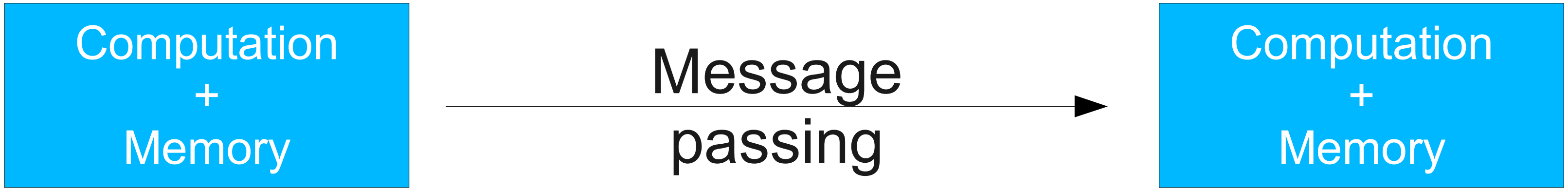
9

- Web applications
  - Distributed architecture
  - Key technologies
    - DOM
    - XML
    - AJAX

Web applications

- Definition:
  - A form of **distributed application** in which **users** access the system through a **browsing context** and (part of the) message passing is realized over an **HTTP transport**
    - Distributed computation: web browser can execute code (typically, Javascript)
    - Distributed memory: each client has its own state
    - Message passing: HTTP as a message-passing protocol



Web applications

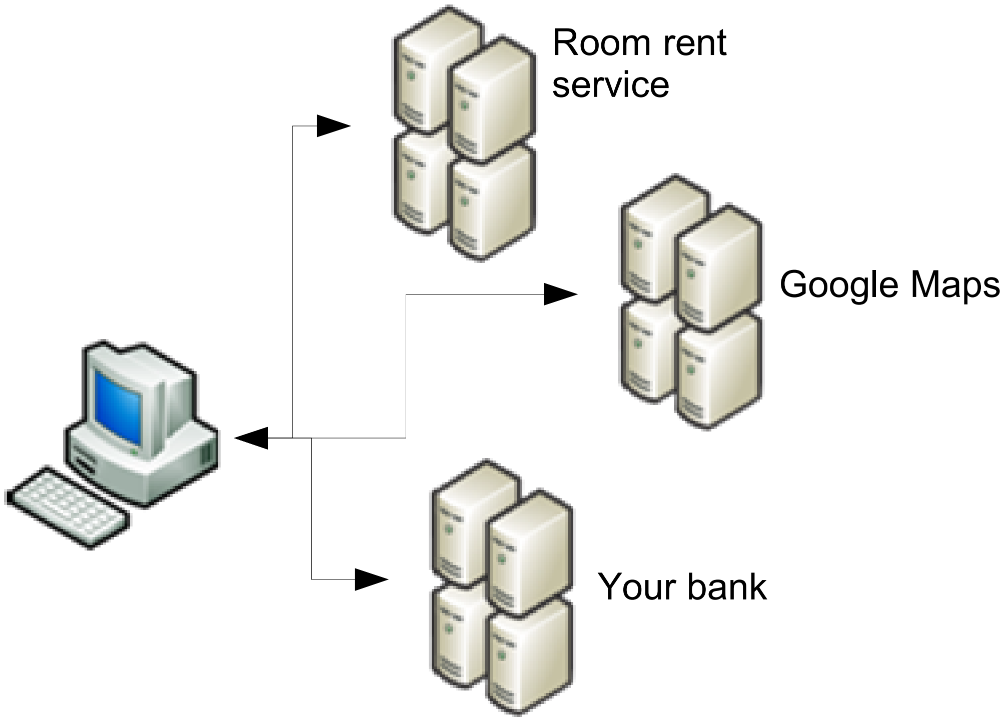
- **Components**
  - At least the user-front layer is made of components **hosted** inside a web browser or a similar browsing context
  - At least the next subsequent layer is made of **web server** components, that can talk to web browsers
    - Not necessarily a full-fledged web server or application server
    - “Micro” HTTP servers are common
  - Further components as deemed necessary

Web applications

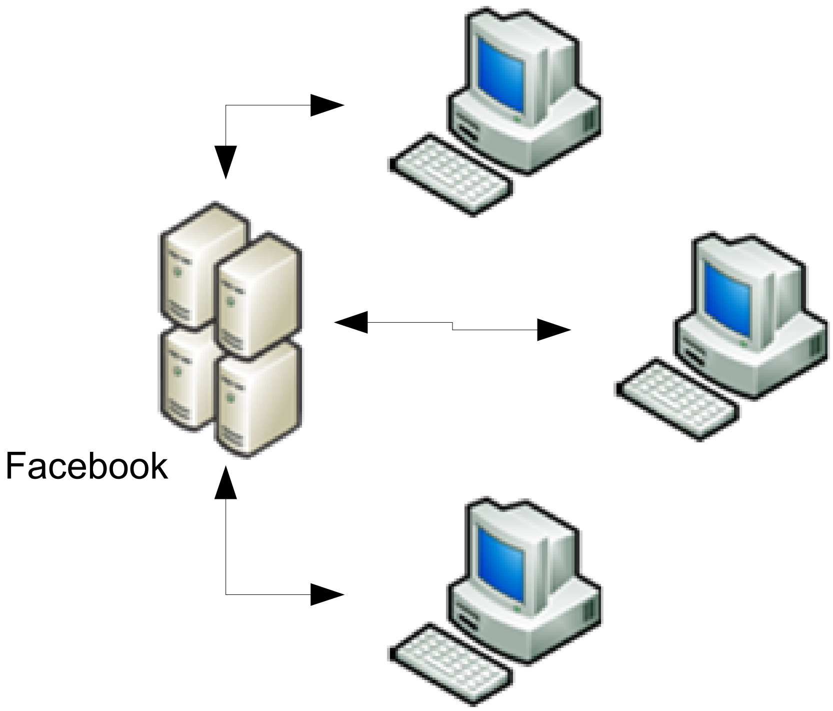
- **Connectors**
  - The connectors between user-front and the first server layer are based on HTTP transport
    - First message **must** be an actual HTTP request
    - Further messages **could** be any format... but HTTP is still the preferred method
  - Connectors between the server layer and further nodes could use different transports
    - Common case: JDBC/ODBC to connect to a DBMS
  - User-front nodes **could** use additional techniques
    - Flash, Java Applets, ActiveX ...

Web applications

- Mash-up
  - One client can access many servers

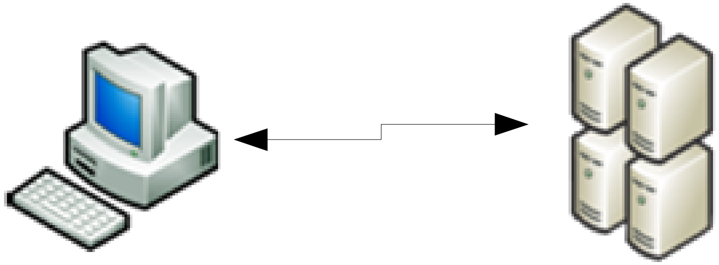


- Server-centric
  - Many clients can access a server

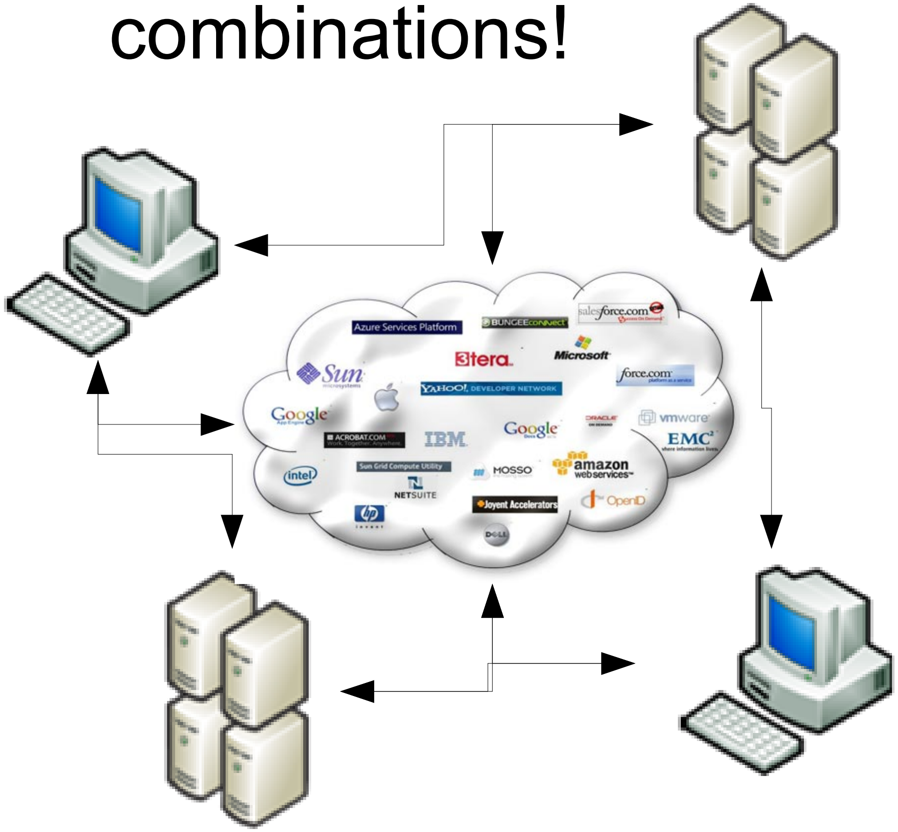


Web applications

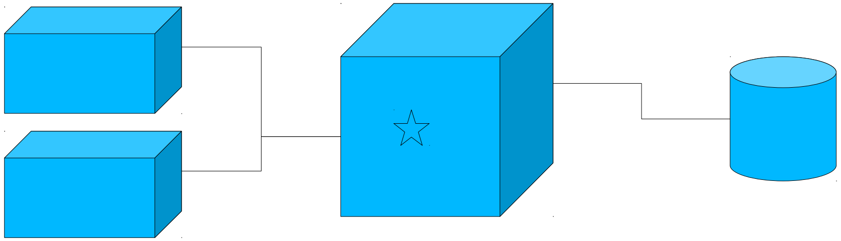
- Hosted application
  - Multiple clients access a server, but are independent of each other
  - Essentially, a traditional client-server architecture



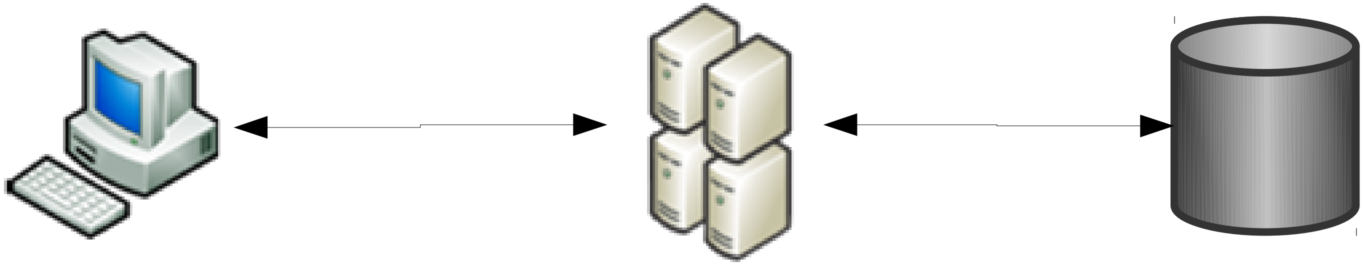
- Fully distributed
  - Endless combinations!



Typical 3-tier



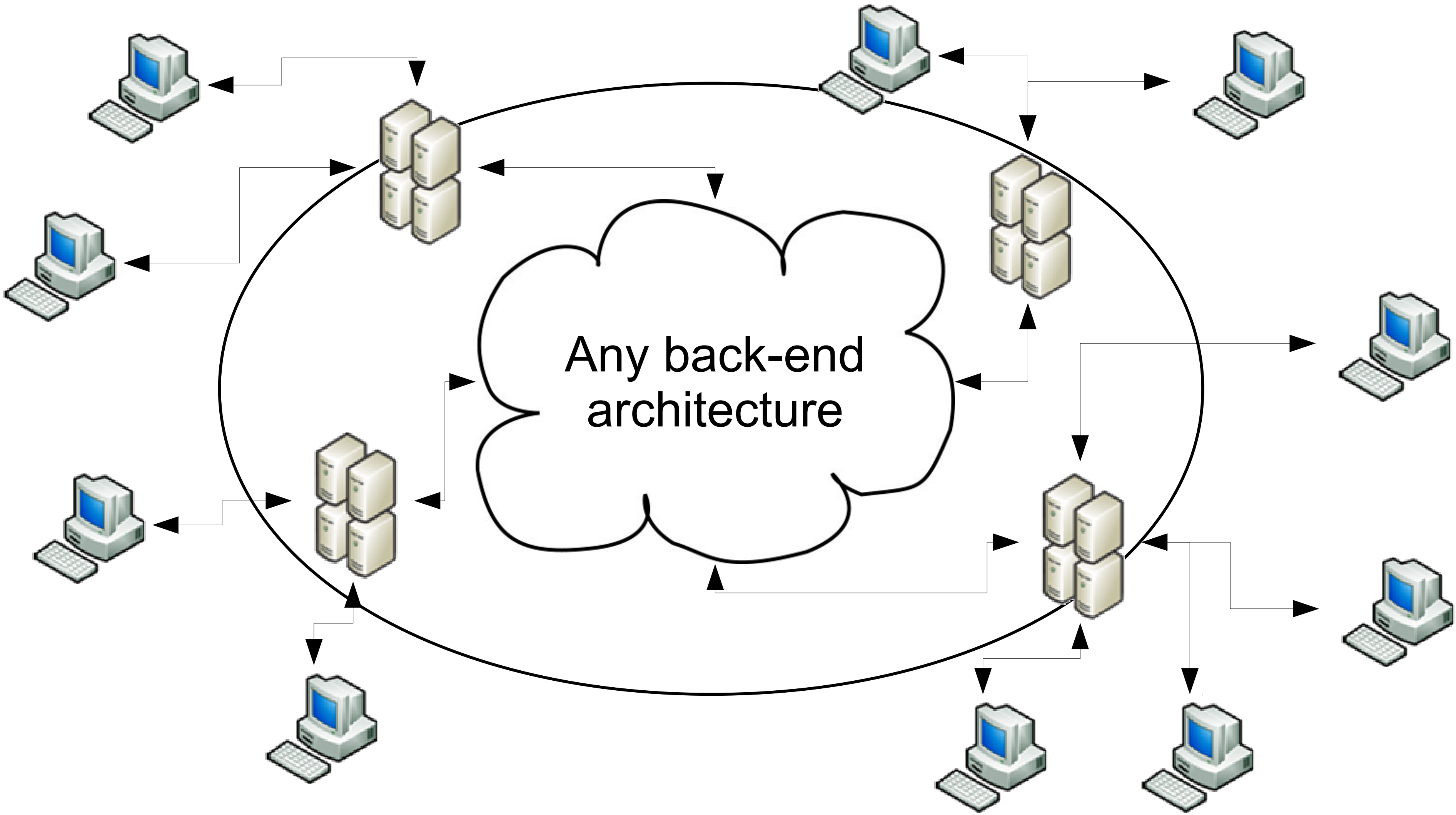
- Most web applications are implementations of the three-tier architecture
  - Most computations on the web server (application server)
  - Some UI-related computations on the client
  - Important data on a DBMS
  - Some UI-related state on the client



Web applications: Other options?

- Clients are asymmetric
  - A web browser is **not** a web server
    - P2P architectures not possible
- Clients are not compositional
  - scalable architectures not possible
    - No Pipe & filter
    - No inbound/outbound tree, no fat tree
- Custom architectures
  - Behind the user-front layer, everything goes!

Custom architectures



HTTP Transport

- Browser sends a HTTP Request to Web Server

```
GET /path/part/of/url.html HTTP/1.0
```

```
POST /path/script.cgi HTTP/1.0
From: frog@jmarshall.com
User-Agent: HTTPTool/1.0
Content-Type: application/x-www-
form-urlencoded
Content-Length: 32

home=Cosby&favorite+flavor=flies
```

- Web Server sends a HTTP Response to Browser

```
HTTP/1.0 200 OK
Date: Fri, 31 Dec 1999 23:59:59
GMT
Content-Type: text/html
Content-Length: 1354


<html>
<body>
<h1>Title</h1> ...
</body>
</html>
```

HTTP Transport

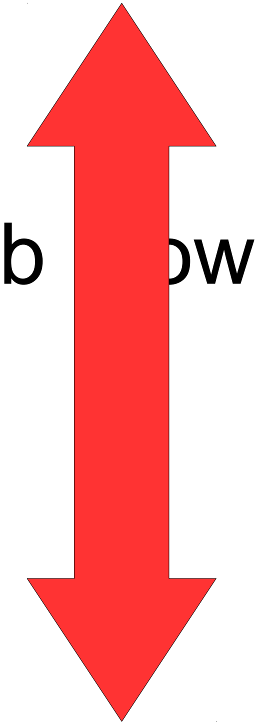
- Most commonly, HTTP is transported itself over TCP/IP
  - One could also use other protocols – but rare
- **Stateless**: each Request/Response pair is a complete communication
  - Any state-related information must be explicitly transported in the request/response
    - Cookies, session-IDs, user-IDs, …
- All communications **initiated by the client**

Key technologies for web apps

- Graphical user interface hosted in a web browser
  - HTML, CSS
- Running code in a web browser
  - Javascript
  - Browser plug-ins
    - Java applet
    - Flash
    - ActiveX components

Key technologies for web apps

- Graphical user interface hosted in a web browser
  - HTML, CSS
- Running code in a web browser
  - Javascript
  - Browser plug-ins
    - Java applet
    - Flash
    - ActiveX components

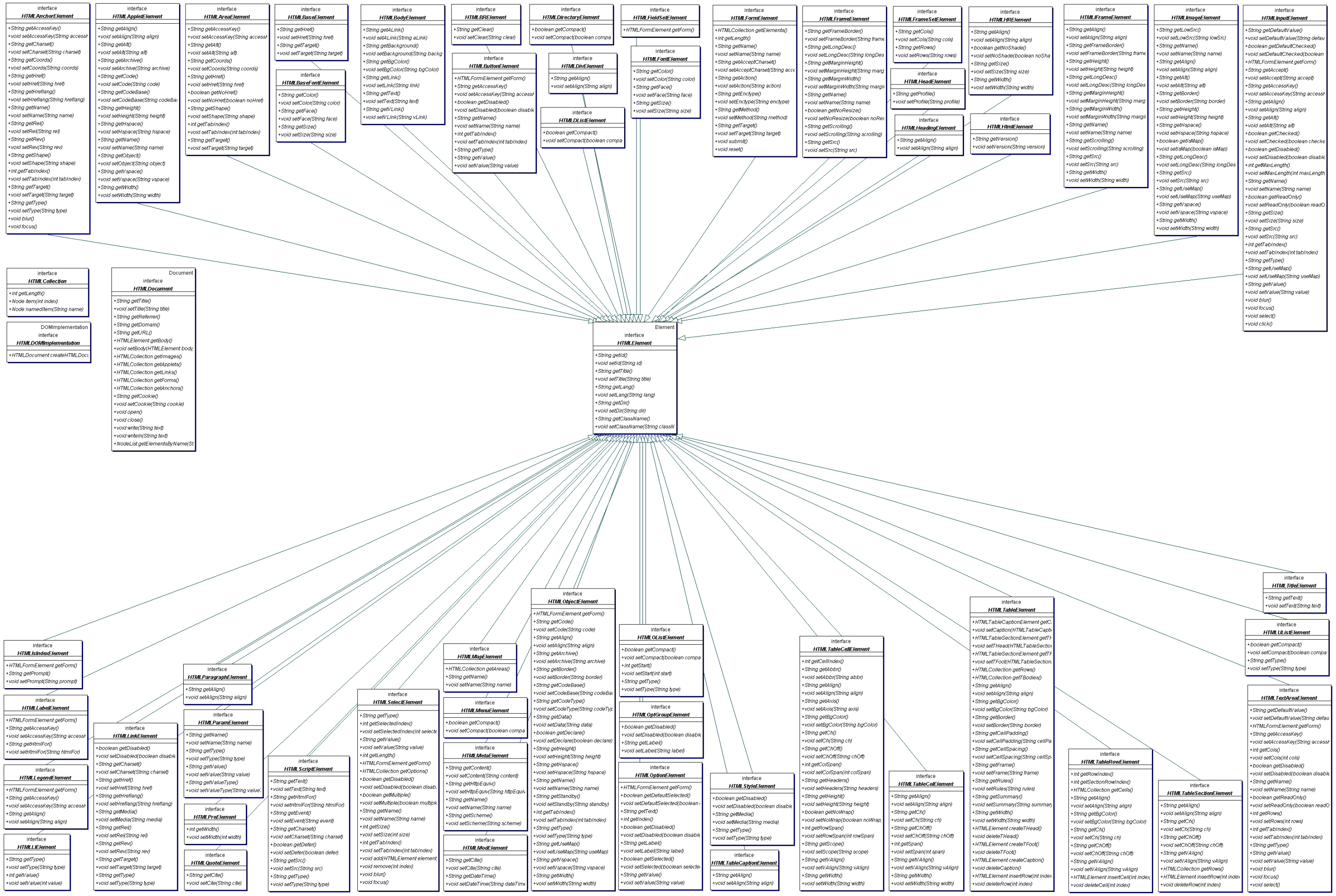


Linking the two sides:
Document Object Model (DOM)

Key technologies for web apps

- Communication via HTTP
  - HTML FORMs
  - Program-controlled HTTP requests
  - General-purpose XML requests over HTTP
- Gluing it all together
  - **AJAX**: Asynchronous Javascript and XML
    - Develop client- and server-side separately
- More advanced frameworks
  - **GWT**: Google Web Toolkit
    - Generates client- and server-side from same source

Document Object Model



- Standard library of Javascript classes
- One object for each node in the Document

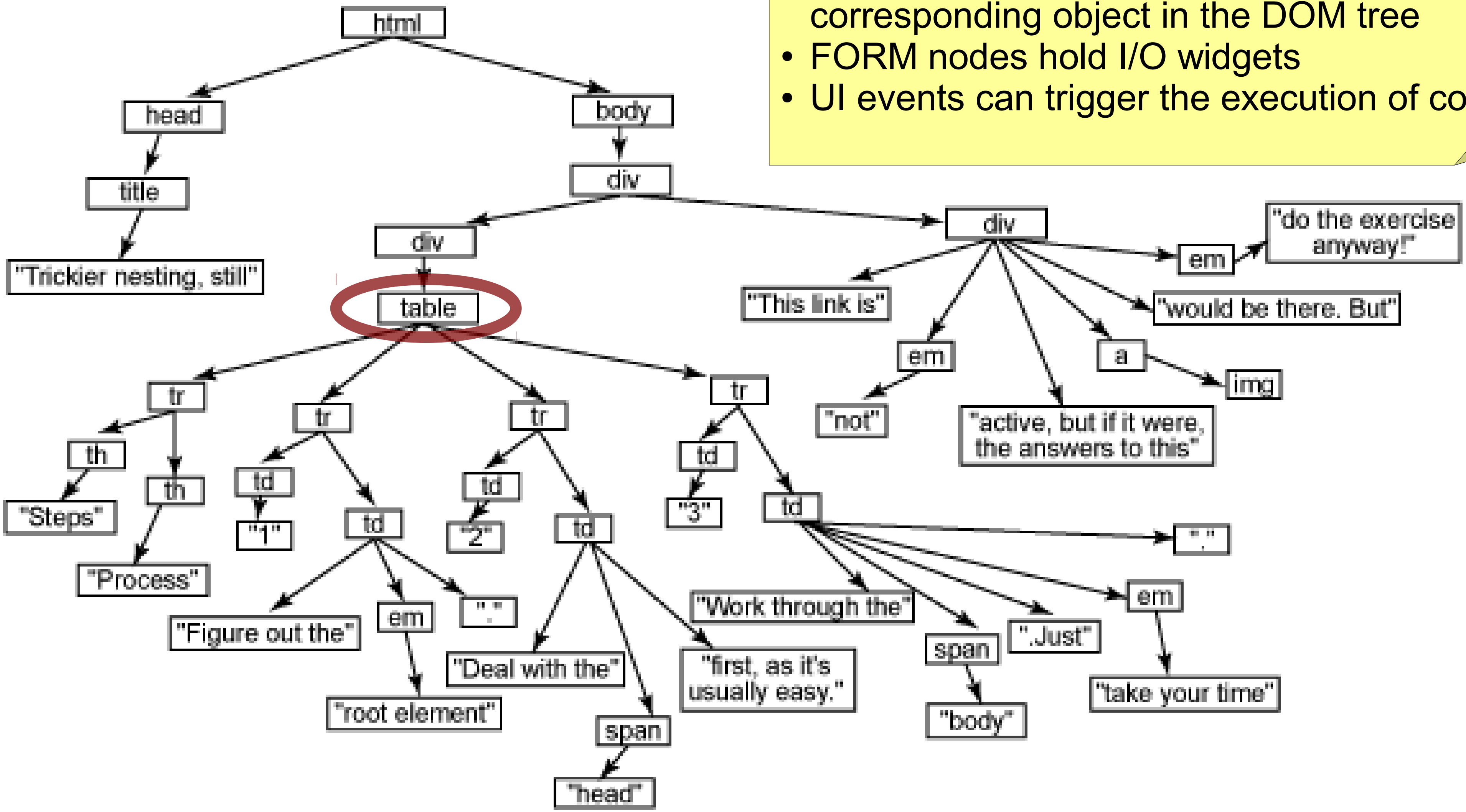
Document Object Model

- When parsing an HTML page, the browser builds a DOM tree
- HTML (and XML) are natural encodings of an attributed tree

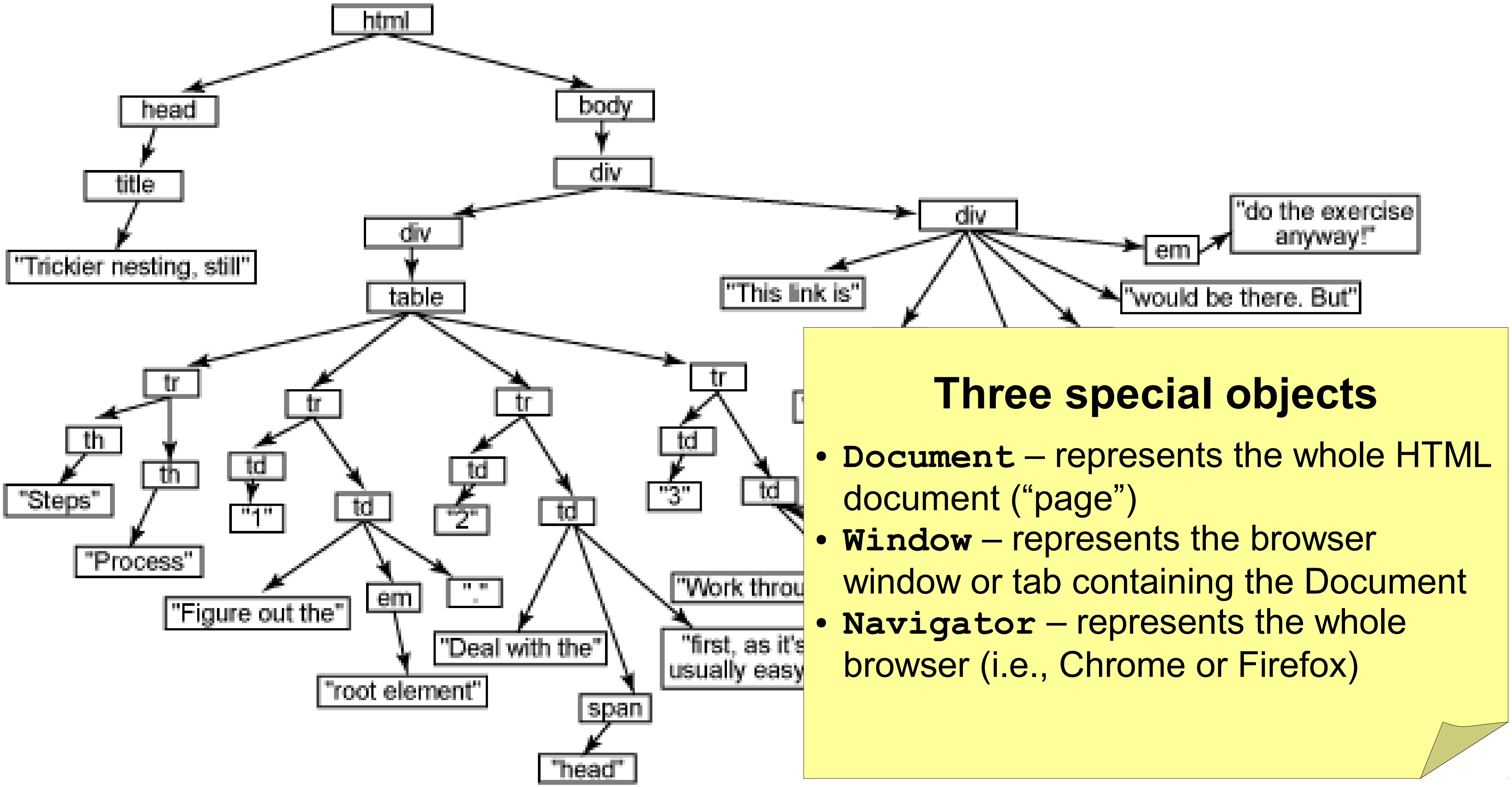
```
<html>
<head>
<title>Trickier nesting, still</title>
</head>
<body>
<div id=”main-body”>
<div id=”contents”>
<table>
<tr><th>Steps</th><th>Process</th></tr>
<tr><td>1</td><td>Figure out the <em>root
element</em>.</td></tr>
<tr><td>2</td><td>Deal with the <span id=”code”>head</span>
first,
as it’s usually easy.</td></tr>
<tr><td>3</td><td>Work through the <span id=”code”>body</span>.
Just <em>take your time</em>.</td></tr>
</table>
</div>
<div id=”closing”>
This link is <em>not</em> active, but if it were, the answers
to this <a href=”answers.html”><img src=”exercise.gif” /></a>
would be there. But <em>do the exercise anyway!</em>
</div>
</div>
</body>
</html>
```

Document Object Model

- Each node in the HTML tree has a corresponding object in the DOM tree
- FORM nodes hold I/O widgets
- UI events can trigger the execution of code



Document Object Model



Three special objects

- **`Document`** – represents the whole HTML document ("page")
- **`Window`** – represents the browser window or tab containing the Document
- **`Navigator`** – represents the whole browser (i.e., Chrome or Firefox)

Example

```
<form name="ex" method="POST"
onsubmit="alert('onsubmit');return false;">
<div align="center">
<select name="sel" size="1"
onchange="alert('onchange')">
<option value="1" selected="selected">1</option>
<option value="2">2</option>
<option value="3">3</option>
</select>
<input type="submit" value="submit" />
</div></form>
```

UI events supported (HTML 5)

All HTML elements, Document object, Window object	All HTML elements except BODY, Document object	Window object
onabort	onblur	onafterprint
oncanplay	onerror	onbeforeprint
oncanplaythrough	onfocus	onbeforeunload
onchange	onload	onblur
onclick	onscroll	onerror
oncontextmenu	onloadstart	onfocus
oncuechange	onmousedown	onhashchange
ondblclick	onmousemove	onload
ondrag	onmouseout	onmessage
ondragend	onmouseover	onoffline
ondragenter	onmouseup	ononline
ondragleave	onmousewheel	onpagehide
ondragover	onpause	onpageshow
ondragstart	onplay	onpopstate
ondrop	onplaying	onredo
ondurationchange	onprogress	onresize
onemptied	onratechange	onscroll
onended	onreadystatechange	onstorage
oninput	onreset	onundo
oninvalid	onseeked	onunload
onkeydown	onseeking	onsuspend
onkeypress	onselect	ontimeupdate
onkeyup	onshow	onvolumechange
onloadeddata	onstalled	onwaiting
onloadedmetadata	onsubmit	

- Useful events
  - change
  - click
  - drag*/drop
  - key*
  - mouse*
  - submit
  - load/unload
  - error

Traditional web “application”

- So-called “post-back” model
  - Some user action triggers an application event
  - The application posts the event (as a FORM) to the web server
  - Application code on the server receives the data from the form, and performs whatever action was requested
  - The server generates a whole new web page, updated according to the user action
  - The new web page is shipped to the browser
  - Rinse and repeat

Traditional web “application”

- So-called “post-back” model
  - Some user a
  - The applicati the web serv
  - Application c from the form requested
  - The server generates a whole new web page, updated according to the user action
  - The new web page is shipped to the browser
  - Rinse and repeat

Terribly wasteful!

- **An entire round-trip (client to server and back) for each user action → high latency**
- **An entire Document sent as response for each user action → low throughput**

AJAX-style application

- So called “differential update” model
  - Some user action triggers an application event
  - The (client-side) application's code crafts a message (in XML) to be sent to the server
  - The server receives the messages, performs the action, and generates an arbitrary reply message
  - The reply is received by the client-side code, which uses it to update the current page (**updating**)
  - Rinse and repeat

AJAX-style application

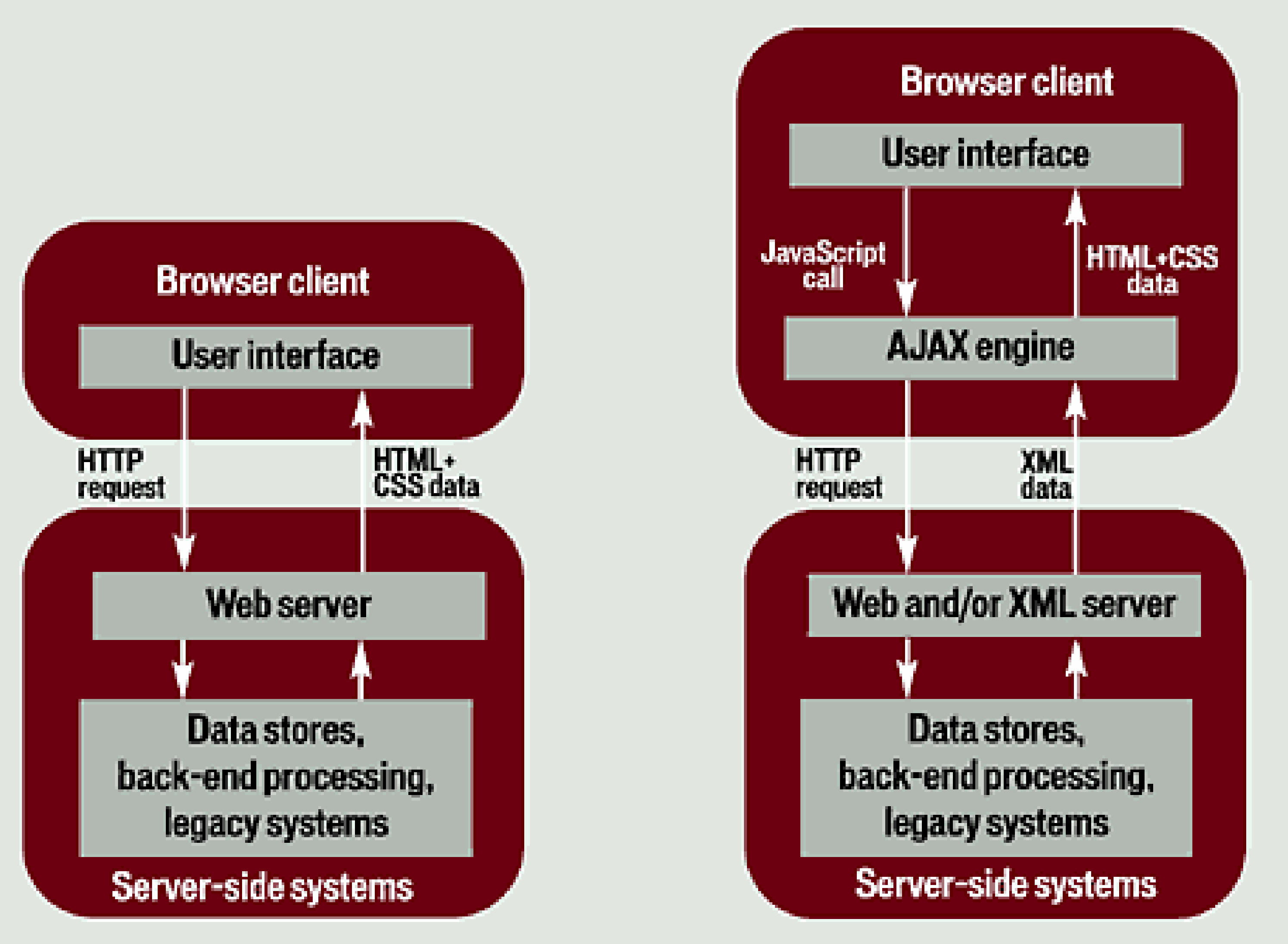
- So called “differential update” model
  - Some user action triggers an application event
  - The (client-side) application's code crafts a message (in XML) to be sent to the server
  - ges, performs the
    ary reply message
  - ent-side code, which
    ge (**updating**)

Much more efficient!

- **Simple updates can be performed locally, no need to go to the server → low latency**
- **Only the changed data are sent back to the client → high throughput**

Traditional vs. AJAX

- Typical AJAX applications rely on a library for routine tasks
- This “AJAX engine” supports **serialization** of objects (JSON)



Coding AJAX (client-side)

- Communication between client and server is performed through an XMLHttpRequest

create

```
var req;
req = new XMLHttpRequest();
...
```

recv

```
req.onreadystatechange = function() {
   // callback function, will run when server
   // replies to the message
}
...
```

Asynchronous!

send

```
req.open('GET', url, true);
req.send(args); // req is sent to server
```

Coding AJAX (client-side)

- In the callback function, the server response can be extracted from the request object

```
function() {
   if (req.readyState === 4) {
      if (req.status === 200) {
         alert(req.responseText);
      } else {
         alert('HTTP error!');
      }
   }
}
```

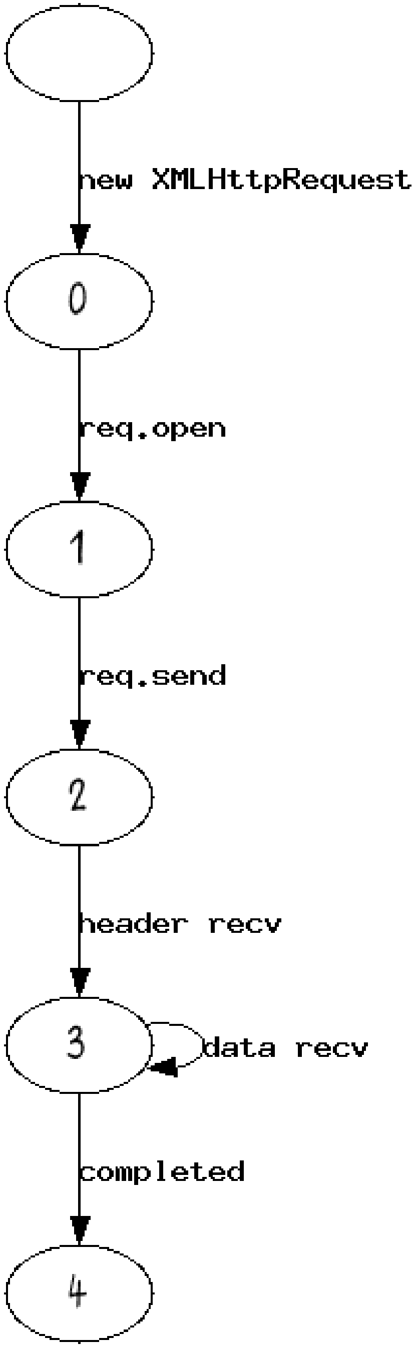
Stronger equality without type coercion

0 = uninitialized
1 = loading
2 = loaded
3 = interactive
4 = complete

Coding AJAX (client-side)

- In the callback function, the server response can be extracted from the request object

```
function() {
   if (req.readyState === 4) {
      if (req.status === 200) {
         alert(req.responseText);
      } else {
         alert('HTTP error!');
      }
   }
}
```



Coding AJAX (client-side)

- In the callback function, the server response can be extracted from the request object

```
function() {
   if (req.readyState === 4) {
      if (req.status === 200) {
         alert(req.responseText
      } else {
         alert('HTTP error!');
      }
   }
}
```

HTTP status codes

2xx = success

200 = ok

201 = created

202 = accepted

204 = no content

…

3xx = redirection

301 = moved

…

4xx = client error

401 = unauthorized

403 = forbidden

404 = not found

…

5xx = server error

501 = not implemented

503 = unavailable

Coding AJAX (client-side)

- In the callback function, the server response can be extracted from the request object

```
function() {
   if (req.readyState === 4) {
      if (req.status === 200) {
         alert(req.responseText);
      } else {
         alert('HTTP error!')
      }
   }
}
```

Whatever the server sent in response to the request

Coding AJAX (client-side)

- The responseText can be used in any way the programmer sees fit
- Some typical uses
  - ResponseText contains a fragment of HTML
    - The client-side code inserts the fragment at an appropriate position in the current page

```
Document.getElementById(mountPoint).innerHTML=req.responseText
```

  - ResponseText contains a serialized object
    - The client-side code deserializes it and uses it in some way

Using JSON with AJAX

- **JSON** (Javascript Serialized Object Notation / JavaScript Object Notation) is a simple standard for representing objects as strings
- A Javascript object is a map: key → value
- Values can be
  - Basic types: integers, floats, strings, booleans...
  - Objects: a nested map
  - Functions: executable code (λ-expressions)
  - Arrays of the above
  - **`null`**

Not in JSON!

Using JSON with AJAX

```
{
     "firstName": "John",
     "lastName": "Smith",
     "age": 25,
     "address":
     {
         "streetAddress": "21 2nd Street",
         "city": "New York",
         "state": "NY",
         "postalCode": "10021"
     },
     "phoneNumber":
     [
         {
           "type": "home",
           "number": "212 555-1234"
         },
         {
           "type": "fax",
           "number": "646 555-4567"
         }
     ]
 }
```

- JSON is the **native** format for object literals in Javascript
- Hence, if **`p`** is a string containing the text on the left, we can write

```
var john =
  eval('('+p+')');
```

Using JSON with AJAX

- Some care must be taken with escaping random strings to be passed to **`eval()`**
- Better alternative: use the JSON utility object
  - Has a method **`parse()`** specifically for JSON data

```
var result = {};
var req = new XMLHttpRequest();
req.open("GET", url, true);
req.onreadystatechange = function () {
  if (req.readyState === 4 && req.status === 200){
       result = JSON.parse(req.responseText);
      // do something with result
  }
};
req.send(args);
```

Coding AJAX (server-side)

- To a web server/application server, HTTP requests coming from an AJAX application are business as usual
- Form-encoded input is retrieved from the HTTP Request, processed, and results are sent back in an HTTP Response
- All usual technologies are applicable
  - CGI, Java Servlet, ASP.NET, JSP, …
  - Apache, Tomcat, ad-hoc servlets, …
  - Responses could even be static files!

Example: MVC

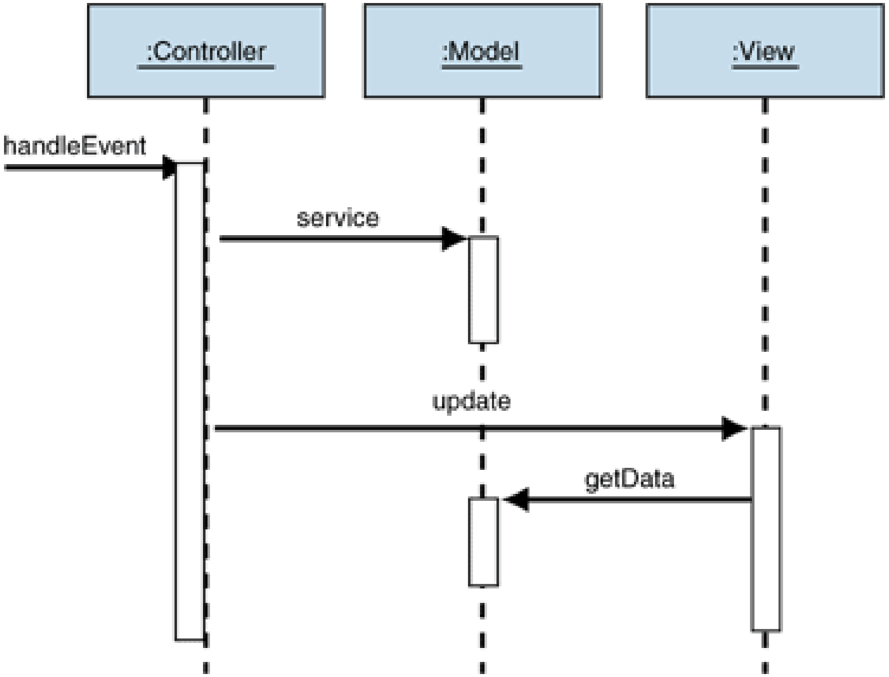
- Browser has the View
  - As a DOM = HTML document or part thereof
- Browser has the Controller
  - As Javascript code, fired by event handlers

- Server has the Model
  - The actual data
    - In-memory → 2-tier
    - In a DBMS → 3-tier

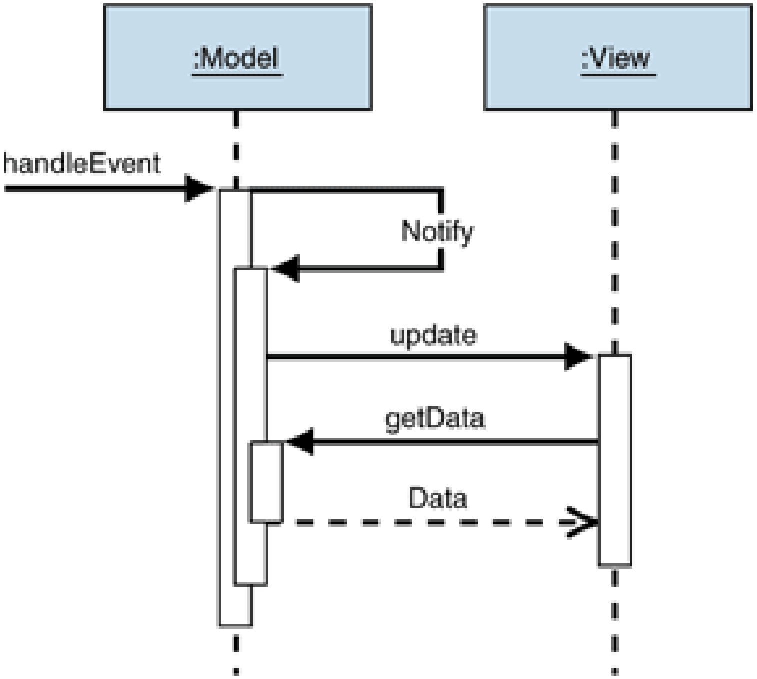
Exercise

Can you spot a problem with MVC on a typical Web Application?

Example: MVC



- Passive mode
  - Changes in the model initiated by the Controller

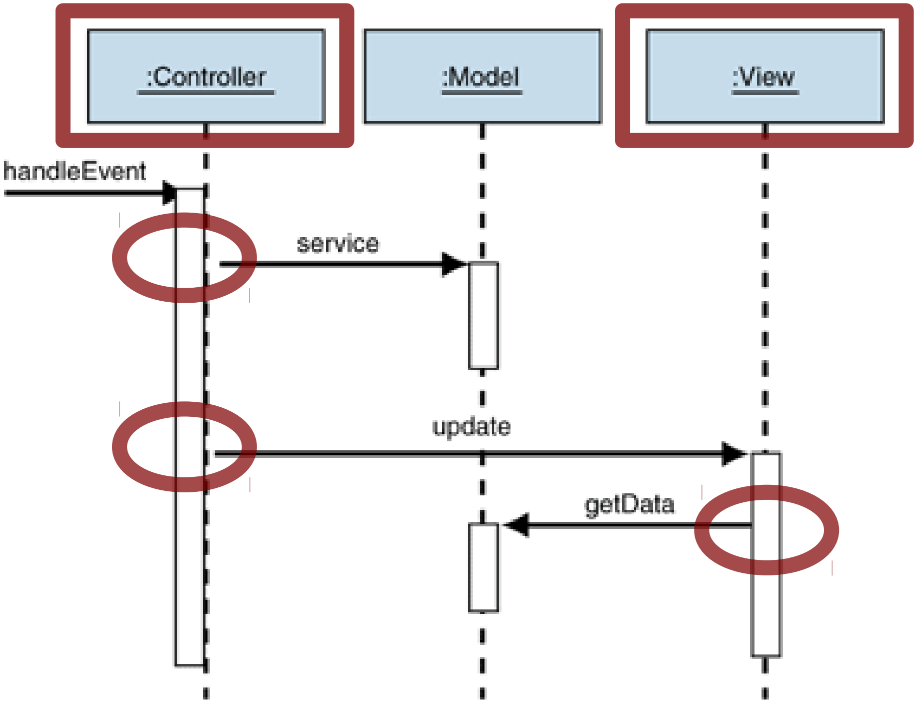


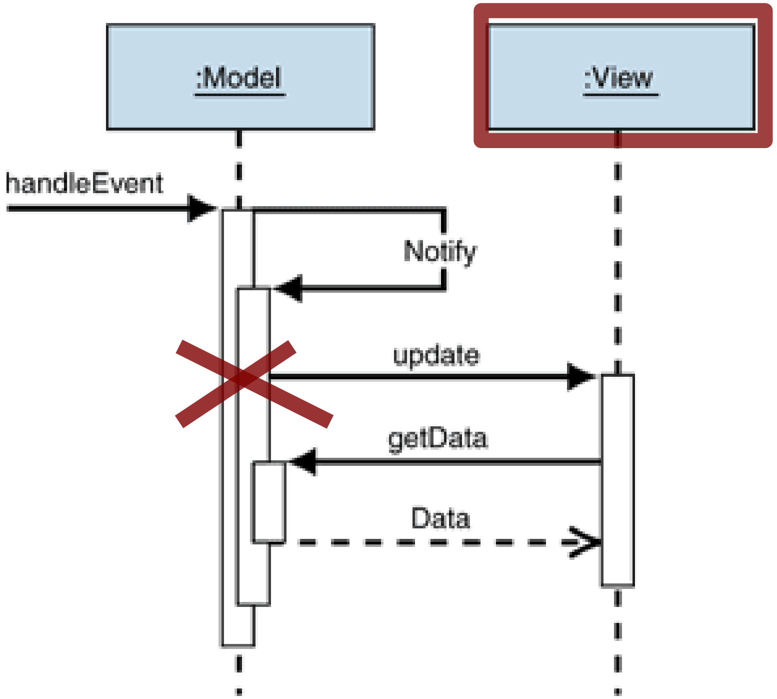
- Active mode
  - Changes in the model “spontaneous” or 3rd party

Example: MVC

Only the client (Browser) can initiate a communication

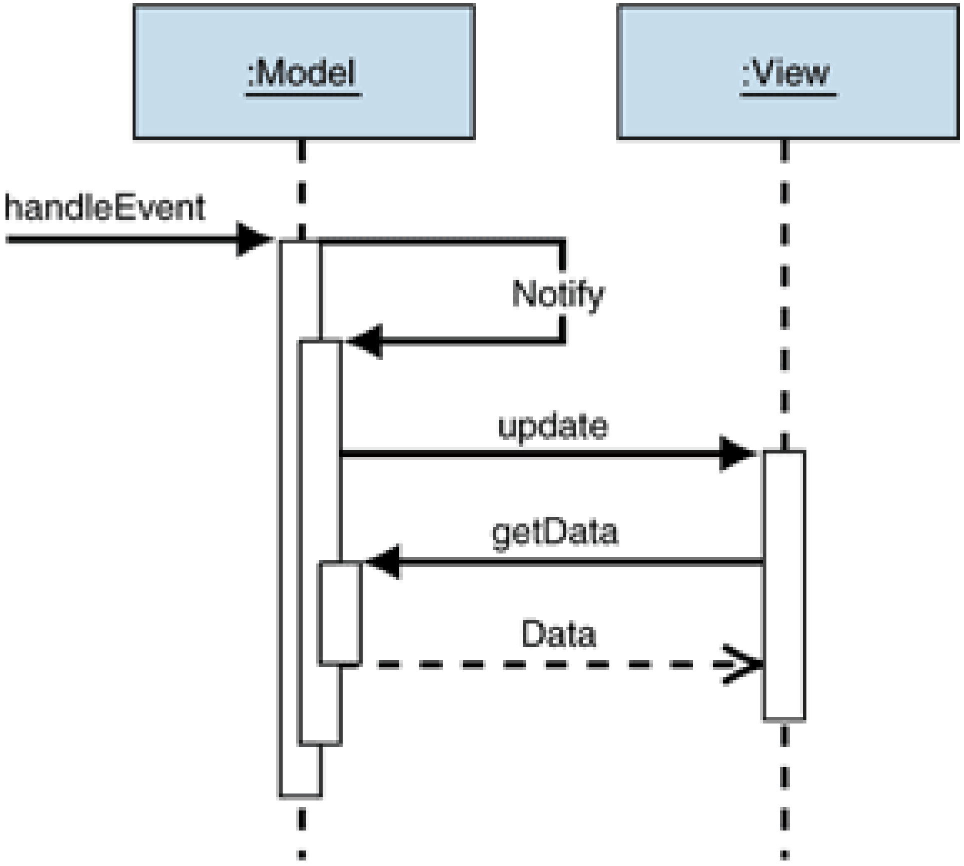
Cannot have active MVC!





- Passive mode
  - Changes in the model initiated by the Controller
- Active mode
  - Changes in the model “spontaneous” or 3rd party

Example: MVC



- However, we can use a trick!
  - AJAX is **asynchronous**
  - Simply keep a request “out”, and implement the update operation as a response to that request
  - On receiveing an update, the View must **always** send out another waitForUpdate request to the Model

Problems with AJAX

- Browsers are still only partially standardized
  - Life can be hard...
  - Writing client-side Javascript code capable of running on all and every browser is not easy
    - Often, lots of “IFs” and ad-hoc work-arounds
- AJAX requires writing a substantial amount of tricky code by hand
  - Requests handling
  - DOM manipulation
  - JSON marshalling/unmarshalling

A more complete solution: GWT

- GWT = Google Web Toolkit
- Write Java code
  - A compiler produces highly optimized Javascript code “corresponding” to the source Java code
  - A different compiled module for each supported browser/version
  - The server will serve to each user the version optimized (and bug-compatible) for his/her particular browser
    - Includes mobile environments, e.g. iPhone or Android

A more complete solution: GWT

- In favour
  - Rich set of HTML+Javascript widgets
  - Extremely robust implementation of communication, serialization, synchronization, etc.
  - Highly efficient, highly portable, future-proof
  - Good development environment
    - Embedded in Eclipse, with graphical GUI designer
- Against
  - Yet another full set of APIs and frameworks to learn!
  - Proprietary technology – no standardization