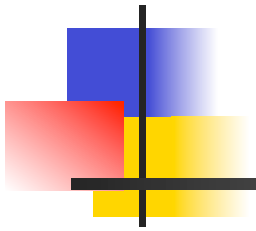


InterProcess Communication

Sockets and Port

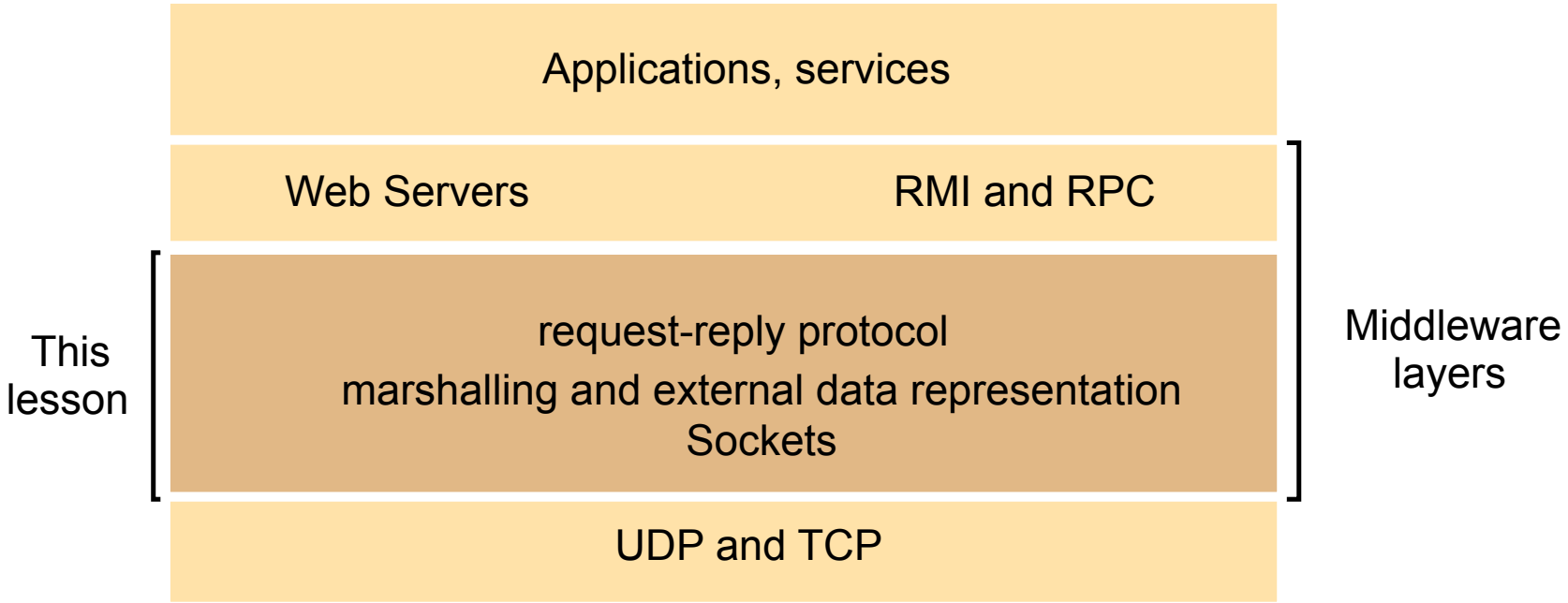
Java Sockets

Marshalling/Unmarshalling data

Request-Reply protocol

HTTP protocol

Middleware layers

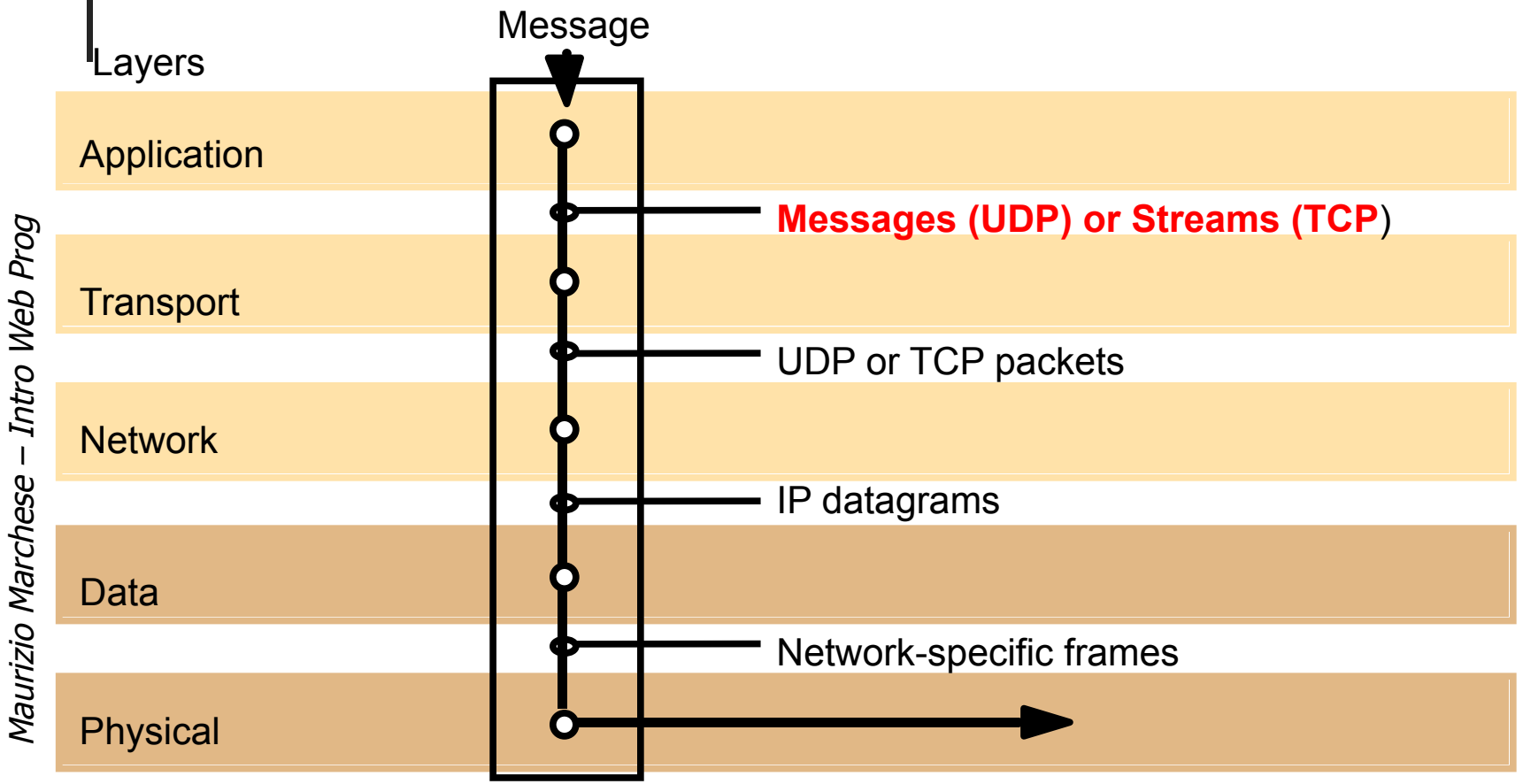


Clients and Servers

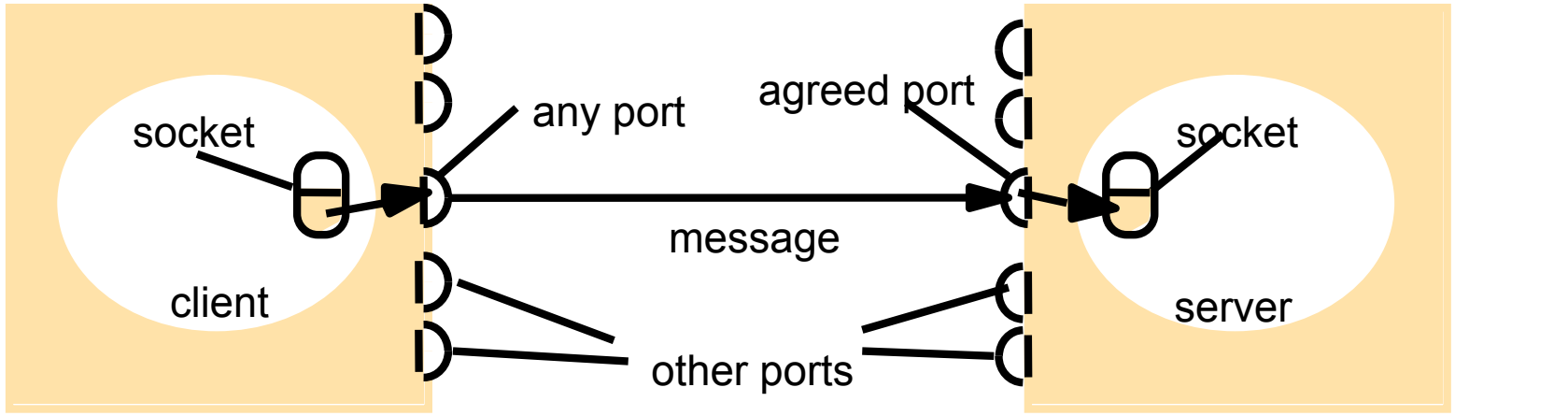
- Quando si scrivono network applications, è comune parlare di clients e servers. La distinzione è piuttosto vaga, ma sostanzialmente possiamo pensare che :
  - chi inizia la conversazione è il *client*.
  - chi accetta la richiesta di parlare e' il *server*.

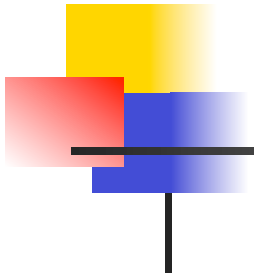
- Per i nostri scopi, la più importante differenza tra client e server è che
  - il client può in qualunque istante creare un canale di comunizazione di rete (per es. socket) per iniziare una conversazione con una server application,
  - mentre un server si deve preparare ad ascoltare in anticipo per possibili conversazioni in arrivo

TCP/IP layers



Sockets and ports



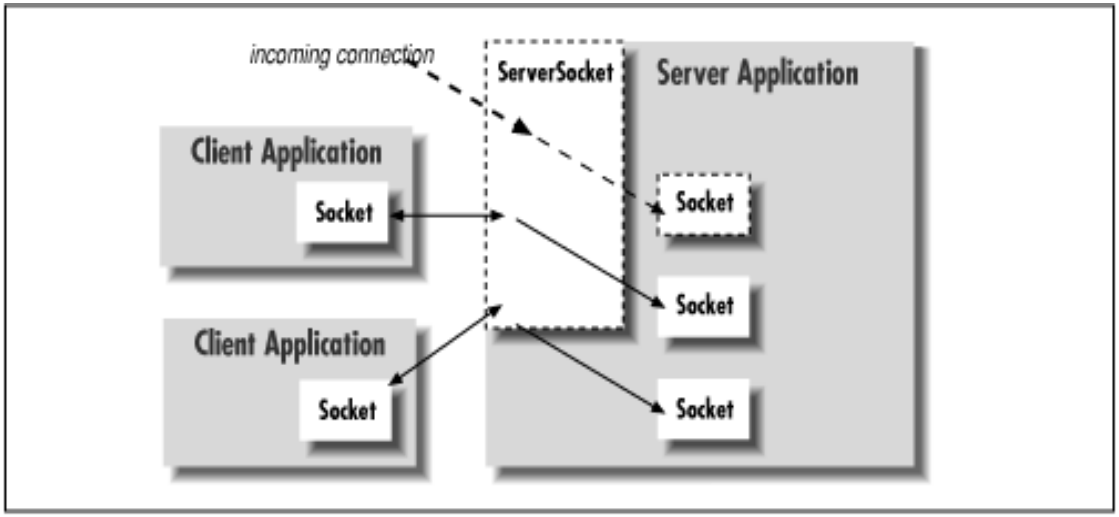


Java Sockets

- Java supporta un accesso semplificato e object-oriented alle sockets.
  - Questo rende la network comunicazione di rete sensibilmente più semplice.
  - Parlare ad un altra applicazione (locale o remota) è semplice come leggere un file o ottenere un input da un utente.
- La java.net.Socket class rappresenta un singolo lato di una connessione socket indifferentemente su un client o su un server.

Java Sockets

- Un server usa la java.net.ServerSocket class per attendere connessioni da clients. Un applicazione agente come server crea un ServerSocket object e attende, bloccato in una chiamata al suo metodo accept(), finché non giunge una connessione. A quel punto, il metodo accept() crea un Socket object che il server usa per comunicare con il client.
- Un server mantiene molte conversazioni simultaneamente. C' e' una sola ServerSocket, mentre c' e un oggetto Socket attivo per ogni cliente.



Server port

- Una applicazione server ascolta su una porta predefinita mentre attende una connessione

```
ServerSocket listenSocket =
        new ServerSocket(7896)
```

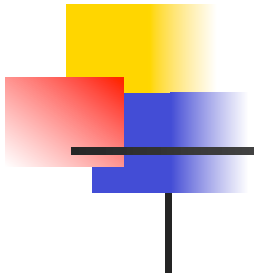
- Un client ha bisogno di due informazioni per connettersi a un server su Internet:
  - un nome di host (per recuperare l'indirizzo del server)
  - un numero di porta (per individuare il processo sulla macchina server).

```
int serverPort = 7896;
s = new Socket("127.0.0.1", serverPort );
```

- I clients selezionano il numero di porta corrispondente al servizio desiderato.
  - I numeri di porta sono codificati nelle RFC (Es. Telnet 23, FTP 21, ecc.), ma possono anche essere scelti (quasi) arbitrariamente per applicazioni custom (ca. 2^{16} → ca. 65000)

Client port

- Il numero di porta del client è tipicamente assegnato dal sistema operativo:
  - la scelta di tale numero non è di solito rilevante.
- Quando una client socket manda un pacchetto a una server socket, il pacchetto include la specifica della porta e dell' indirizzo del client:
  - così il server è in grado di rispondere.



Sockets protocols

- Usando le sockets, si può decidere che tipo di protocollo si desidera per il trasporto di pacchetti sulla rete:
  - Un protocollo connection-oriented (TCP) o
  - Un protocollo connectionless (UDP).

Connection-oriented protocols

- Un protocollo connection-oriented offre l'equivalente di una conversazione telefonica.
  - Dopo aver stabilito la connessione, due applicazioni possono scambiarsi dati.
  - La connessione rimane in essere anche se nessuno parla.
- Il protocollo garantisce che non vengano persi dati e che questi arrivino sempre nell'ordine corretto.
- La classe Java Socket usa un protocollo *connection-oriented*, e "parla" TCP

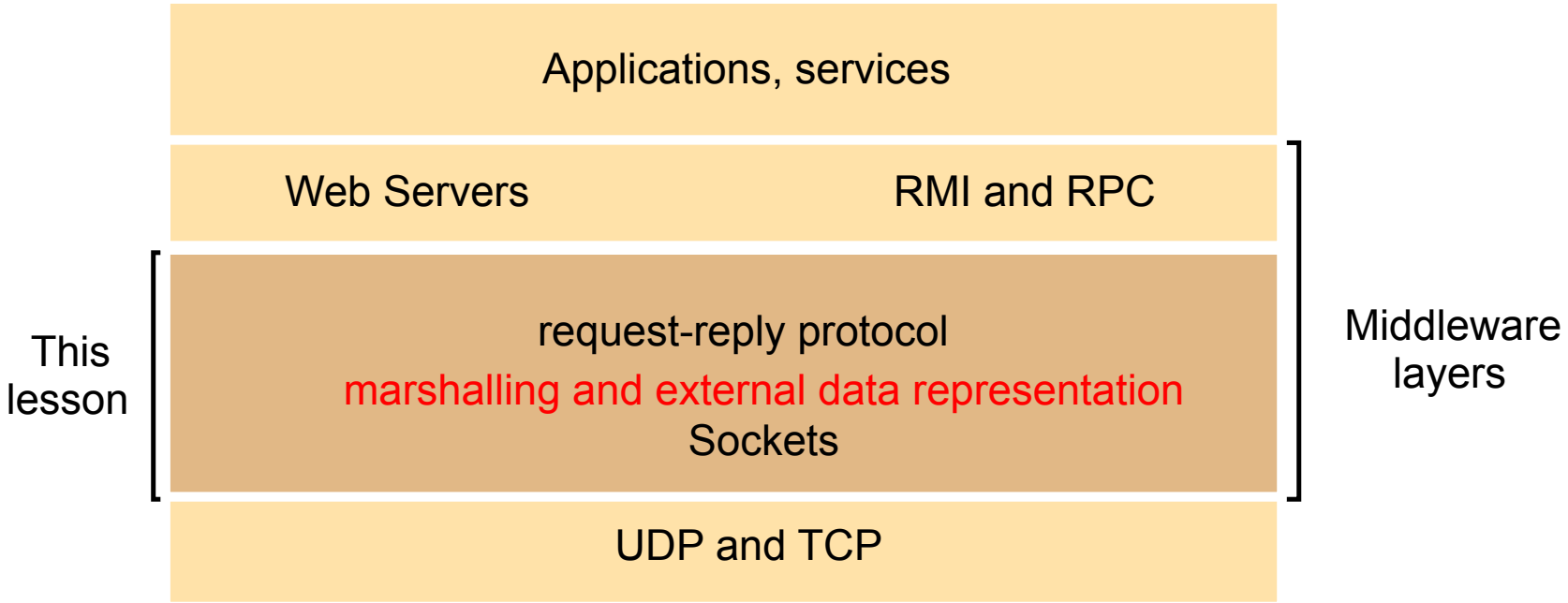
Connection oriented Protocols

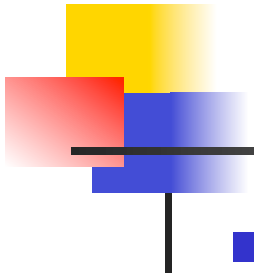
- Con un connection-oriented protocol, una client socket stabilisce, alla sua creazione, una connessione con una server socket. Stabilita la connessione, un protocollo connection-oriented assicura la consegna affidabile dei dati, ovvero:
  - per ogni pacchetto spedito, il pacchetto viene consegnato.
  - ad ogni spedizione, la socket si aspetta di ricevere un acknowledgement che il pacchetto è stato ricevuto con sucesso.
- Se la socket non riceve l'acknowledgement entro un tempo prestabilito, la socket ri-invia il pacchetto. La socket continua a provare finché la trasmissione ha successo, o finché decide che la consegna è impossibile.
  - Ogni pacchetto IP ha un identificatore; il ricevente può quindi riordinare i pacchetti e/o rifiutare duplicati

Connectionless Protocols

- Un protocollo connectionless somiglia al servizio postale.
- Le applicazioni possono inviarsi brevi messaggi, ma non viene tenuta aperta una connessione tra un messaggio e l'altro.
- Il protocollo NON garantisce che non vadano persi dati ne' che questi arrivino nell'ordine corretto.
- La classe Java DatagramSocket usa un protocollo *connectionless*, e "parla" UDP

Middleware layers



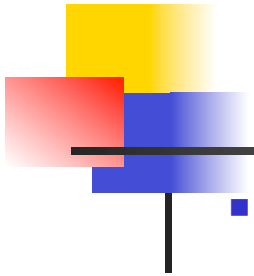


Some Problems

- How to transfer data in different hosts
  - external data representation
  - marshalling data
  - www → html
- Define a shared communication protocol
  - request-reply protocol
  - www → http

External Data Representation

- Information stores in running programs → data structures/objects
- Information in messages → sequences of bytes
- Main Methods:
  - values are converted to an a-priori agreed format and converted back on receipt (marshalling and unmarshalling of data)
  - values are transmitted in the sender's format, together with information of the format



Approaches

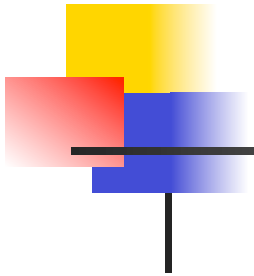
- String / ASCII / UNICODE
- COM/DCOM/COM+
  - Microsoft
  - Uniform Data Transfer
- CORBA: Common Object Request Broker Architecture
  - OMG specifications (consortium)
  - Multi languages
- Java's object serialization
  - SUN
  - only Java
- Web Services
  - XML-based Open standards: XML-RPC, RESTful Web Services
  - SOAP: Simple Object Access Protocol
  - Multi languages

CORBA Common Data Representation

- Primitive types:
  - 15 primitive types [short,long, unsigned long(4 bytes) ,float, doubel, char, boolean….]
- Constructed types
  - note that it does not deal with objects

Type	*Representation*
sequence	length (unsigned long) followed by elements in order
string	length (unsigned long) followed by characters in order (can also can have wide characters)
array	array elements in order (no length specified because it is fixed)
struct	in the order of declaration of the components
enumerated	unsigned long (the values are specified by the order declared)
union	type tag followed by the selected member

Java serialized form



Serialized values

Person	8-byte version number		h0
3	int year	java.lang.String name:	java.lang.String place:
1934	5 Smith	6 London	h1

Explanation

class name, version number

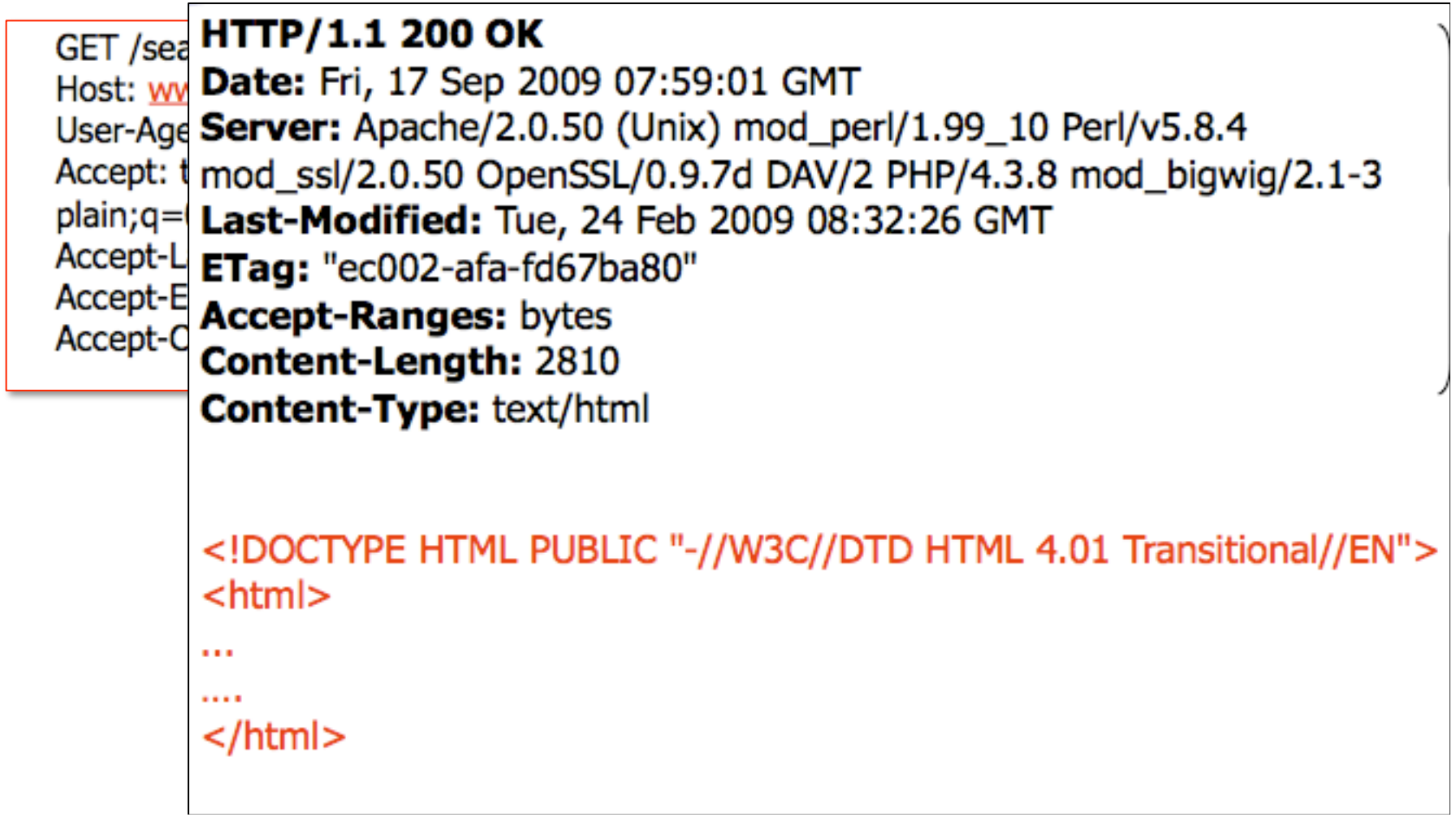
number, type and name of instance variables

values of instance variables

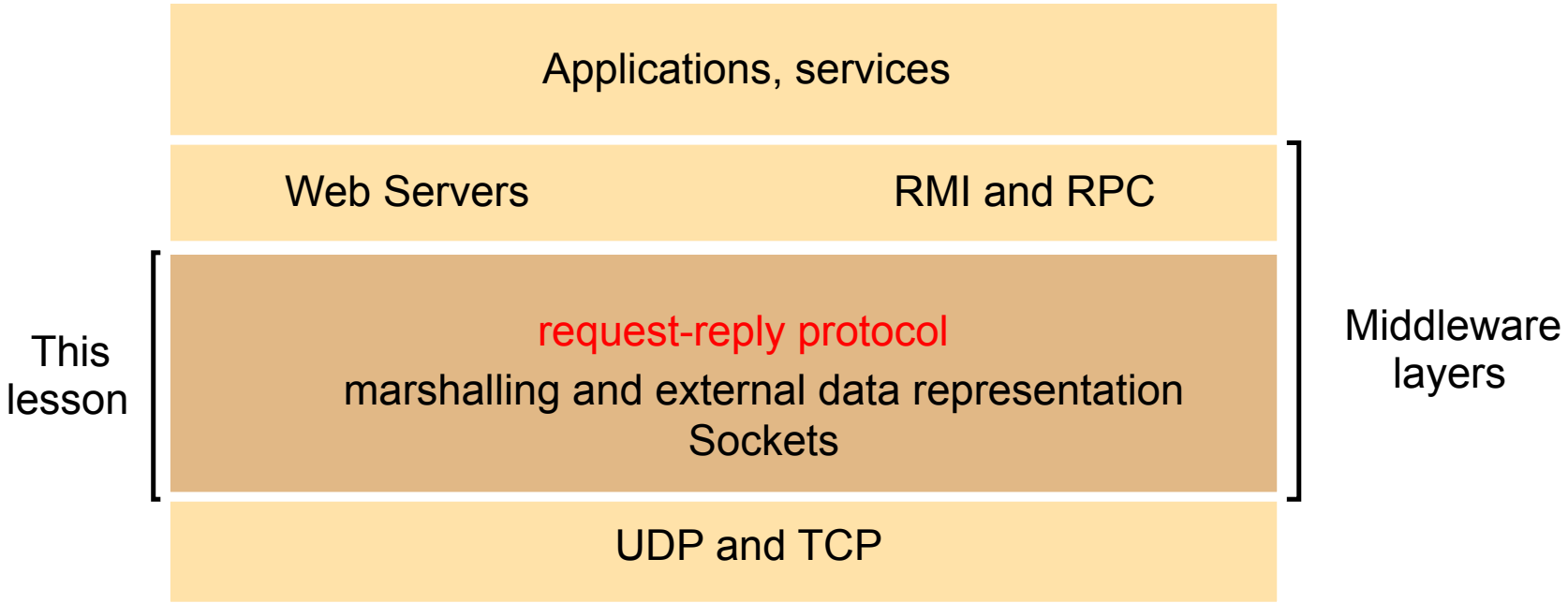
- h0 and h1 are handles (references for internal objects: i.e. String,..)
- deep copy of both public and private variable
- use of transient keyword to avoid serialization of specific fields
- reflection: ability to enquire class properties and create classes from names + default constructor

World Wide Web

- HTML - XML

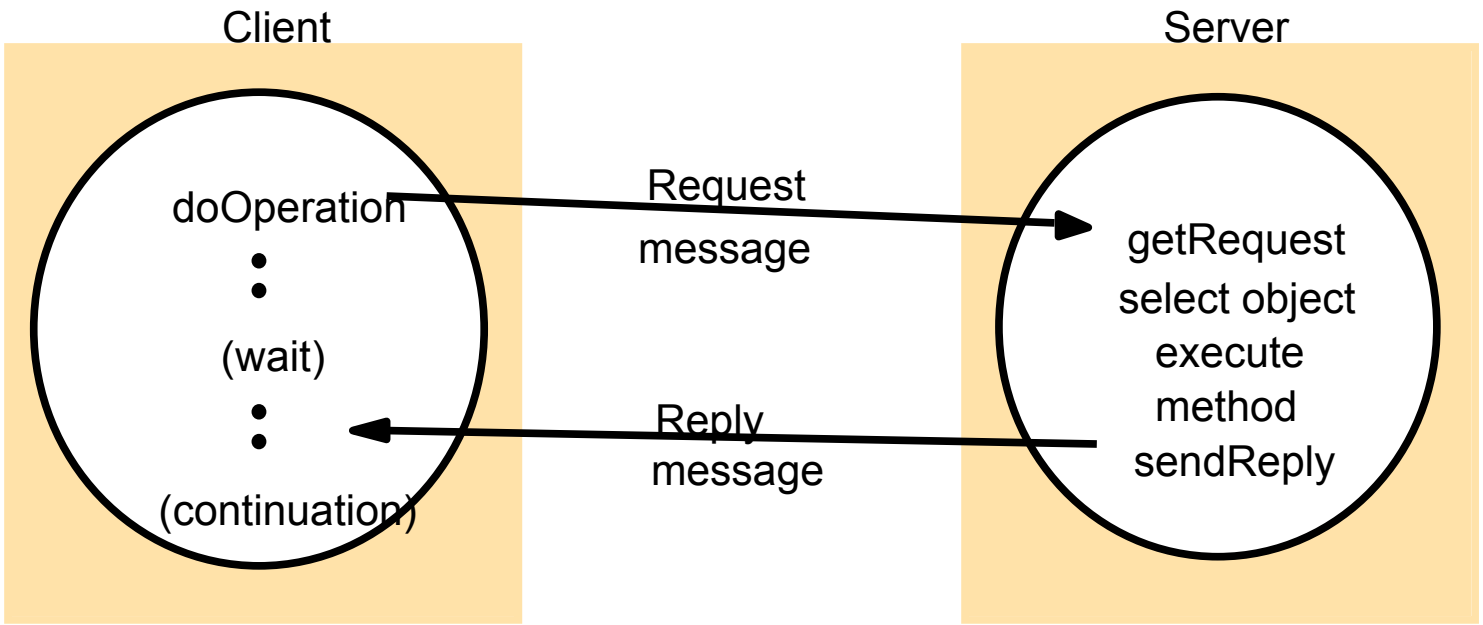


Middleware layers

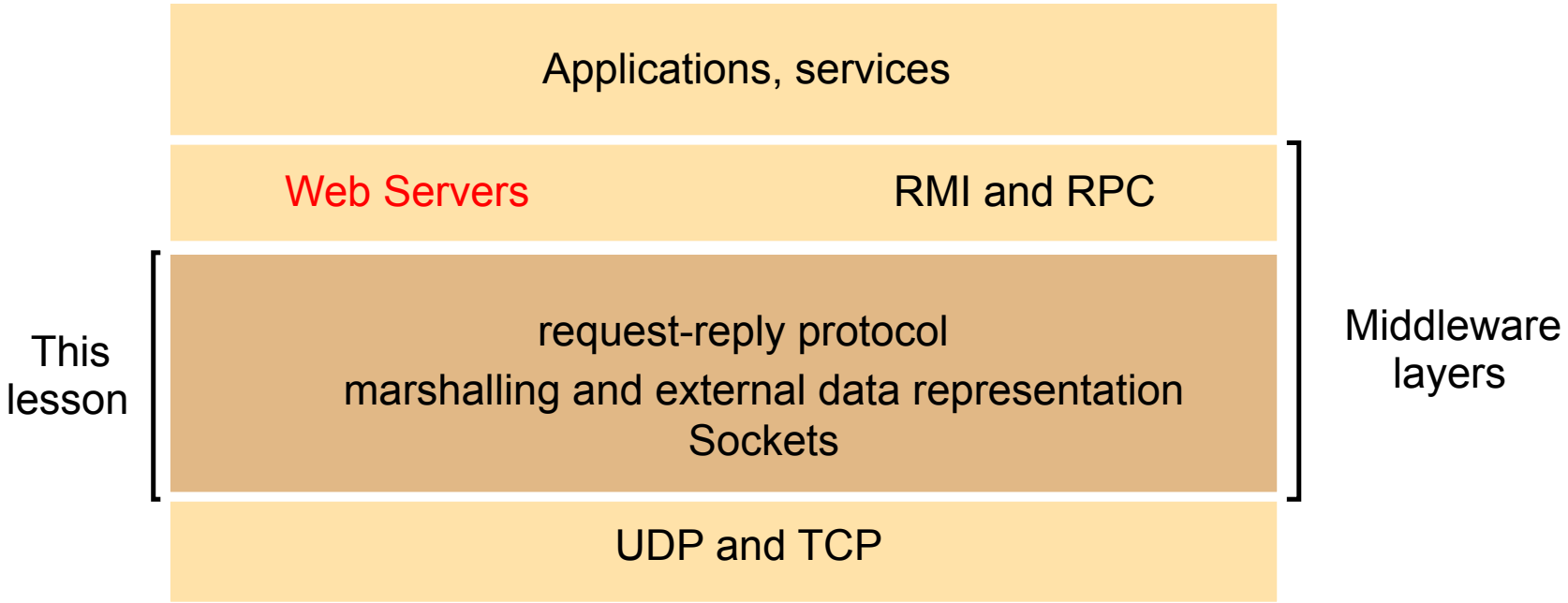


Request-reply communication

- The majority of communication protocols (RPC/ RMI/HTTP..) support such communication schema



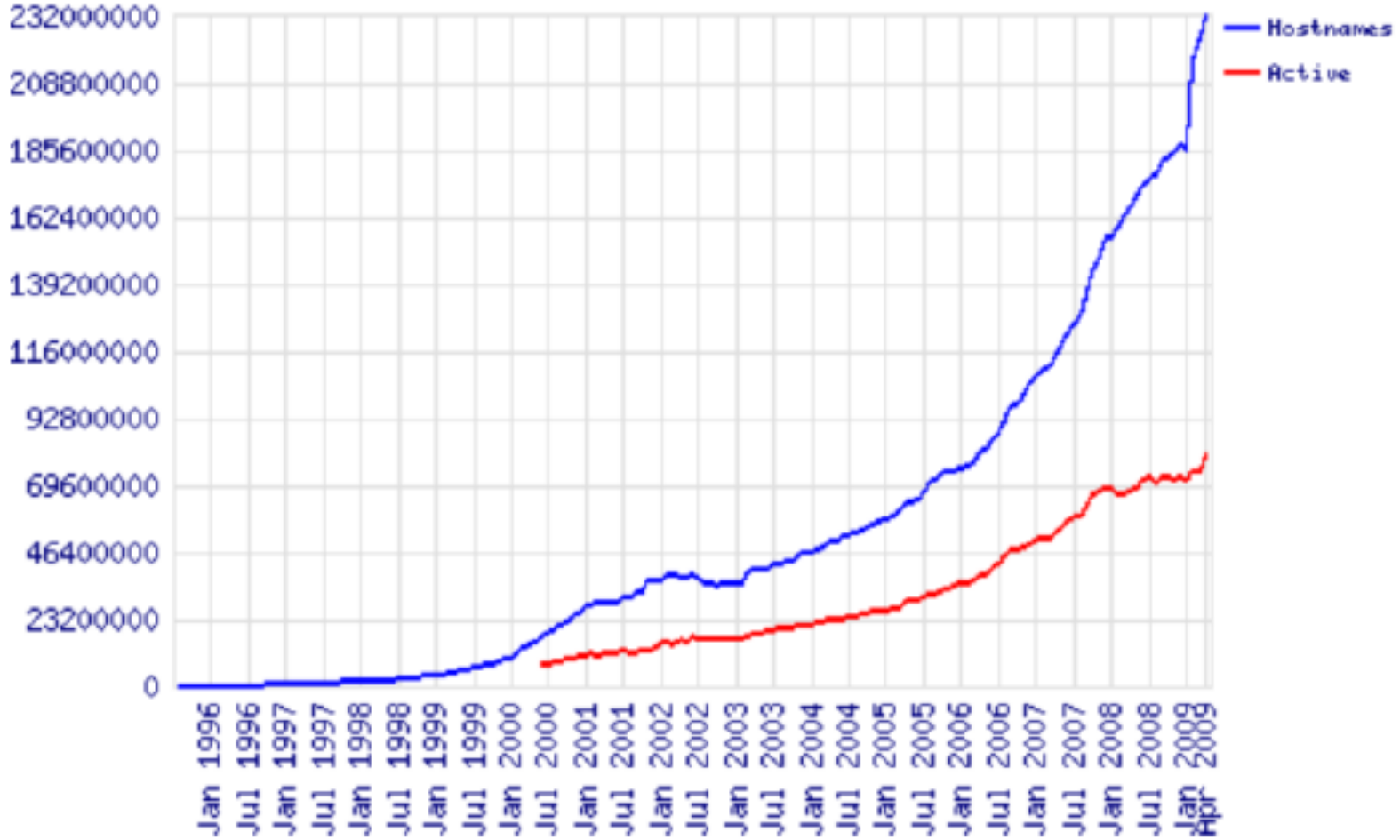
Middleware layers



Brief History of HTTP

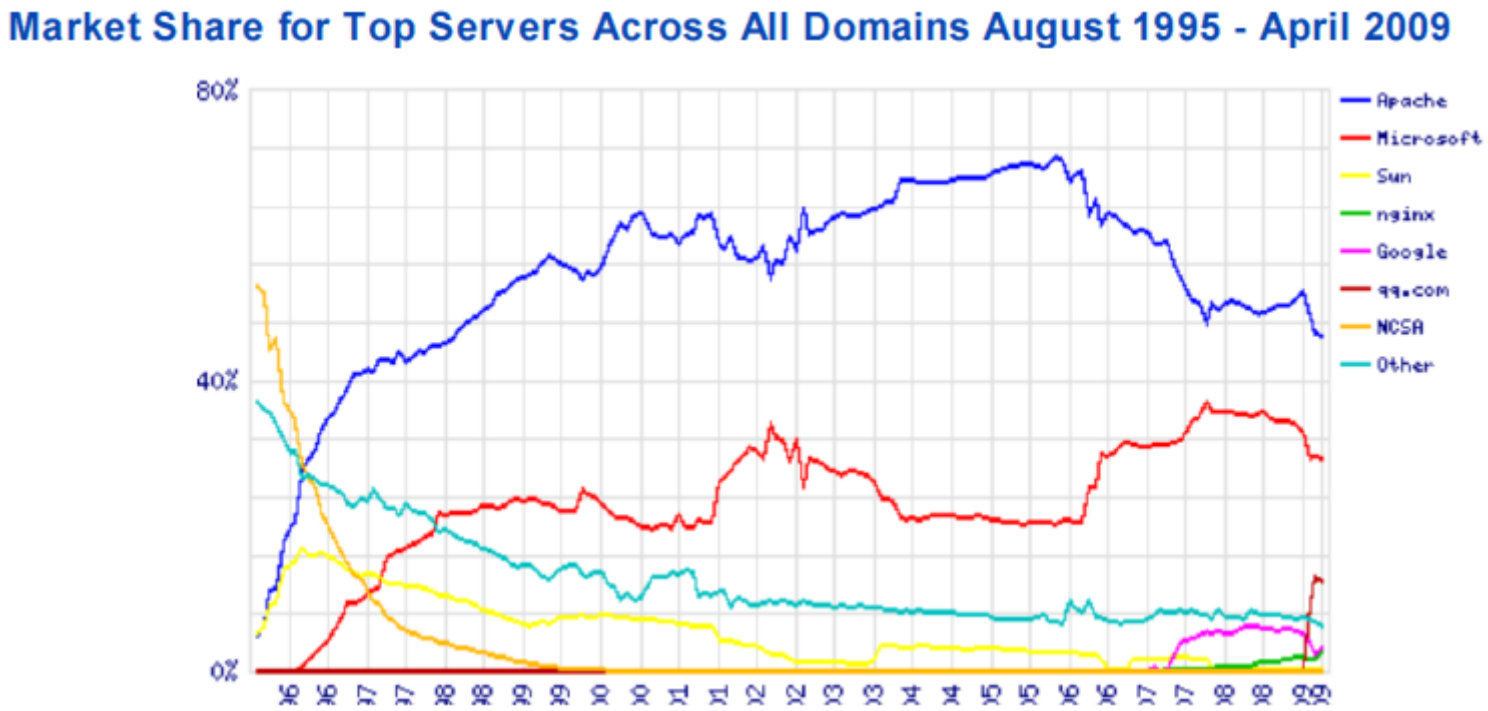
- 1990, the Http protocol was proposed by Tim Bernes Lee and was used at CERN
- Summer of 1991, the http protocol was released to the public.
- 1993: Marc Andreessen (then a grad student at NSCA) posts Mosaic on an ftp site. New features include:
  - Hyperlinks
  - Embedded images
- December 1993: Mosaic growth makes the front page of New York Times
- 1994: Marc Andreessen and colleagues leave NSCA to form Mosaic Corp. (later renamed "Netscape")

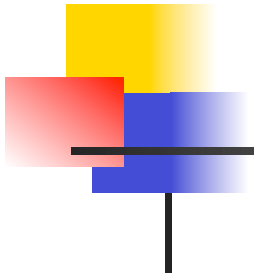
Web Growth


232000000
208800000
185600000
162400000
139200000
116000000
92800000
69600000
46400000
23200000
0
Hostnames
Active
Jan 1996
Jul 1996
Jan 1997
Jul 1997
Jan 1998
Jul 1998
Jan 1999
Jul 1999
Jan 2000
Jul 2000
Jan 2001
Jul 2001
Jan 2002
Jul 2002
Jan 2003
Jul 2003
Jan 2004
Jul 2004
Jan 2005
Jul 2005
Jan 2006
Jul 2006
Jan 2007
Jul 2007
Jan 2008
Jul 2008
Jan 2009
Apr 2009

Web Server Statistics

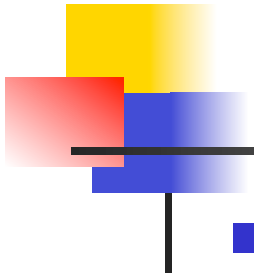
- Apache is the most popular web server today (freely available)
- Microsoft IIS is gaining ground





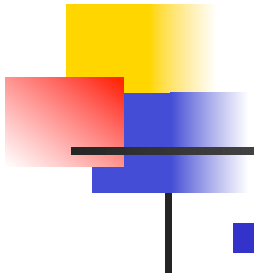
Versions of HTTP

- Early protocol is HTTP 0.9
  - read only
- More recent versions:
  - HTTP 1.0
    - read, input, delete, ...
  - HTTP 1.1
    - performance optimizations



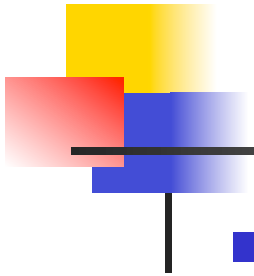
HTTP Overview

- Client (browser) sends HTTP request to server
- Request specifies affected *URL*
- Request specifies desired *operation*
- Server performs *operation* on *URL*
- Server sends response
- Request and reply headers are in pure text



Static Content and HTML

- Most static web content is written in HTML
- HTML allows
  - Text formatting commands
  - Embedded objects
  - Links to other objects
- Server do not need to understand or interpret HTML; Browsers do

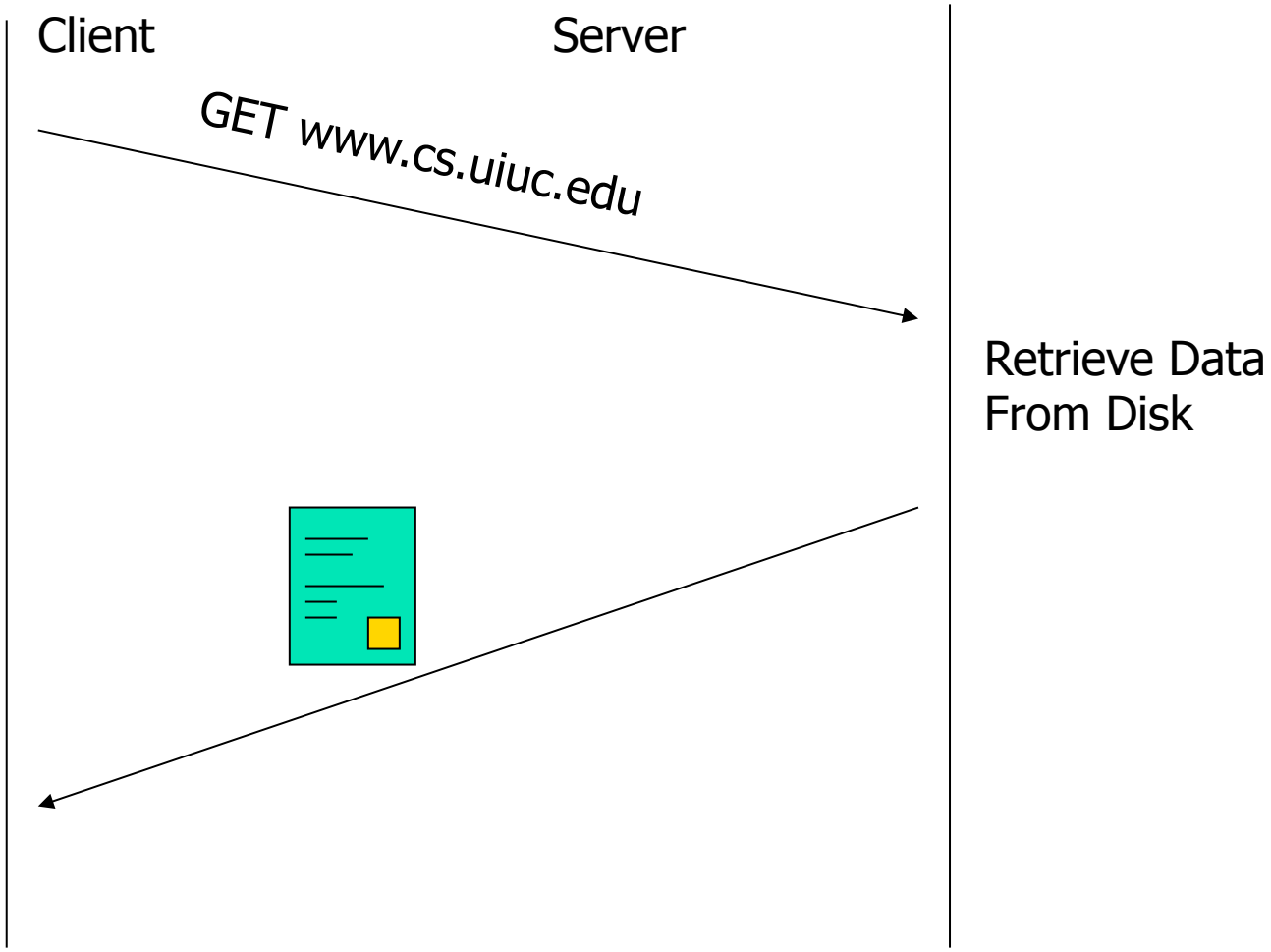


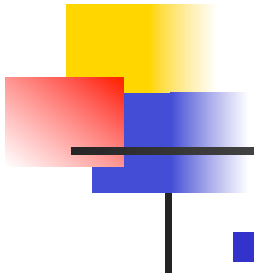
HTTP Request Operations

- GET: retrieves URL (most widely used)
- HEAD: retrieves only response header
- POST: posts data to server
- PUT: puts page on server
- DELETE: deletes page from server

more later…

Example of an HTTP 1.0 Exchange

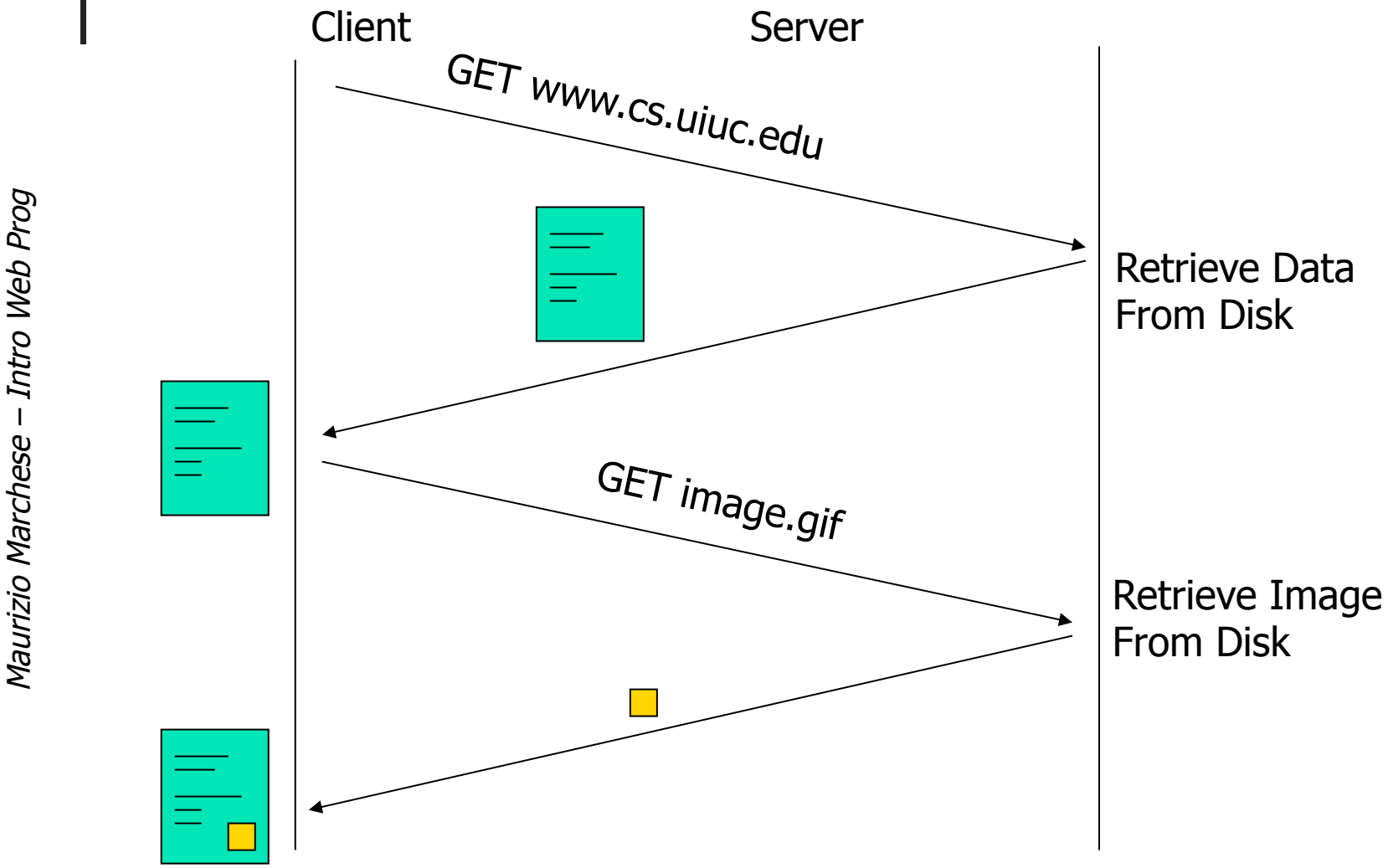


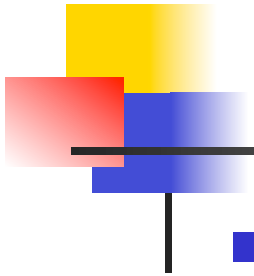


Fetching Multiple Objects (1.0)

- Most web-pages contain embedded objects (e.g., images, backgrounds, etc)
- Browser requests HTML page
- Server sends HTML file
- Browser parses file and requests embedded objects
- Server sends requested objects

Fetching Embedded Objects (1.0)





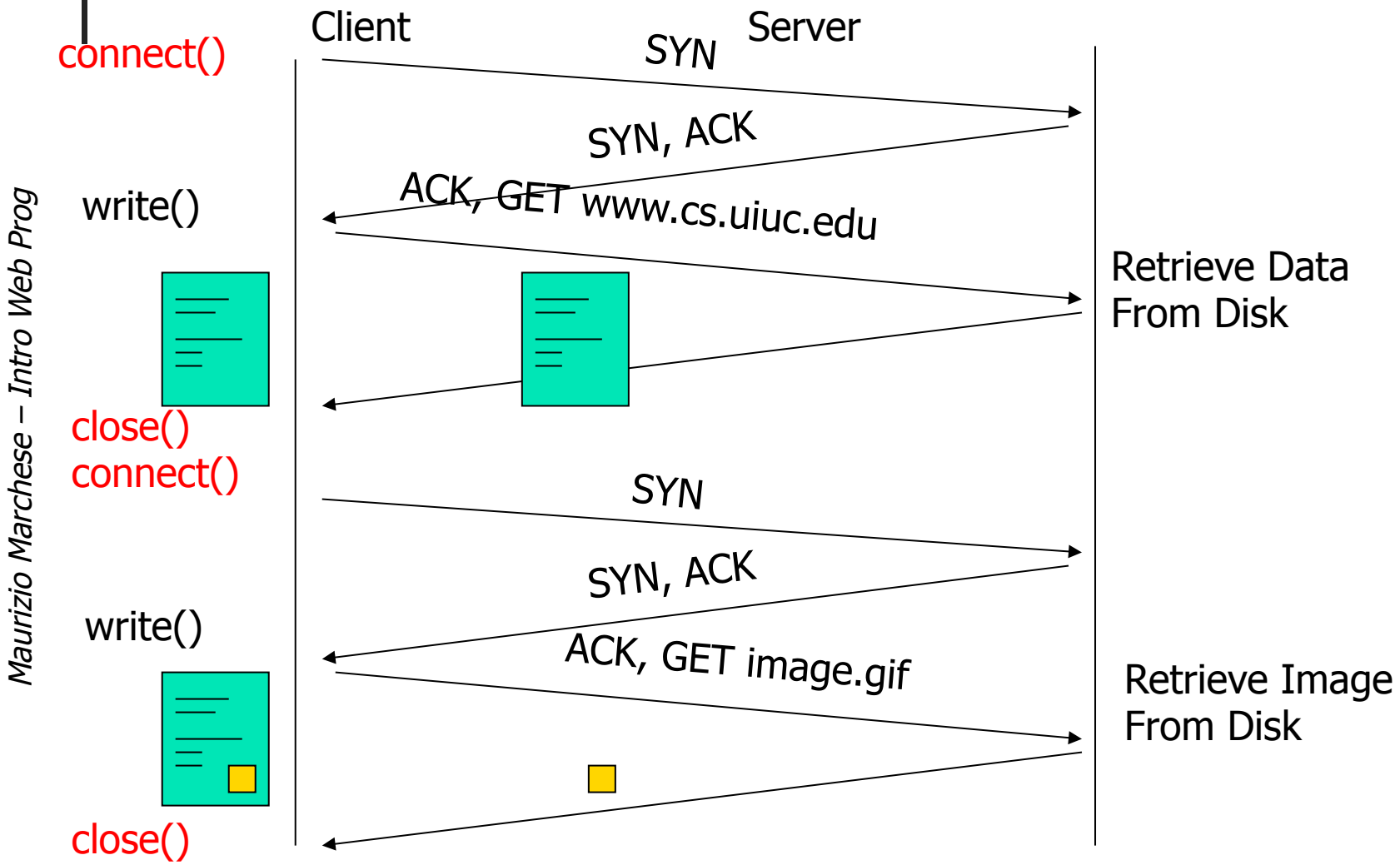
HTTP Reply

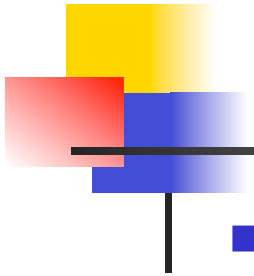
- The http response from the server to the client is sent using the same TCP connection as the request (NB TCP connection are bidirectional)
- It consist of an **Header** and a **Body**
  - **Header**: Status, Date, Server info, Body info..
  - **Body**: if request is successful contains the resource that has been requested

HTTP 1.0

- Client opens a separate TCP connection for each requested object
- Object is served and connection is closed
- Advantages
  - maximum concurrency
- Limitations
  - TCP connection setup/tear-down overhead
  - TCP slow start overhead

HTTP 1.0

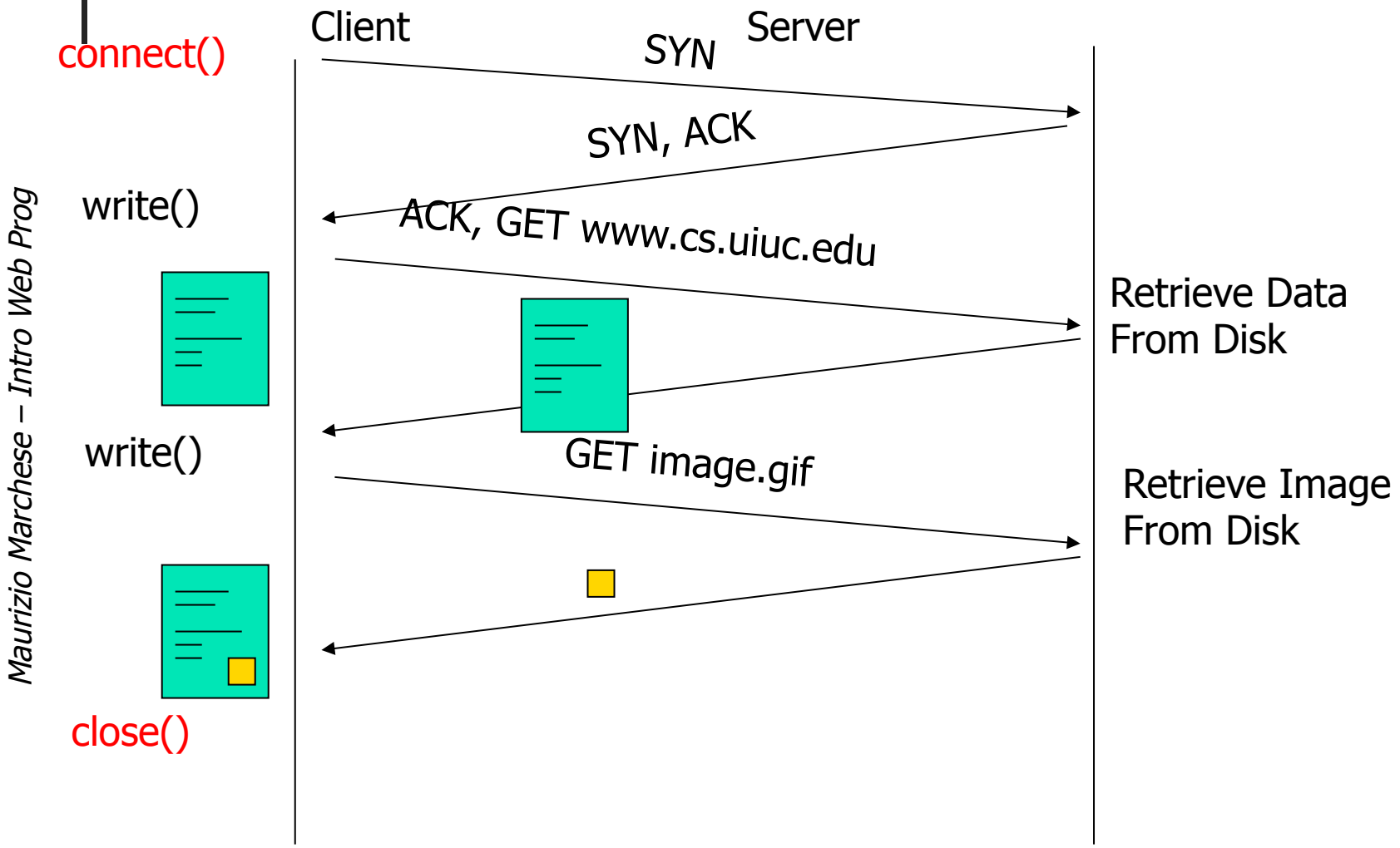

Client
Server
connect()
SYN
SYN, ACK
write()
ACK, GET www.cs.uiuc.edu
Retrieve Data
From Disk
close()
connect()
SYN
SYN, ACK
write()
ACK, GET image.gif
Retrieve Image
From Disk
close()
Maurizio Marchese – Intro Web Prog



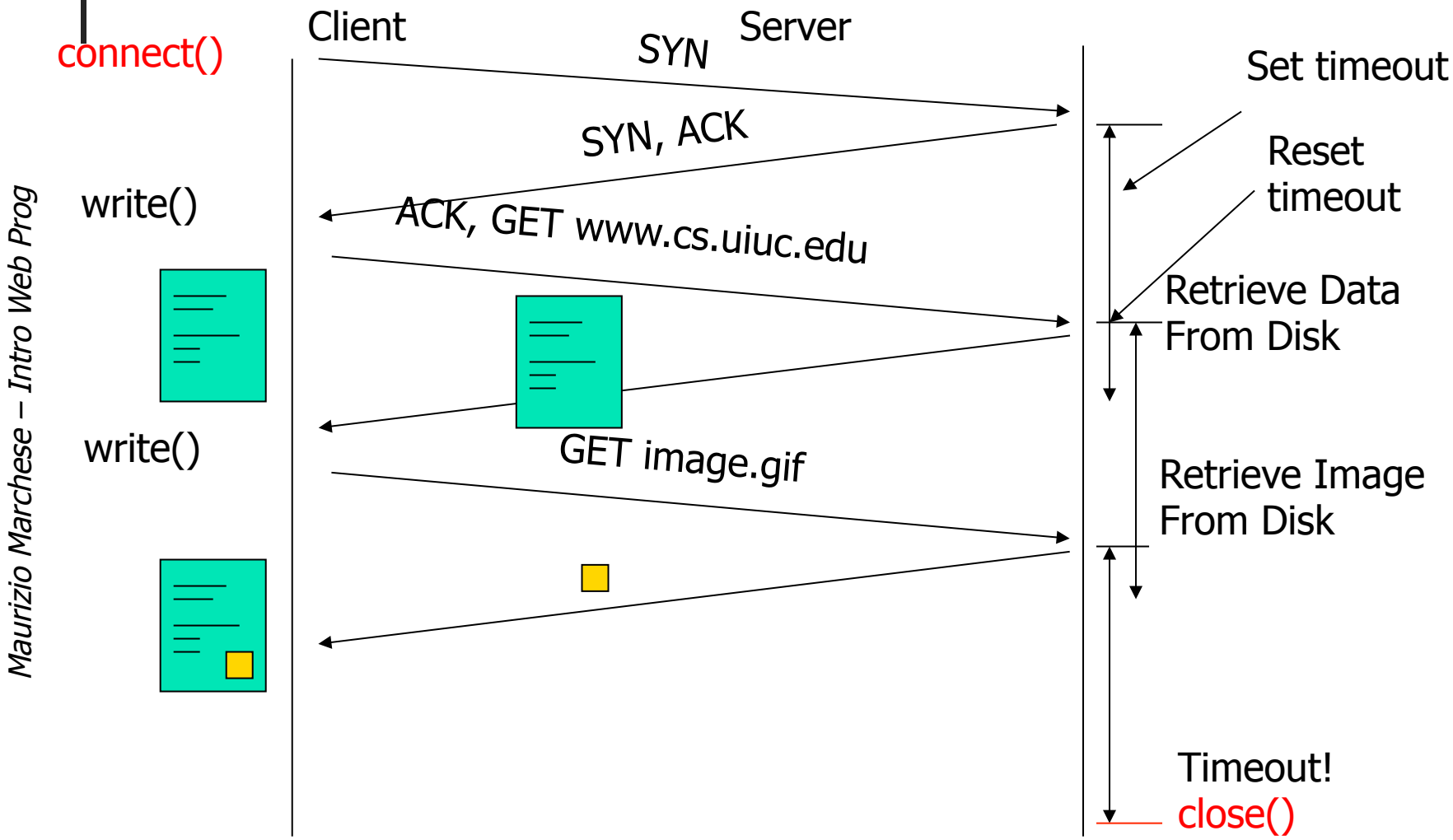
HTTP 1.1

- To avoid a connection per object model, HTTP 1.1 supports *request persistent connections*
- Client opens TCP connection to server
- All requests use same connection
- When the whole page (including sub-objetcs) is loaded the TCP connection is closed
- Problems
  - Less concurrency
  - Server does not know when to close idle connections

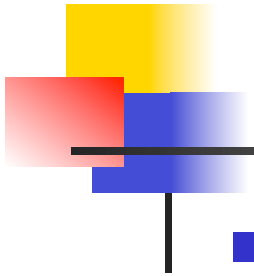
HTTP 1.1


Client
Server
connect()
SYN
SYN, ACK
write()
ACK, GET www.cs.uiuc.edu
Retrieve Data
From Disk
write()
GET image.gif
Retrieve Image
From Disk
close()
Maurizio Marchese – Intro Web Prog

Server Side Close()



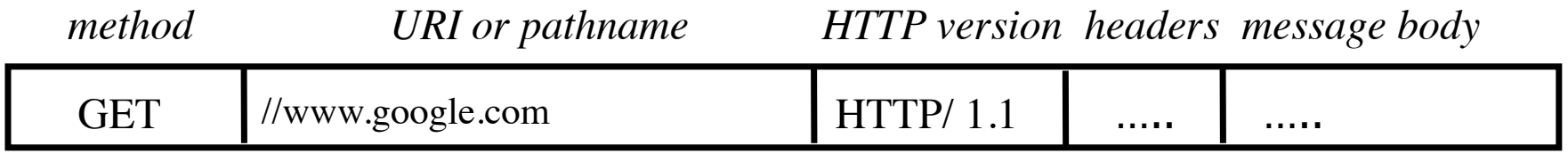
HTTP Operations



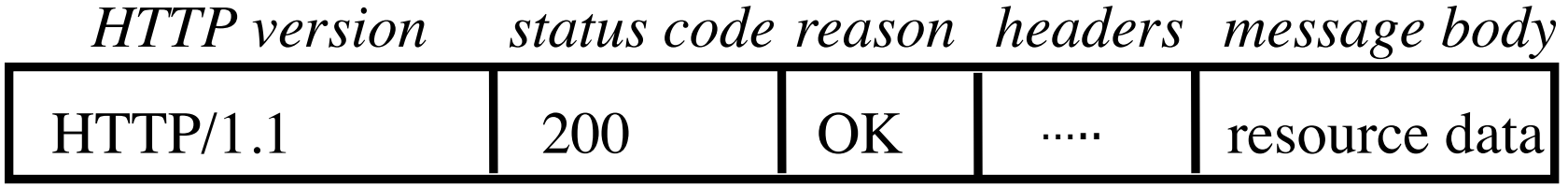
- GET: retrieves URL (most widely used)
- HEAD: retrieves only response header
- POST: posts data to server
- PUT: puts page on server
- DELETE: deletes page from server

HTTP request-replay message

- request message



- replay message



URI = URL+URN

- URI *Uniform Resource Identifier* is a standard mechanism to identify electronic resources
- URI = *Uniform Resource Locator + Uniform Resource Name*

- Syntax:

  <scheme> : <scheme dependent info>

- Example:

  http : //www.isoc.org/internet/history/

HTTP Request

- HTTP uses Uniform Resources Identifier (URI)
  - abstract URI: http://host:port/path?query
  - real URI: http://www.google.com/search?q=An+Introduction+to+Web+Technologies
- The HTTP request is sent using TCP.
  - in the above example, the browser will create a message like the following

```
GET /search?q=An+Introduction+to+Web+Technologies HTTP/1.1
Host: www.google.com
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv1.7.2)
Accept: text/xml, application/xml, application/xhtml+xml, text/html;q=0.9, text/
plain;q=0.8, image/png, */*;q=0.5
Accept-Language: it, en-us;q=0.8,en; q=0.5,sw; q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1, utf-8; q=0.7,*; q=0.7
...
```

HTTP GET anatomy (1)

GET /search?q=An+Introduction+to+Web+Technologies HTTP/1.1

→ ask the server to the resourse "search?q=An+Introduction+to+ Web+Technologies" using version 1.1. of the HTTP protocol

→ It uses a GET because the creator of the web page has programmed so in the code

→ all other lines are header lines each having the form field:value

Host: www.google.com

→ field: Host

→ value: domain name of the server

User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv1.7.2)

→ value: information about the user agent (browser or search robot) to tailor the response

HTTP GET anatomy (2)

Accept: text/xml, application/xml, application/xhtml+xml, image/png, text/html;q=0.9, text/plain;q=0.8, */*; q=0.5

→ value: specify media types that are acceptable as a response (MIME - Multipurpose Internet Mail Extensions - types)
The q parametes indicate a relative quality factor (0: not acceptable; 1:default)

Accept-Language: it, en-us;q=0.8,en; q=0.5,sw; q=0.3
→ value: acceptable/preferred natural languages

Accept-Encoding: gzip,deflate
→ value: acceptable/preferred content codings

Accept-Charset: ISO-8859-1, utf-8; q=0.7,*; q=0. 7
→ value: acceptable/preferred character encoding

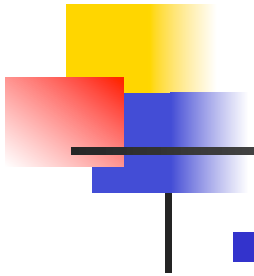
HTTP Response anatomy

HTTP/1.1 200 OK
Date: Fri, 17 Sep 2009 07:59:01 GMT
Server: Apache/2.0.50 (Unix) mod_perl/1.99_10 Perl/v5.8.4 mod_ssl/2.0.50 OpenSSL/0.9.7d DAV/2 PHP/4.3.8 mod_bigwig/2.1-3
Last-Modified: Tue, 24 Feb 2009 08:32:26 GMT
ETag: "ec002-afa-fd67ba80"
Accept-Ranges: bytes
Content-Length: 2810
Content-Type: text/html

HEADER

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
...
....
</html>

BODY



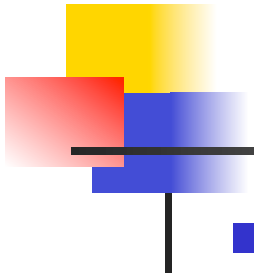
Status Codes

- 200 OK
- 301 Moved Permanently
- 400 Bad Request
- 401 Unauthorized
- 403 Forbidden
- 404 Not Found
- 500 Internal Server Error
- 503 Service Unavailable

Other HTTP methods: POST

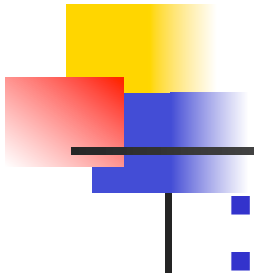
- The HTTP-GET protocol creates a query string of the name-and-value pairs and then appends the query string to the URL of the script on the server that handles the request.
  - Therefore, you can "read" easily the request.

- The HTTP-POST protocol passes the name-and-value pairs in the body of the HTTP request message. It is useful:
  - to provide data to data-handling process (servlet,cgi,php,…)
  - to interact with databases
  - to improve (not much) privacy



Limitations of HTTP

- **Stateless**, no built-in support for tracking clients (session management)
  - Need for special programming techniques for session management: Cookies, Sessions, URL rewriting, Hidden Form Fields.
- No built-in **security** mechanisms
  - Need for special security protocols



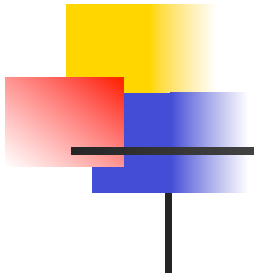
Security

- SSL: *Secure Sockets Layer*
- TLS: *Transport Layer Security* (newer version)
- Layer between HTTP and TCP, accessed by HTTPS://...

- Based on public-key cryptography
  - private key + public key
  - certificate (usually for server authentication only)

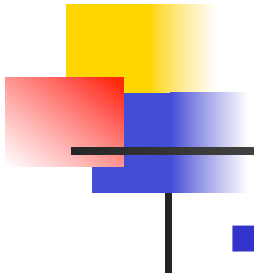
- The main idea of HTTPS is to create a secure channel over an insecure network.
  - This ensures reasonable protection from eavesdroppers and man-in-the-middle attacks, provided that adequate cipher suites are used and that the server certificate is verified and trusted.

- The trust inherent in HTTPS is based on major certificate authorities that come pre-installed in browser software
  - this is equivalent to saying "I trust certificate authority (e.g. VeriSign/ Microsoft/etc.) to tell me whom I should trust”



Dynamic Content

- Web pages can be created as requests arrive
- Advantages
  - Personalization (e.g., my.yahoo.com),
  - interaction with client input
  - interaction with back-end applications
- Disadvantages
  - Performance penalty
- Generating dynamic content (CGI, ASP, PHP, ColdFusion, JavaScript, Flash, …)



CGI Scripts

- CGI scripts are URLs with a .cgi extension
- The script is a program (e.g., C, JAVA, …)
- When the URL is requested, server invokes the named script, passing to it client info
- Script outputs HTML page to standard output (redirected to server)
- Server sends page to client

CGI Execution