Engaging the Managing Intelligence of Nature

Computer Science 545

Web Applications: Architecture and Frameworks

Diversity Arising from Unity

Professional problem solving case study

- A manager, a mechanical engineer, and software analyst are driving back from a convention through the mountains. Suddenly, as they crest a hill, the brakes on the car go out and they fly careening down the mountain. After scraping against numerous guardrails, they come to a stop in the ditch. Everyone gets out of the car to assess the damage.
- The manager says, "Let's form a group to collaborate ideas on how we can solve this issue."
- The mechanical engineer suggests, "We should disassemble the car and analyze each part for failure."
- The software analyst says, "Let's push it back up the hill and see if it does it again."

Web Applications

- Browser to container via HTTP
- Principle form of software delivery
- Machine to machine via HTTP (web services)

- Development challenges
  - Multiple technologies and computing environments
    - Browsers
      - HTML
      - XML
      - Javascript
      - Ajax
    - Web container
      - Concurrency
      - Parse and generate HTTP
      - Networking
      - State management
      - Request and application lifecycles

Web Application Architecture

- HTTP defines the nature of the Web
  - Simple
  - Efficient
  - Scalable
- HTTP is the underlying unity or laws of nature

Web Application Frameworks

- Industry best practices for web application development
- Separation of concerns
  - Model: business logic
  - View: HTML displayed in browser
  - Controller: middleware that connects the user interface to the business logic
- Platform independent control operations
  - Retrieve input parameters from request messages
  - Conversion and validation of input parameters
  - Handling errors in parameters
  - Invoking business model
  - Create and send response messages
- Actions in accord with laws of nature

Emphasis on platform and framework independent principles

- Java platform
  - JSF (JSP)
  - JPA/Hibernate, JavaBeans, Grails, SEAM, Spring, Struts
- Comparative platforms
  - PHP
  - .Net, ASP.Net, ASPMVC
  - Ruby/Rails, Python/Django, …
- Platform independent
  - HTTP, HTML, XML, Javascript, CSS
  - Ajax
  - Web services
    - SOAP
    - REST

Course goals

- Understand nature and operation of the HTTP protocol and how it determines web application architectures and frameworks
- design and implement web applications in enterprise environments.
- understand underlying principles and patterns in cross-platform manner.

Course objectives

- understand role and relationships of HTTP, HTML, XML, Javascript, Java servlets, JSPs, JSF and other web application technologies such as Ajax
- design and implement enterprise design patterns in an MVC model-2 web architecture
- understand model-2 web versus model-1 architectures
- understand how and why web application frameworks embody good architecture patterns in lifecycles
  - request processing,
  - conversion/validation,
  - component management,
  - event handling,
  - interacting with a domain model,
  - generation of HTTP responses
- implement web applications using servlets, JSP, JSF, JSF custom tags and components, Ajax, HTML, CSS, JavaScript, and with enterprise technologies for persistence, transactions, and security
-

Lesson 1

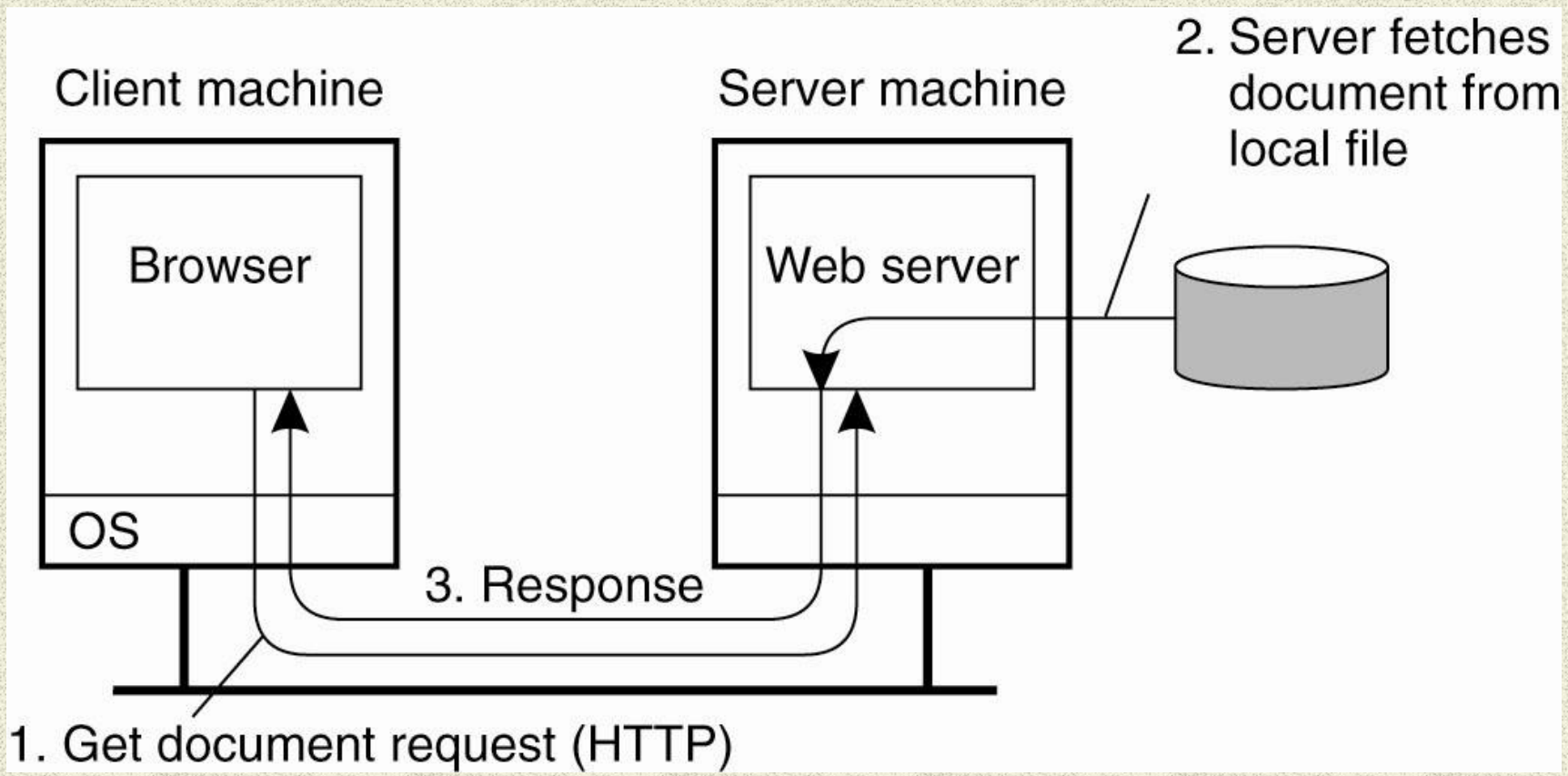
Basic Architecture of a Web App

1. Deployment and networking architecture
2. Internet architecture: TCP/IP network protocol
3. HTTP structure
    1. Anatomy
    2. Requests
    3. Responses
4. Example of http request/response between client and server

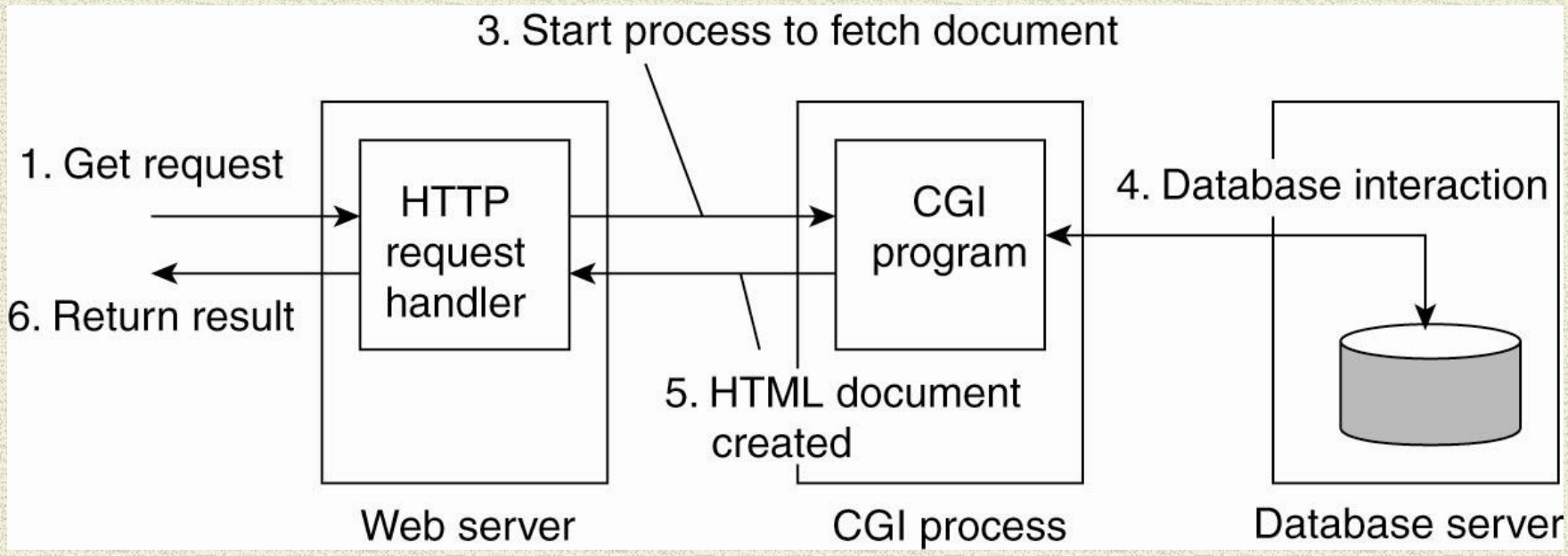
Main point 1

The basic structure of a web application is a client machine sending a request to a web server, which sends a response back to the client. Major steps in the evolution of web applications include dynamic generation of responses, machine to machine communication via web services, and more dynamic and responsive interfaces through asynchronous partial page refreshing (Ajax). **Science of Consciousness:** Parallel to the evolution we have seen in the sophistication of web application architecture, our own awareness naturally evolves to more sophisticated states of consciousness, and this process is greatly accelerated through the regular experience of pure consciousness.

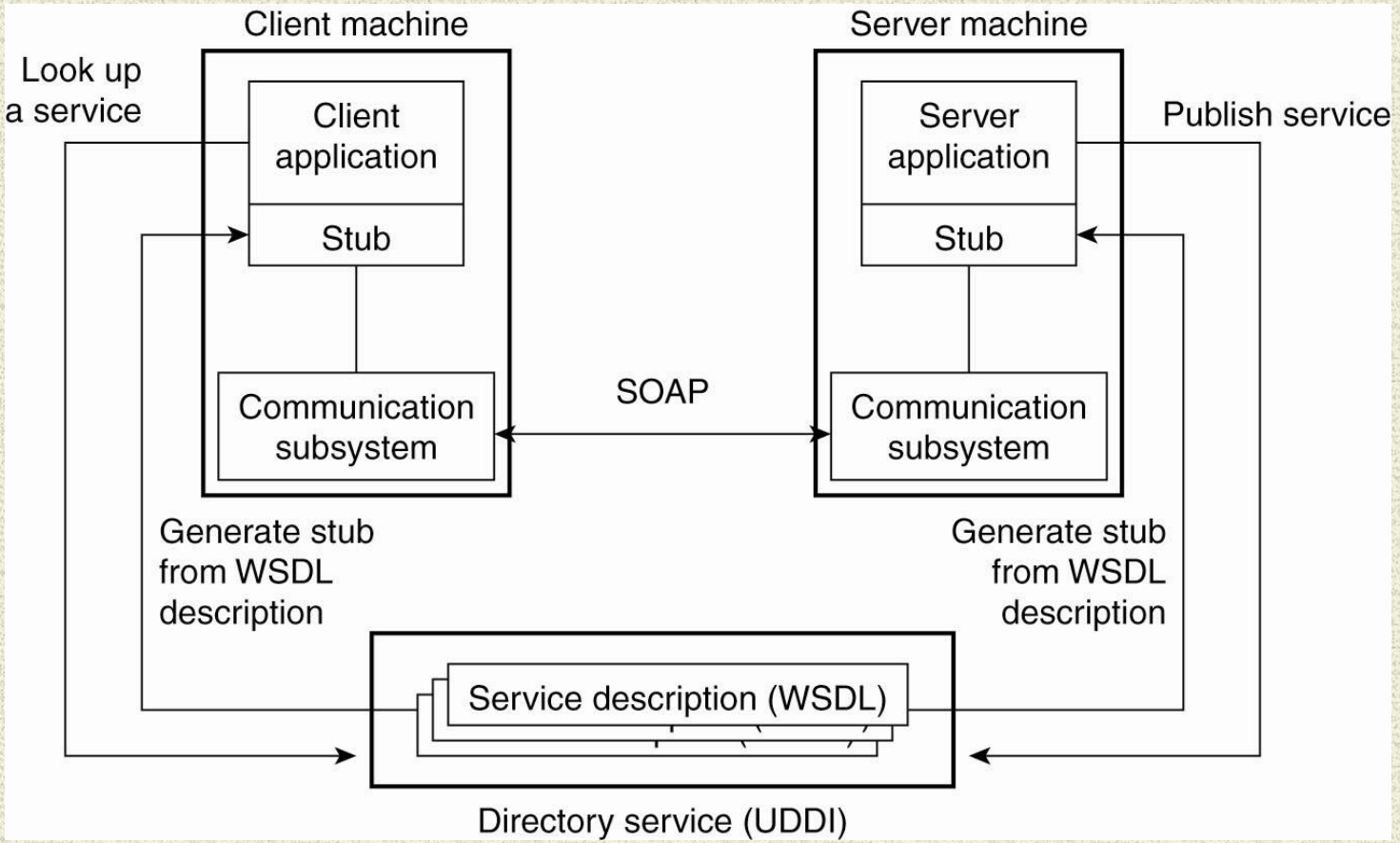
Original Web-Based Systems



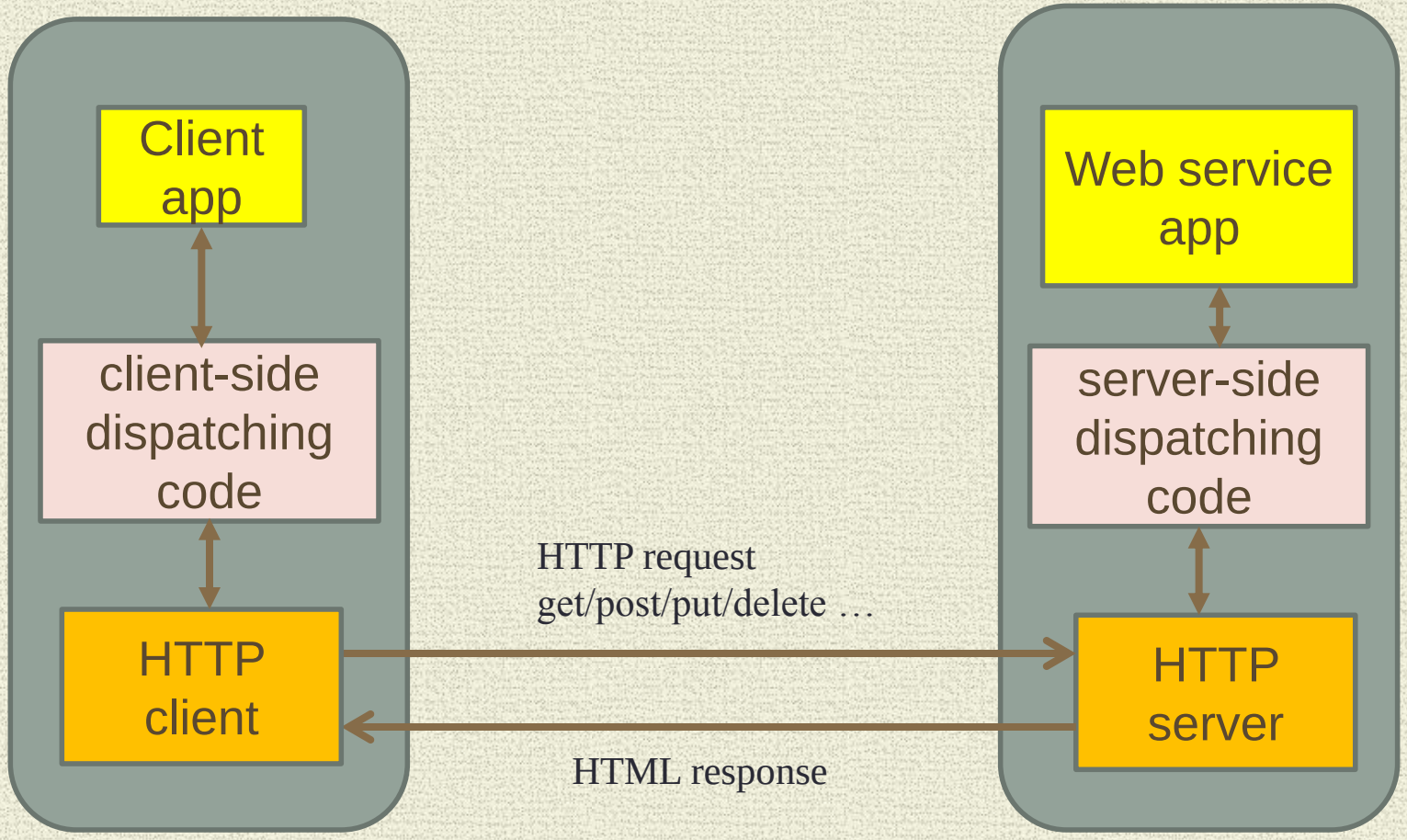
Multitiered Architectures



SOAP Web Service Architecture

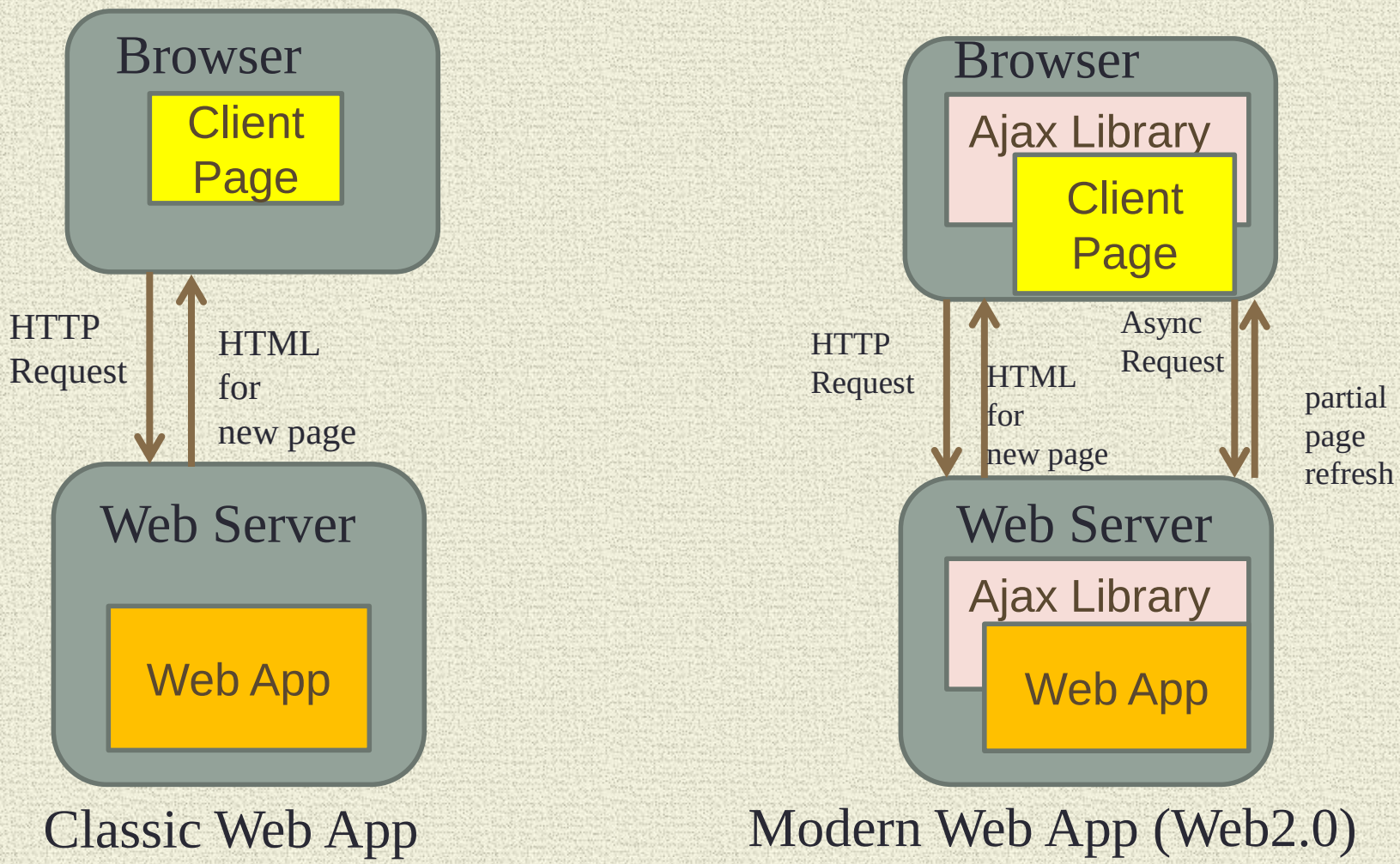


RESTful Web Service Architecture

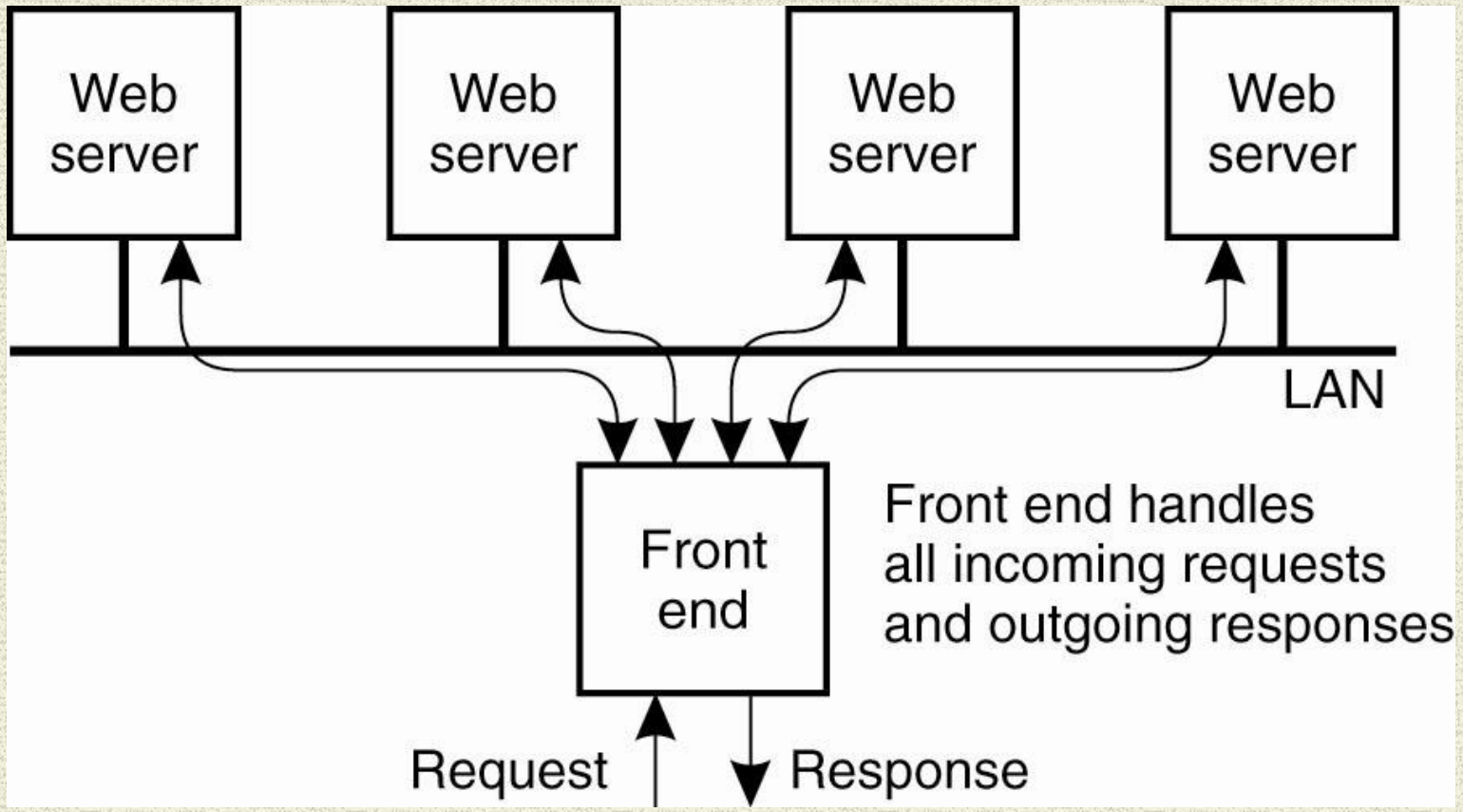


Modern Web App

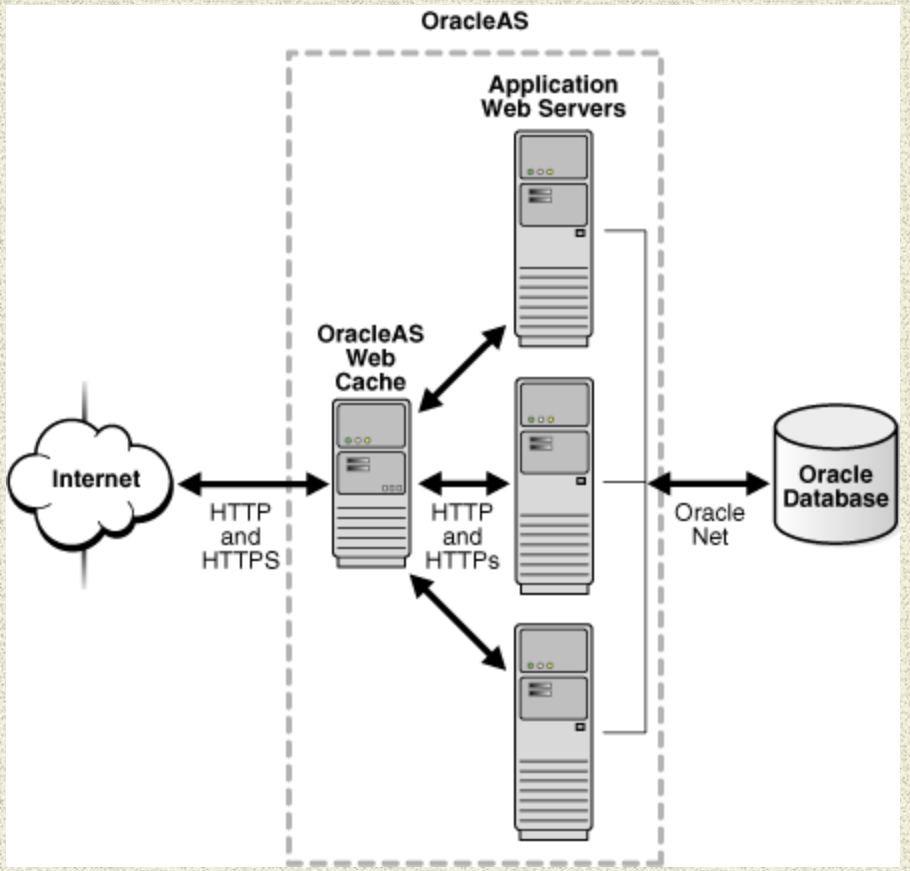
1. Rich and responsive GUI
2. Replacing many desktop applications



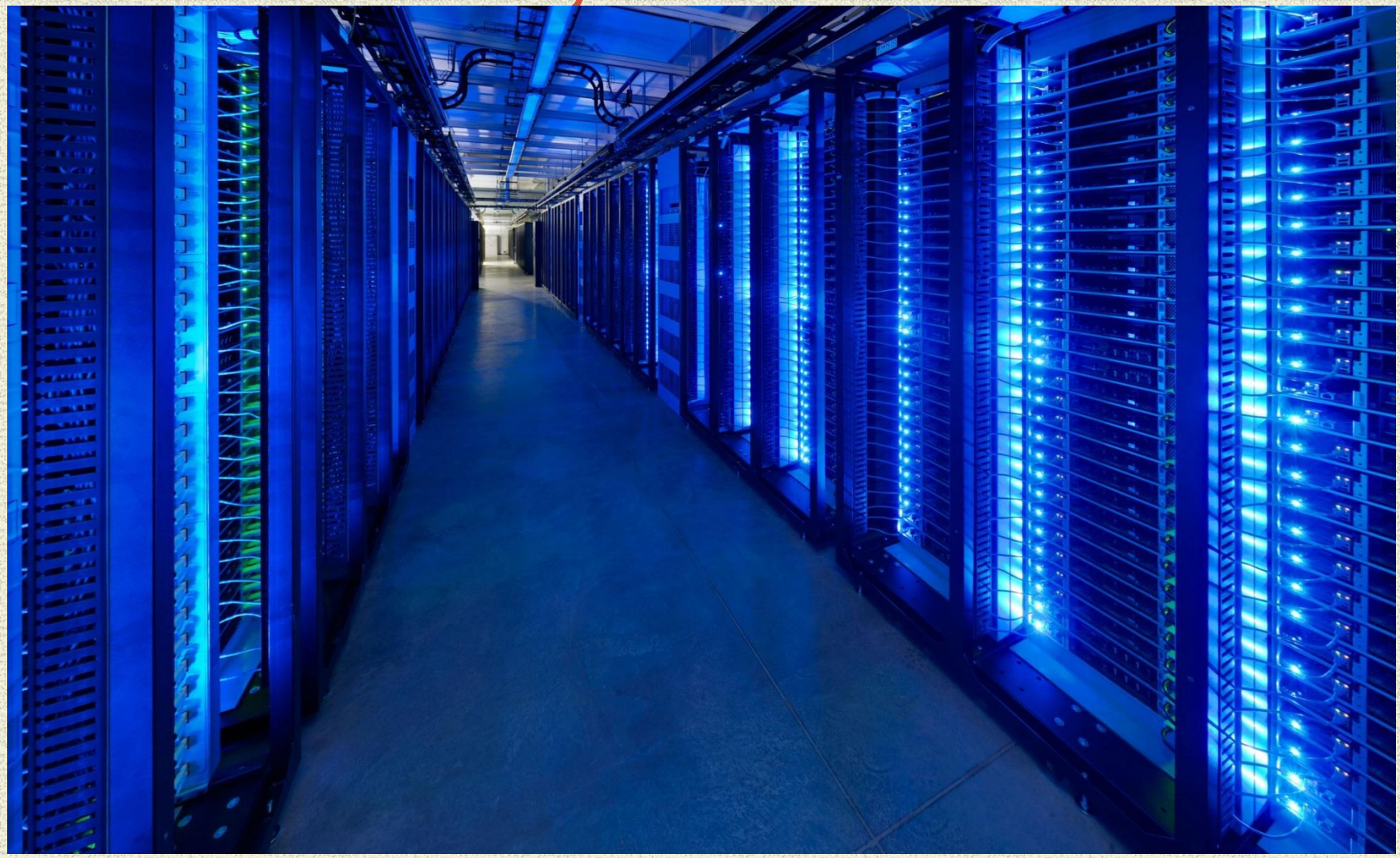
Web Server Clusters



Web Server Cache Architecture



Web server facility



massive web server facilities

Main point 1

The basic structure of a web application is a client machine sending a request to a web server, which sends a response back to the client. Major steps in the evolution of web applications include dynamic generation of responses, machine to machine communication via web services, and more dynamic and responsive interfaces through asynchronous partial page refreshing (Ajax). **Science of Consciousness:** Parallel to the evolution we have seen in the sophistication of web application architecture, our own awareness naturally evolves to more sophisticated states of consciousness, and this process is greatly accelerated through the regular experience of pure consciousness.

Main point 2

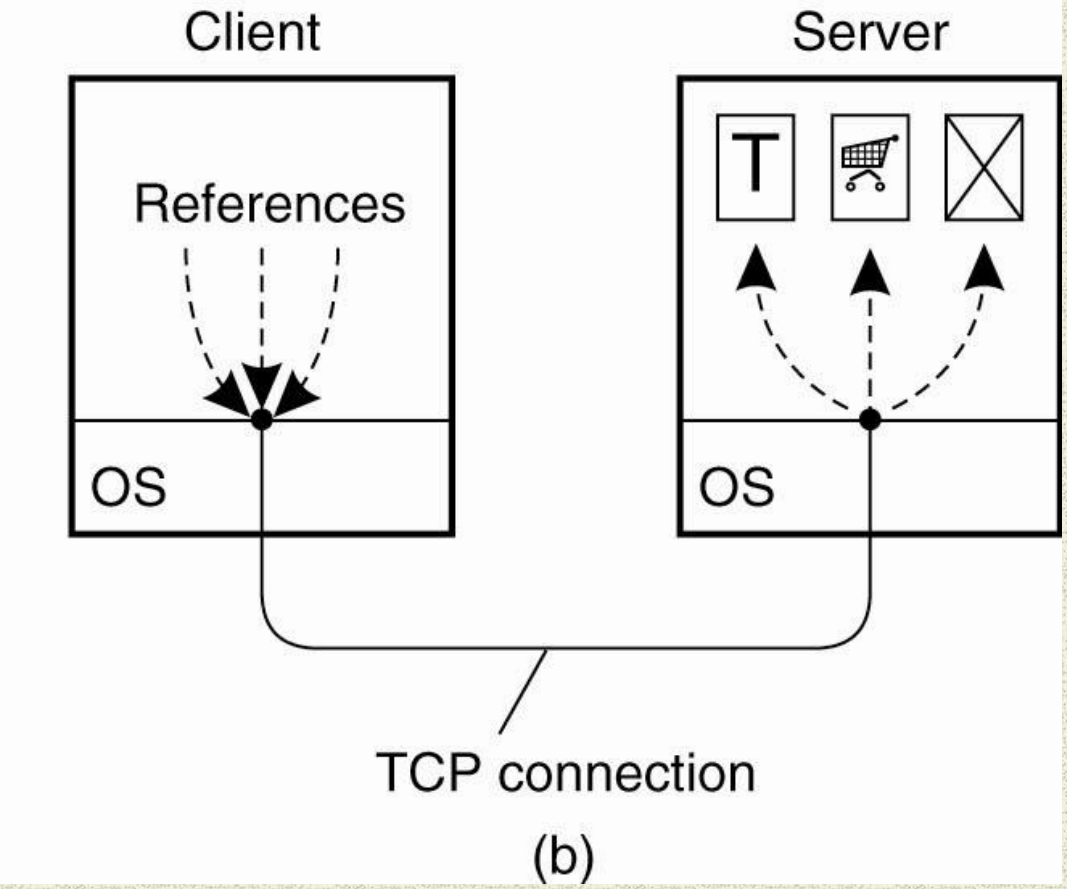
Network protocols are structured in layers. HTTP is built on top of TCP/IP. IP provides best effort delivery of messages between any two machines on the Internet. TCP provides for reliable delivery of messages. **Science of Consciousness:** The TM Technique develops an individual's ability to experience all the layers of thought, from the everyday level of thinking to the finest levels of thought and transcending thought to pure awareness. Thoughts become more effective and powerful if they are connected to more abstract and comprehensive levels of awareness, as verified by many research studies showing increased intelligence, optimization of brain functioning, and improved academic performance.

Intelligence 29 (2001): 419-440

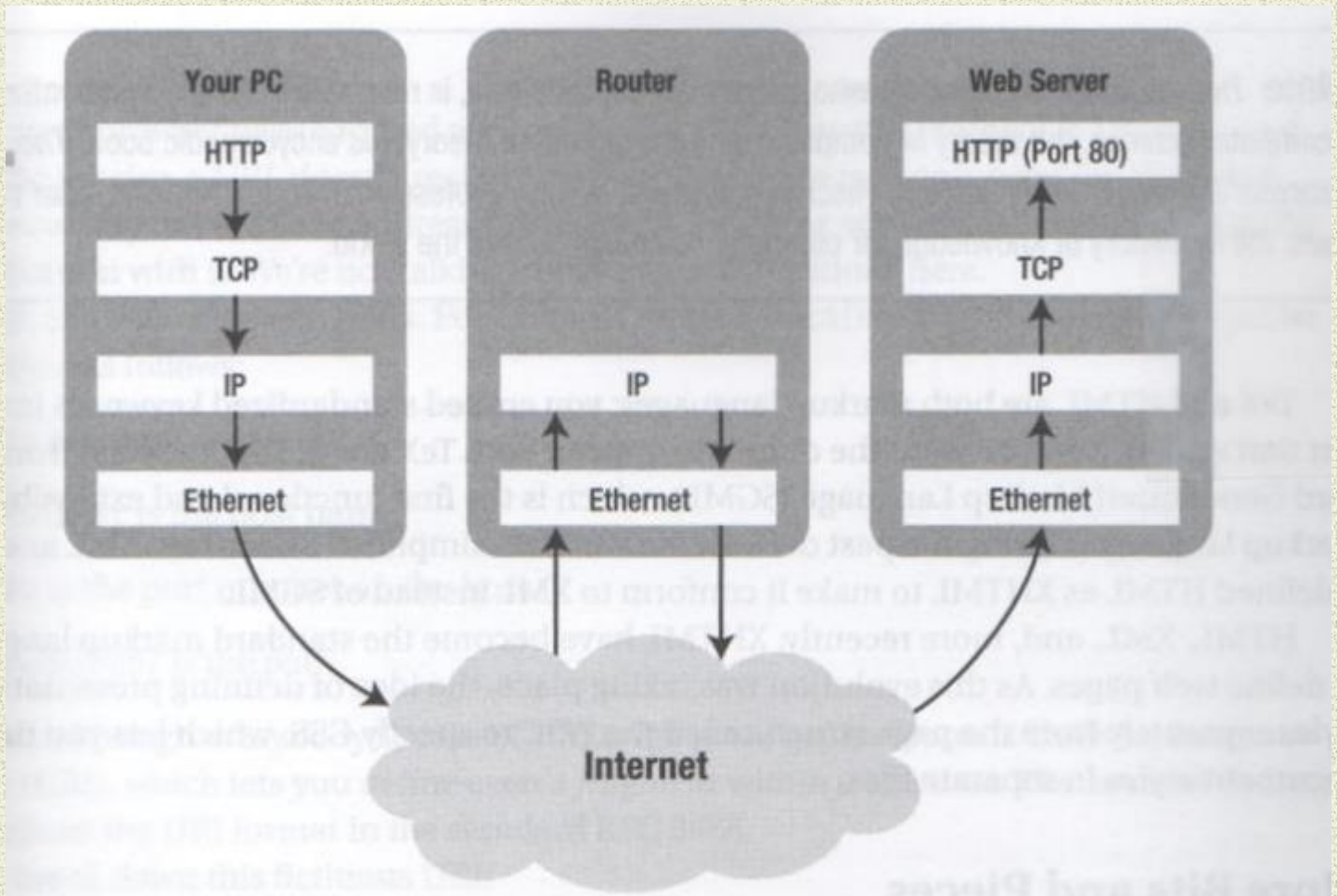
International Journal of Neuroscience 13 (1981) 211-217 and 15 (1981) 151-157.

British Journal of Educational Psychology 55 (1985): 164-166.

Internet Connectivity

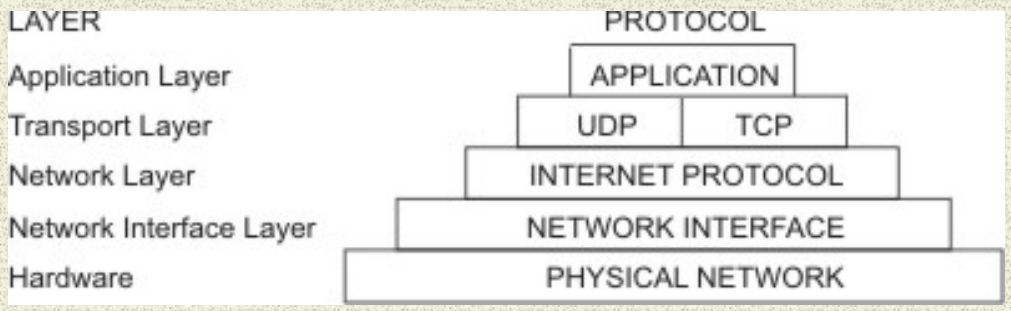


HTTP Connections



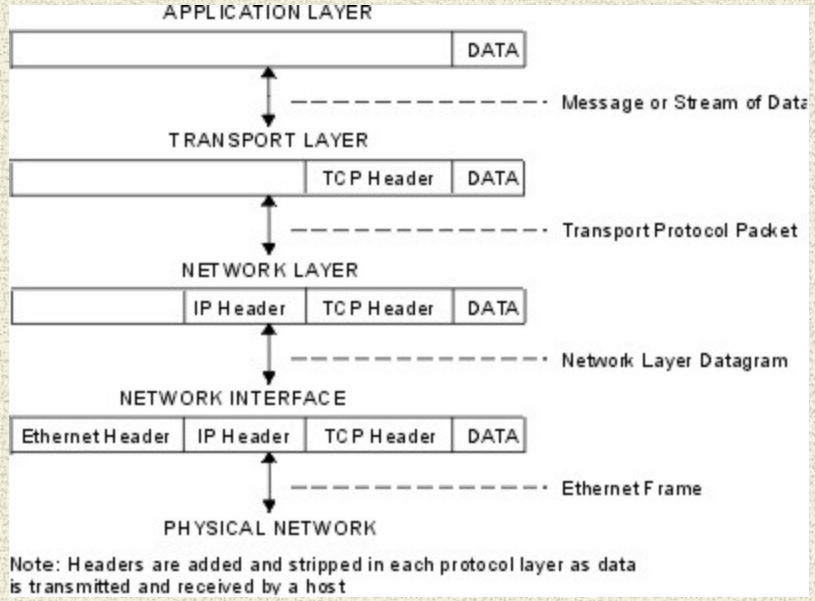
1. *Following an HTTP request through the Internet protocol stacks*

TCP/IP Suite of Protocols



Ref: http://publib.boulder.ibm.com/infocenter/aix/v6r1/index.jsp?topic=/com.ibm.aix.commadmn/doc/commadmndita/tcpip_protocols.htm

TCP/IP flow of information



Ref: http://publib.boulder.ibm.com/infocenter/aix/v6r1/index.jsp?topic=/com.ibm.aix.commadmn/doc/commadmndita/tcpip_protocols.htm

Main point 2

Network protocols are structured in layers. HTTP is built on top of TCP/IP. IP provides best effort delivery of messages between any two machines on the Internet. TCP provides for reliable delivery of messages. **Science of Consciousness:** The TM Technique develops an individual's ability to experience all the layers of thought, from the everyday level of thinking to the finest levels of thought and transcending thought to pure awareness. Thoughts become more effective and powerful if they are connected to more abstract and comprehensive levels of awareness, as verified by many research studies showing increased intelligence, optimization of brain functioning, and improved academic performance.

Intelligence 29 (2001): 419-440

International Journal of Neuroscience 13 (1981) 211-217 and 15 (1981) 151-157.

British Journal of Educational Psychology 55 (1985): 164-166.

UDP vs TCP

- What's the best thing about UDP jokes?
  - I don't care if you get them
- What's the best part about TCP jokes?
  - I get to keep telling them until you get them.

Main point 3

Successful architectures are as simple and efficient as possible. HTTP is a simple and efficient protocol that has two types of messages, requests and responses. Requests contain a method type and resource identifier. Responses contain a success code. **Science of Consciousness:** The TM Technique is a simple natural process that allows ones awareness to be as simple and efficient as possible.

HTTP: request/response protocol

A client sends request containing

- request method, URI, and protocol version
- followed by a MIME-like message containing request modifiers, client information, and possible body content

The server responds

- status line, including protocol version and a success or error code,
- followed by a MIME-like message containing server information, entity meta-information, and possible entity-body content.

Anatomy of a HTTP message

```
generic-message = start-line
                  (message-header CRLF)*
                  CRLF
                  [ message-body ]
start-line      = Request-Line | Status-Line
Request-Line    = Method SP Request-URI SP HTTP-Version CRLF
Status-Line     = HTTP-Version SP Status-Code SP Reason-Phrase CRLF
message-header  = field-name ":" [ field-value ]
field-value     = OCTET*
message-body    = entity-body |<encoded/compressed entity-body>
entity-body     = OCTET*
```

The HTTP specification is here

Examples

Request line (request message)

	URI		
Method	**Context**	**File**	**Protocol Version**
GET	/Lab01x	/Login	HTTP/1.1
POST	/Lab01x	/Login	HTTP/1.1

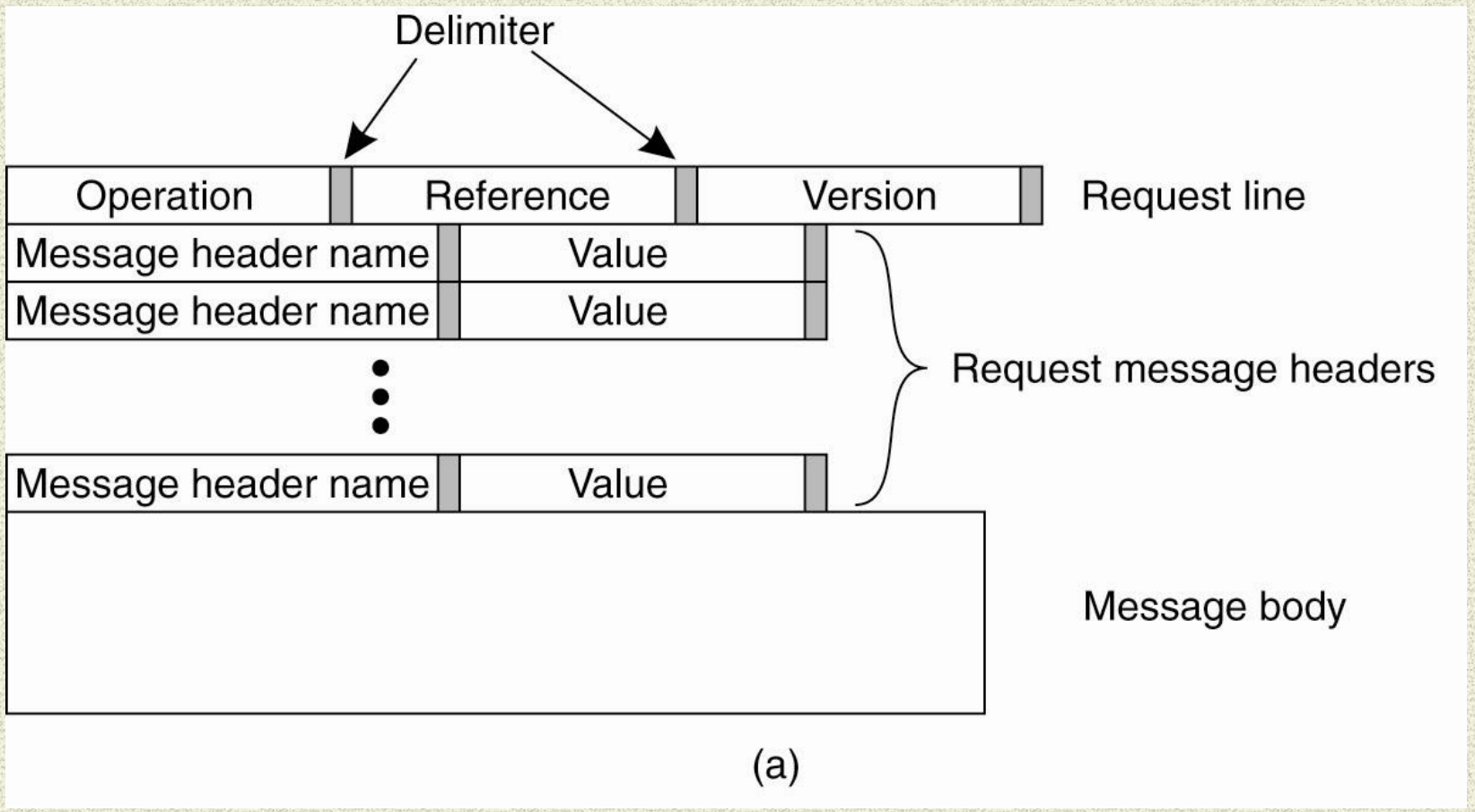
Status line (response message)

HTTP-version	**status-code**	**reason-phrase**
HTTP/1.1	200	OK
HTTP/1.1	404	/Lab01/Lab01z.html

Message header (both)

field-name	**field-value**
Accept	*/*
Accept-Language	en-us
Content-Type	text/html
Content-Length	2222

HTTP Request Message



HTTP Request Methods (operations)

Operation	Description
Head	Request to return the header of a document
Get	Request to return a document to the client
Put	Request to store a document
Post	Provide data that are to be added to a document (collection)
Delete	Request to delete a document

GET VS POST

The method name lets the server know what type of request it is, and how the rest of the message will be formatted.

GET

- Is simple, to get something back from the server.
- Limited characters in the GET request.
- Data sent in the GET is appended to the URL.
- Idempotent. Multiple requests have no side affects.

POST

- Powerful, request but purpose is to send (post) data to the server.
- The page cannot be bookmarked.
- Non-idempotent.

Uniform Resource Identifiers (URI)

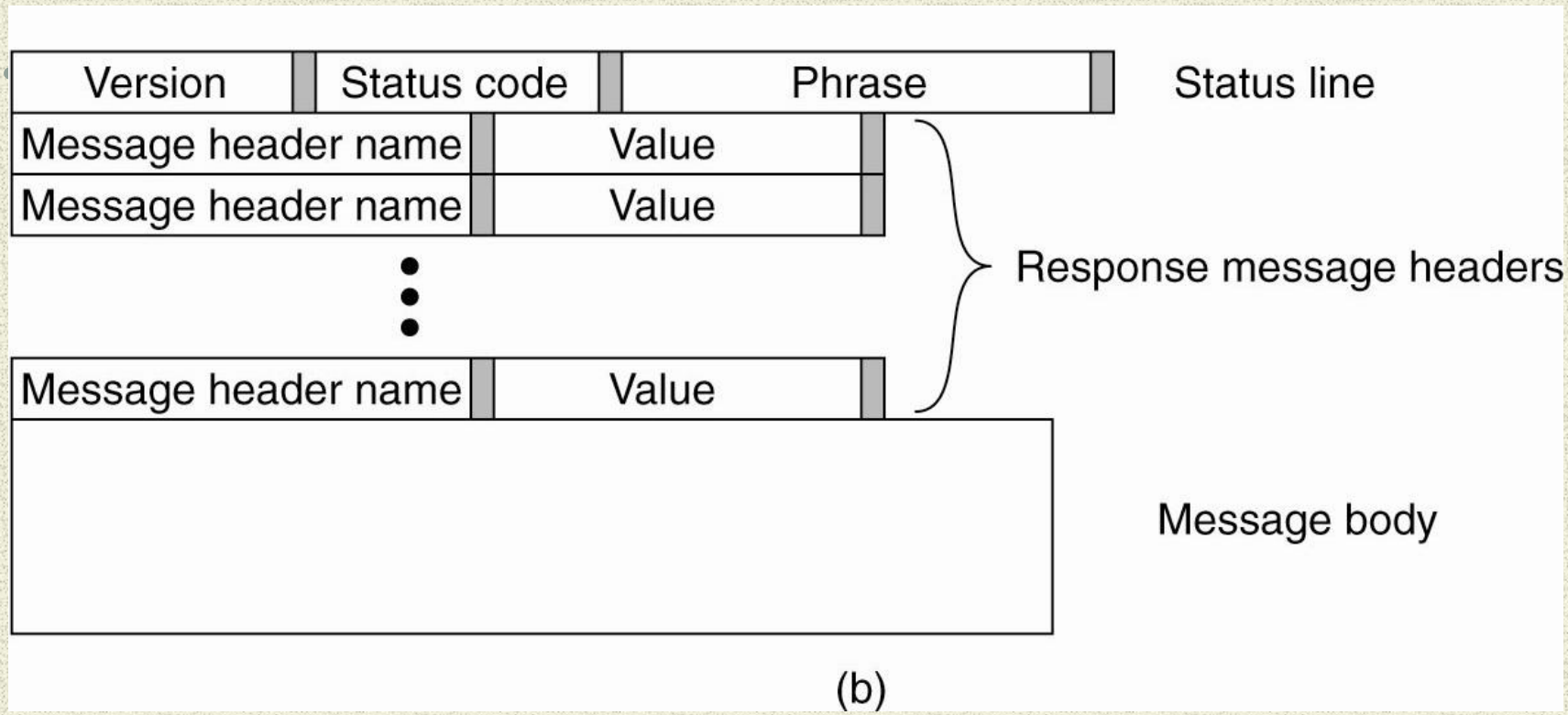
- many names: WWW addresses, Universal Document Identifiers, Universal Resource Identifiers, Uniform Resource Locators (URL) and Names (URN).

- URIs are simply formatted strings that identify a resource
  - via name, location, or any other characteristic

- http URL = "http:" "//" host [":" port] [abs_path ["?" query]]
  - If port not given, port 80 assumed.

- E.g., http://www.weather.com/weather/wxdetail/52556?dayNum=1

HTTP Response Messages



Status-Code

- 1xx: Informational - Request received, continuing process
- 2xx: Success - action successfully received, understood, and accepted
- 3xx: Redirection - Further action must be taken to complete request
- 4xx: Client Error - The request contains bad syntax or cannot be fulfilled
- 5xx: Server Error - The server failed to fulfill an apparently valid request

HTTP Headers

- HTTP 1.1 defines 46 headers, one (**Host:)** is required in requests.

- headers that describe the body.

Content-Type: MIME-type, e.g., **text/html** or **image/gif**.
Content-Length: number of bytes in the body.

- Request headers

from: email address of who's making request
User-Agent: program making request,
e.g., Netscape 3.0 sends **Mozilla/3.0Gold**

- Server headers

Server: program sending response
e.g., Apache sends **Apache/1.2b3-dev**

HTTP Headers

Header	Source	Contents
Accept	Client	The type of documents the client can handle
Accept-Charset	Client	The character sets are acceptable for the client
Accept-Encoding	Client	The document encodings the client can handle
Accept-Language	Client	The natural language the client can handle
Authorization	Client	A list of the client's credentials
WWW-Authenticate	Server	Security challenge the client should respond to
Date	Both	Date and time the message was sent
ETag	Server	The tags associated with the returned document
Expires	Server	The time for how long the response remains valid
From	Client	The client's e-mail address
Host	Client	The DNS name of the document's server

HTTP Headers

Header	Source	Contents
If-Match	Client	The tags the document should have
If-None-Match	Client	The tags the document should not have
If-Modified-Since	Client	Tells the server to return a document only if it has been modified since the specified time
If-Unmodified-Since	Client	Tells the server to return a document only if it has not been modified since the specified time
Last-Modified	Server	The time the returned document was last modified
Location	Server	A document reference to which the client should redirect its request
Referer	Client	Refers to client's most recently requested document
Upgrade	Both	The application protocol the sender wants to switch to
Warning	Both	Information about the status of the data in the message

MIME types and common subtypes

- *Multipurpose Internet Mail Extensions*
- Found in Content-Type and Accept headers
- provide open and extensible data typing
- Accept: video/mpeg

Type	Subtype	Description
Text	Plain	Unformatted text
	HTML	Text including HTML markup commands
	XML	Text including XML markup commands
Image	GIF	Still image in GIF format
	JPEG	Still image in JPEG format
Audio	Basic	Audio, 8-bit PCM sampled at 8000 Hz
	Tone	A specific audible tone
Video	MPEG	Movie in MPEG format
	Pointer	Representation of a pointer device for presentations
Application	Octet-stream	An uninterpreted byte sequence
	Postscript	A printable document in Postscript
	PDF	A printable document in PDF
Multipart	Mixed	Independent parts in the specified order
	Parallel	Parts must be viewed simultaneously

Main point 3

Successful architectures are as simple and efficient as possible. HTTP is a simple and efficient protocol that has two types of messages, requests and responses. Requests contain a method type and resource identifier. Responses contain a success code. **Science of Consciousness:** The TM Technique is a simple natural process that allows ones awareness to be as simple and efficient as possible.

Main point 4

Sockets are an important architectural mechanism in web application clients and servers. Sockets implement an API to the TCP/IP protocol. A Socket is an abstraction that hides the details of the lower level network connections. **Science of Consciousness:** The TM Technique cultures the ability to maintain awareness and think at all levels of thought, from the everyday thinking level to the most abstract level of thought, pure awareness.

Sample HTTP Exchange

- To retrieve the file at URL http://www.somehost.com/path/file.html
- first open a socket to the host **www.somehost.com**, port 80 (use the default port of 80 because none is specified in the URL).
- Then, send something like the following through the socket:

  GET /path/file.html HTTP/1.0
  From: someuser@jmarshall.com
  User-Agent: HTTPTool/1.0
  [blank line here]

- The server should respond with something like, via same socket:

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

  <html> <body> <h1>Welcome to WAArF!</h1> blah blah blah . . . </body> </html>

Sockets

- Network connection endpoints.
  - Enable applications to read from and write to the network.
  - API for TCP (or UDP or IP)

- “client” sockets make a connection to a server
  - public Socket(hostName, portNumber);
  - Creates a socket and connects it to the specified port number on the named host.
  -

- servers stand by waiting for client requests
  - public ServerSocket(portNumber);
  - creates a server socket, bound to the specified port on this server
  - server socket blocks (on accept), waiting for a connection request.
  - creates a Socket instance to handle the communication with the client.

Ports

- What is a port?
- 16 bit number that identifies a program on the receiver end.
- TCP port numbers 0-1023 are reserved.
- HTTP port is 80.
- FTP?
- SMTP?
- HTTPS?
- POP3?

Minimal Web Client

- Define variables
  - String hostName = “yahoo.com”;
  - int portNumber = 80;
  - String defaultPage = “”;
  - boolean autoflush = true;

- Create socket

```
Socket socket = new Socket(hostName, portNumber);
```

- Create stream writer (to send requests)

```
PrintWriter out = new PrintWriter(socket.getOutputStream(), autoflush);
```

- Send request

```
out.println("GET /"+defaultPage+" HTTP/1.1");
out.println("Host: yahoo.com:80");
out.println("Connection: Close");
out.println();
```

Minimal Web Server

- Define variables
  - int port = 80;
- Create a ServerSocket
  - ServerSocket server = new ServerSocket(port);
- Wait for incoming requests
  - Socket socket = server.accept();
- When a request comes in:
  - Read the request from the socket.
  - Write the response to the socket.

Socket Programming for a simple web server

- **Demo Objectives:** This demo looks at a very simple web server. A web server is a program that is responsible for accepting HTTP requests from *clients*, and serving back HTTP responses along with optional data contents, which usually are web pages such as HTML documents and linked objects (images, etc.). Common examples of web servers include Apache, Microsoft IIS, Google GWS, etc – we will be using Sun's open source Glassfish webserver functionality; Glassfish is also a "web container" (i.e., a server program that contains and supports servlets and JSPs/Facelets), and even a full-fledged application server that supports all of Java EE6.
- Gain direct hands-on experience working with the HTTP protocol
- See how a simple webserver is built on top of a socket network connection
- See how to work with the Netbeans IDE
- Use the Netbeans debugger to inspect the operation of the web server. In particular, when "inspecting" watch for
  - Server started and listening
  - Accepted connection
  - Received following – look for the HTTP request message components
- The demo is a good introduction to the afternoon's lab, which is basically a repeat of the demo that adds use of a packet sniffer to see the HTTP messages that are received and sent

class WebServer

```
{

    /* static class data/methods */
    static File root; // the web server's virtual root
    static Scanner sc;
    public static void main(String[] a)
    {
        try
        {
            sc = new Scanner(System.in);
            root = new File(System.getProperty("user.dir"));
            int port = 8090;

            ServerSocket ss = new ServerSocket(port);
            System.out.println(String.format("Web server started. Listening on port %d\r\n", port));
            while (true) // listen for requests on port 8090
            {
                Socket s = ss.accept();
                System.out.println("Accepted connection");
                handleHTTPRequest(s);  // process the request
            }
        } catch (Throwable ex)
        {
            System.out.println(ex.getMessage());
        }
    }
```

```
// process GET or POST request from a browser
   public static void handleHTTPRequest(Socket s) throws Exception
   {   byte[] b = null;
       FileInputStream fis = null;
       try
       {
           InputStream is = new BufferedInputStream(s.getInputStream());
           s.setSoTimeout(2000);                    // block on read for this many milliseconds

           StringBuilder sb = new StringBuilder();
           String line = null;
           int contentLength = -1;
           do
           {
              line = readLine(is);

              if (line.toLowerCase().startsWith("content-length"))
              {
                 // Remember length of the content.
                 Pattern p = Pattern.compile("\\d+$");
                 Matcher m = p.matcher(line);
                 if (m.find())
                 {
                    contentLength = Integer.parseInt(m.group());
                 }
              }
              sb.append(String.format("%s\r\n", line));
           } while (line.length() > 0);
```

```
    if (contentLength != -1)
    {
        b = new byte[contentLength];
        is.read(b);
        sb.append(new String(b, "UTF-8"));
    }

    System.out.format("The browser sent the following request\r\n%s\r\nEnter name of file with your HTTP response\r\n> ",
                      sb.toString());

    String fileName = sc.next();
    fis = new FileInputStream(fileName);
    int len = fis.available();
    b = new byte[len];
    fis.read(b);
    OutputStream os = s.getOutputStream();
    os.write(b);
} catch (Exception ex)
```

Main point 4

Sockets are an important architectural mechanism in web application clients and servers. Sockets implement an API to the TCP/IP protocol. A Socket is an abstraction that hides the details of the lower level network connections. **Science of Consciousness:** The TM Technique cultures the ability to maintain awareness and think at all levels of thought, from the everyday thinking level to the most abstract level of thought, pure awareness.