try free now.
Amazon Web Services has 7,031 followers on Google+
Storage — Choosing a Cloud Platform
Compute

Understanding Cloud Computing
Ad www.wso2.com/wso2-Library
Learn More on Open Source Solutions White Papers, Case Studies, etc.
WSO2 has 833 followers on Google+

Virtualization and Cloud - HDS.com
Ad www.hds.com/EnterpriseVirtualization
Free White Paper on the First Business Defined IT Architecture.
Hitachi Data Systems has 1,978 followers on Google+

Cloud computing - Wikipedia, the free encyclopedia
en.wikipedia.org/wiki/**Cloud_computing** Wikipedia
Cloud computing is computing in which large groups of remote servers are networked to allow centralized data storage and online access to computer services ...
Moore's Law - Virtualization - Category:Cloud infrastructure - Electrical grid

What cloud computing really means | InfoWorld
www.infoworld.com/.../**cloud-computing**/what-**cloud-computi**... InfoWorld
Cloud computing is all the rage. "It's become the phrase du jour," says Gartner senior analyst Ben Pring, echoing many of his peers. The problem is that (as with ...

Ads

Google Cloud Computing
cloud.google.com/**Cloud-Computing**
Build and Run Your Apps With a 99.95%. SLA. Try It Now For Free!

Upgrading ERP Software
www.snp-ag.com/
Optimize your technical base and business processes - with SNP

Cloud Computing
www.wusys.com/
+49 69 85096030
Your all.in **Cloud** Solution - flexible, redundant, high available

Whatis Cloud Computing
www.smarter-**computing**now.com/
Transform Your **Cloud** and Big Data Solutions. Expert Tips & Tools.

Cloud Computing
www.proximus.be/ICT
Les solutions informatiques dont vous avez besoin pour votre PME.

Cloud Computing

- How would you do this with pure JavaScript?

XMLHttpRequest

- A JavaScript object that enables web pages to dynamically load more content
  - Example: Ask the server for search suggestions while the user is typing the search term
- Request can be asynchronous
  - Browser performs the HTTP request in the background while the user continues to interact with the web page
  - Script defines a callback function that should be invoked when the requested content has arrived
- Content does not have to be XML
  - But often is XML (or JSON)

XMLHttpRequest workflow

1. Instantiate a new XMLHttpRequest object
2. Prepare the object
   - Call open() to set the URL and the method (GET, POST, ...)
   - Can add headers, HTTP authentication, ...
   - Need to send a callback function that will be called by the browser when the results are available
3. Send the request
   - Invoke send(), optionally with data to submit (for POST)
4. Handle invocations of the callback function
   - Do something with the response if request was successful
   - Optionally, handle errors

Example: Client side

```
<html>
  <head><title>Test</title></head>
  <script type="text/javascript">
  <!--
    function updateSuggestions() {
      var term = document.getElementById('abc').value;
      request = new XMLHttpRequest();
      request.open("GET", "http://localhost:8080/suggest/"+escape(term));
      request.onreadystatechange = function() {
        if ((request.readyState == 4) && (request.status == 200)) {
          var xmldoc = request.responseXML;
          var root = xmldoc.getElementsByTagName('root').item(0);
          var elements = root.getElementsByTagName('element');
          var htmlOut = (elements.length)+ " suggestion(s):<br><br>";
          for (var i=0; i<elements.length; i++)
            htmlOut += "#"+(1+i)+": "+elements.item(i).textContent+"<br>";
          document.getElementById('xyz').innerHTML = htmlOut;
        }
      }
      request.send();
    } // -->
  </script>
  <body>
    <h1>Input a search term</h1>
    <form action="" method="" onSubmit="return false">
      <input type="text" name="thetext" size="20" id="abc" onKeyUp="updateSuggestions()">
      <input type="submit" value="Replace">
    </form>
    <div id="xyz">(this is where the text will go)</div>
  </body>
</html>
```

URL of server-side component (servlet)

Callback function

Get data from XML

Put data into page via DOM

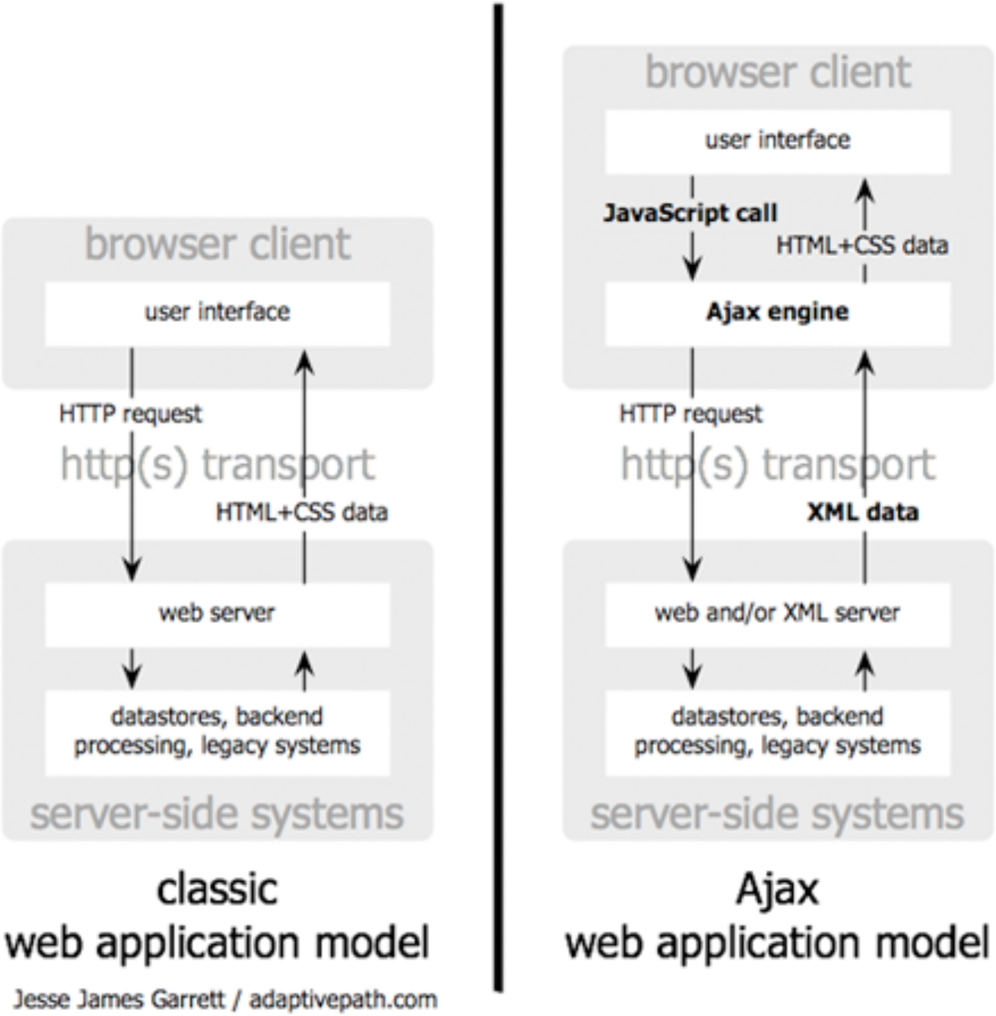
Registers event handler

What is AJAX?

- Asynchronous JavaScript and XML
  - Firsty mentioned by Jesse James Garrett in 2005

- Not a single technology - a mix of technologies for building faster web apps
  - HTML and CSS for presentation
  - DOM for dynamic display
  - XML for data interchange
  - XMLHttpRequest for asynchronous requests
  - JavaScript for binding everything together

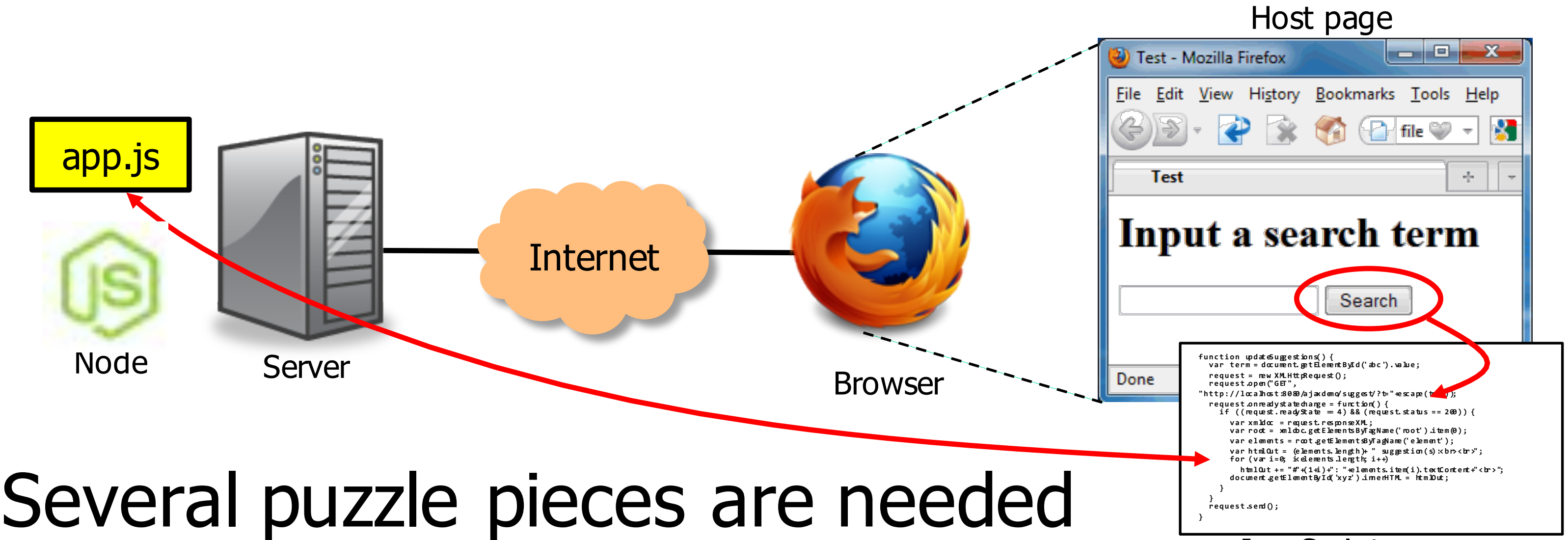
How does AJAX work?



http://www.adaptivepath.com/images/publications/essays/ajax-fig1_small.png

© 2015 M. Canini

Building Web applications with AJAX



```
function updateSuggestions() {
  var term = document.getElementById('abc').value;
  request = new XMLHttpRequest();
  request.open("GET",
"http://localhost:8080/ajaxdemo/suggest/?t="+escape(term));
  request.onreadystatechange = function() {
    if ((request.readyState == 4) && (request.status == 200)) {
      var xmldoc = request.responseXML;
      var root = xmldoc.getElementsByTagName('root').item(0);
      var elements = root.getElementsByTagName('element');
      var htmlOut = (elements.length)+ " suggestion(s):<br><br>";
      for (var i=0; i<elements.length; i++)
        htmlOut += "#"+(1+i)+": "+elements.item(i).textContent+"<br>";
      document.getElementById('xyz').innerHTML = htmlOut;
    }
  }
  request.send();
}
```

- Several puzzle pieces are needed
  - Host page: A web page that we'd like to make interactive
  - Client-side script: A JavaScript program that
    - registers handlers for relevant events, such as inputs or mouse clicks
    - requests additional data from the server using XMLHttpRequest objects
    - integrates the responses with the web page using the DOM
  - Server side: Another JavaScript program that supplies the data
    - Example: Given a partial search term, return XML with suggestions
  - Client-side script runs in browser, server-side in Node!

© 2015 M. Canini

Pros and cons of AJAX

- Much more responsive than plain HTML
  - Can avoid wide-area latency in many cases (why not all?)
  - Faster - can transfer just the required information after each interaction, rather than the entire page (+less bandwidth)
- Difficult to integrate navigation elements
  - 'Back' button, bookmarks, external links from other pages etc. require special care (window.location.hash)
- Difficult to accommodate search engines
  - Need to use site maps or carefully construct initial page
- JavaScript compatibility issues
- Messy to develop and debug
  - But new frameworks are simplifying things: e.g., AngularJS

© 2015 M. Canini

Dynamic content

- Web application technologies
  - Background: CGI ✓
  - Java Servlets ✓
- Node.js / Express / EJS ✓
- AJAX ✓
- Session management and cookies NEXT

© 2015 M. Canini

Client-side vs server-side (last time)

- What if web app needs to remember information between requests in a session?
  - Example: Contents of shopping cart, login name of user, ...
- Recap from last time: Client-side/server-side
  - Even if the actual information is kept on the server side, client still needs some kind of identifier (session ID)
- Now: Discuss four common approaches
  - URL rewriting and hidden variables
  - Cookies
  - Session object

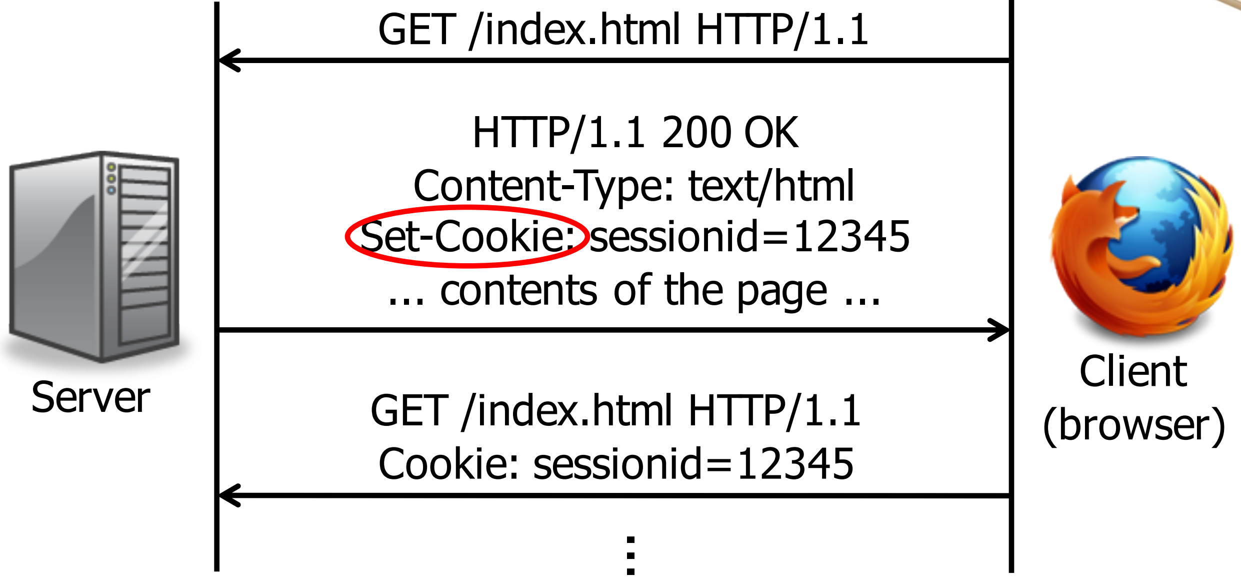
© 2015 M. Canini

URL rewriting and hidden variables

- Idea: Session ID is part of every URL
  - Example 1: http://my.server.com/shoppingCart?sid=012345
  - Example 2: http://my.server.com/012345/shoppingCart
  - Why is the first one better?
- Technique #1: Rewrite all the URLs
  - Before returning the page to the client, look for hyperlinks and append the session ID
    - Example: <a href="foo.html"> → <a href="foo.html?sid=012345">
    - In which cases will this approach not work?
- Technique #2: Hidden variables
  - <input type="hidden" name="sid" value="012345">
  - Hidden fields are not shown by the browser

© 2015 M. Canini

HTTP cookies



- What is a cookie?
  - A set of key-value pairs that a web site can store in your browser (example: 'sessionid=12345')
  - Created with a Set-Cookie header in the HTTP response
  - Browser sends the cookie in all subsequent requests to the same web site until it expires

© 2015 M. Canini

Node solution: express.session

```
var cookieParser = require('cookie-parser')
var session = require('express-session')
app.use(cookieParser());
app.use(session({secret: 'thisIsMySecret'});
...
app.get('/test', function(req, res) {
  if (req.session.lastPage)
    req.write('Last page was: '+req.session.lastPage);
  req.session.lastPage = '/test';
  req.send('This is a test.');
}
```

- Abstracts away details of session management
  - Developer only sees a key-value store
  - Behind the scenes, cookies are used to implement it
  - State is stored and retrieved via the 'req.session' object

© 2015 M. Canini

A few more words on cookies

```
...
Set-Cookie: sessionid=12345;
            expires=Tue, 02-Nov-2010 23:59:59 GMT;
            path=/;
            domain=.mkse.net
...
```

- Each cookie can have several attributes:
  - An expiration date
    - If not specified, defaults to end of current session
  - A domain and a path
- Browser only sends the cookies whose path and domain match the requested page
  - Why this restriction?

© 2015 M. Canini

What are cookies being used for?

- Many useful things:
  - Convenient session management (compare: URL rewriting)
  - Remembering user preferences on web sites
  - Storing contents of shopping carts etc.

- Some problematic things:
  - Storing sensitive information (e.g., passwords)
  - Tracking users across sessions & across different web sites to gather information about them

© 2015 M. Canini

Recap: Session management, cookies

- Several ways to manage sessions
  - URL rewriting, hidden variables, cookies...
- HttpSession
  - Abstract key-value store for session state
  - Implemented by the servlet container, e.g., with URL rewriting or with cookies
- Cookies
  - Small pieces of data that web sites can store in browsers
  - Cookies can persist even after the browser is closed
  - Useful for many things, but also for tracking users