MONTANA STATE UNIVERSITY

Chapter 2

Socket communications and HTTP

Process communications

Application processes send msgs.

Via sockets

7: Application	Facebook				7: Application	Application
4: Transport	TCP				4: Transport	
3: Network	IP	3: Network	IP		3: Network	OS
2: Data link	GSM	2: Data link	GSM		2: Data link	Network cards and other h/w
1: Physical	EM	1: Physical	EM		1: Physical	

Control by:

Internet transport protocols

User Datagram Prot. (UDP)

Unreliable data transfer

- Connection-less
  - Don't know if receiver is present
- No flow control
  - Buffer overflow at receiver possible
- No congestion control
  - Sender can overload the network
- No guarantees on
  - End-to-end delay
  - Throughput
  - Security

Transmission Control Prot. (TCP)

Reliable stream transport

- Connection-oriented
  - Establishes receiver presence
- Flow control
  - Sender won't overwhelm receiver
- Congestion control
  - Senders won't overload network
- No guarantees on
  - End-to-end delay
  - Throughput
  - Security

Socket programming

UDP

```
import socket

#SERVER
s = socket.socket(socket.AF_INET,
                  socket.SOCK_DGRAM)
s.bind(('127.0.0.1', 5000))

#CLIENT
s = socket.socket(socket.AF_INET,
                  socket.SOCK_DGRAM)
s.sendto(bytes('hello'),
         ('153.90.118.46',5000))

#SERVER
data, addr = s.recvfrom(BUFFER_SIZE)
```

UDP socket

Address family Internet

TCP

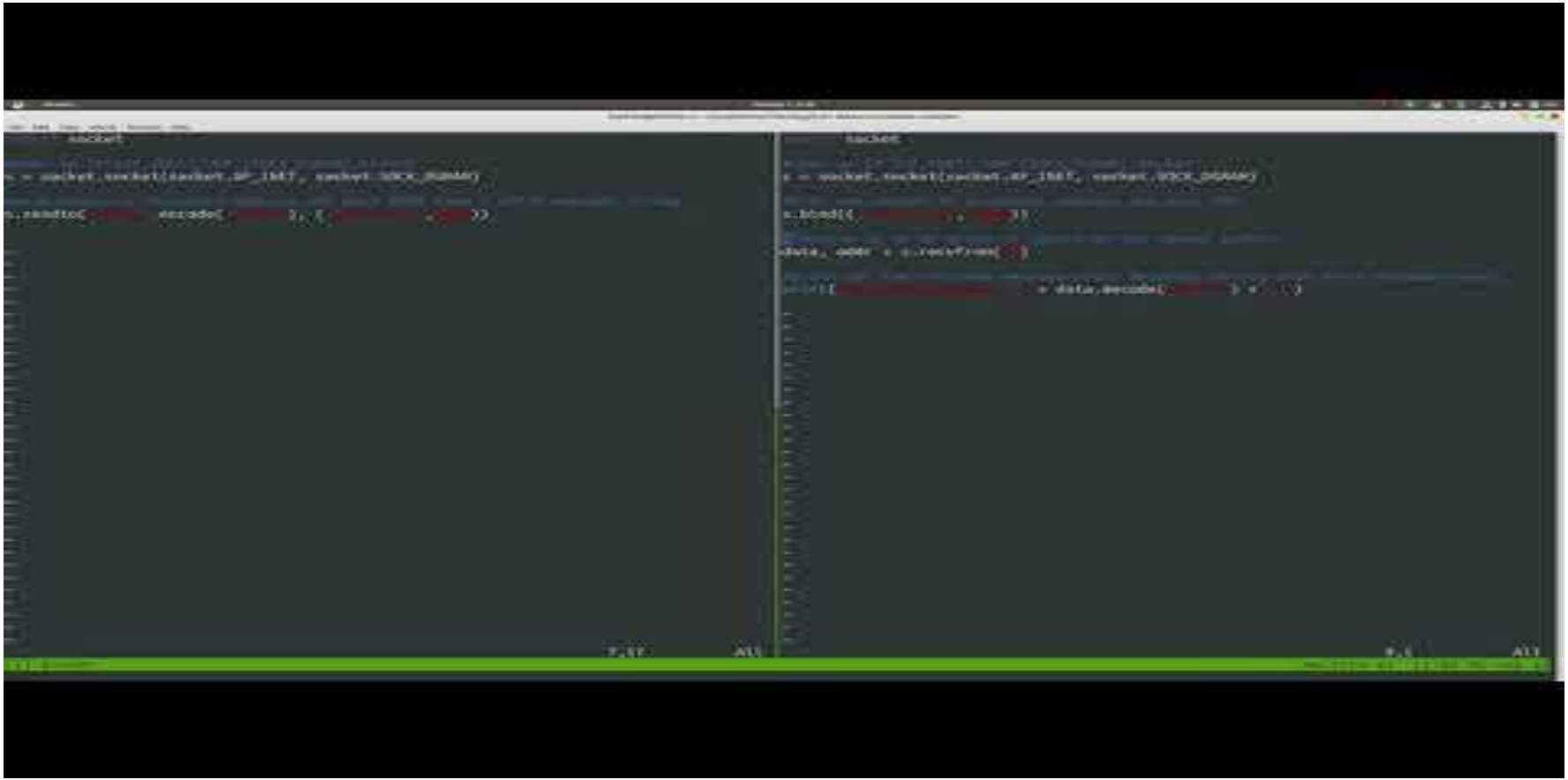
```
#SERVER
s = socket.socket(socket.AF_INET,
                  socket.SOCK_STREAM)
s.bind (('127.0.0.1', 80))
s.listen(1)

#CLIENT
s = socket.socket(socket.AF_INET,
                  socket.SOCK_STREAM)
s.connect(('72.21.211.176', 80))
s.send(bytes('GET ...'))
data = s.recv(BUFFER_SIZE)
s.close()

#SERVER
conn, addr = s.accept()
data = conn.recv(BUFFER_SIZE)
conn.send(data)  # echo
conn.close()
```

TCP socket

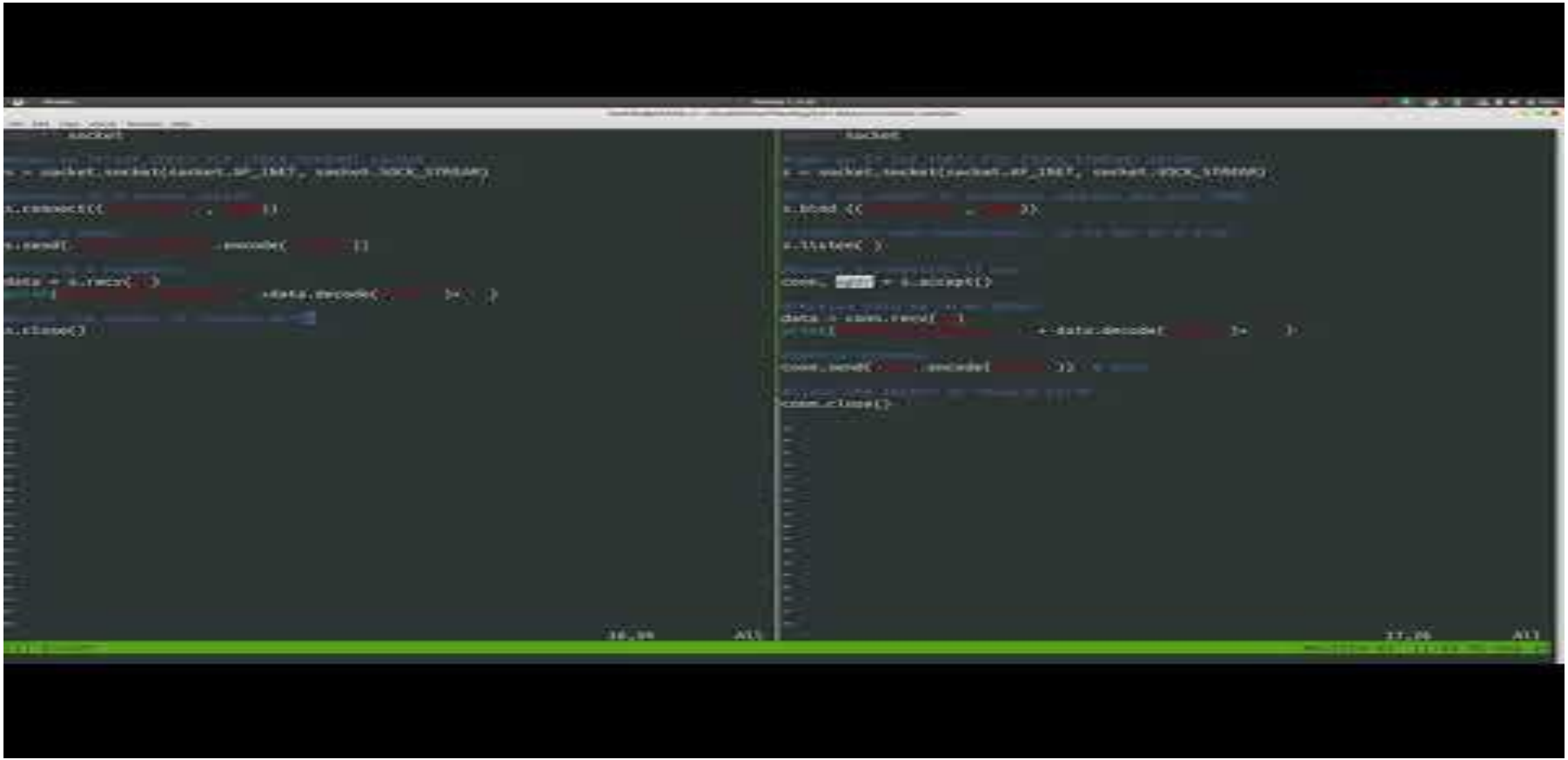
New socket

Remote address

UDP Communication Example



TCP Communication Example



Connection details

153.90.118.46

72.21.211.176

Time

[(127.0.0.1,80), srv_sock]

[(153.90.118.46,5000), (72.21.211.176,80)]

[(127.0.0.1,1234), 153.90.118.46,5000)]

Socket uniquely identified by a four-tuple

TCP

```
#SERVER
s = socket.socket(socket.AF_INET,
                  socket.SOCK_STREAM)
s.bind ((‘127.0.0.1’, 80))
s.listen(1)

#CLIENT
s = socket.socket(socket.AF_INET,
                  socket.SOCK_STREAM)
s.connect((‘72.21.211.176’, 80))
s.send(bytes(‘hello’))
data = s.recv(BUFFER_SIZE)
s.close()

#SERVER
conn, addr = s.accept()
data = conn.recv(BUFFER_SIZE)
conn.send(data)  # echo
conn.close()
```

HyperText Transfer Protocol (HTTP)

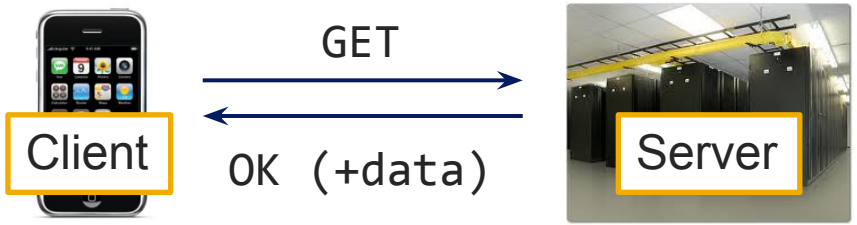
- HyperText Markup Language (HTML)
  - Language of the Web - now HTML5
  - Allows grouping of objects as content

- Uniform Resource Locator (URL)
  - Addressing scheme for Web objects

```
scheme://domain:port/
        path?query_string#fragment_id
```

- HTTP supports HTML
  - Protocol for fetching objects from URLs
  - Uniform API for different platforms
  - Stateless protocol
    - Servers do not 'remember' past client requests

```
http://server.com/pathto/index.htm
```




```
GET /index.html HTTP/1.1\r\n
Host: www.server.com\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html\r\n
Accept-Language: en-us,en; \r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

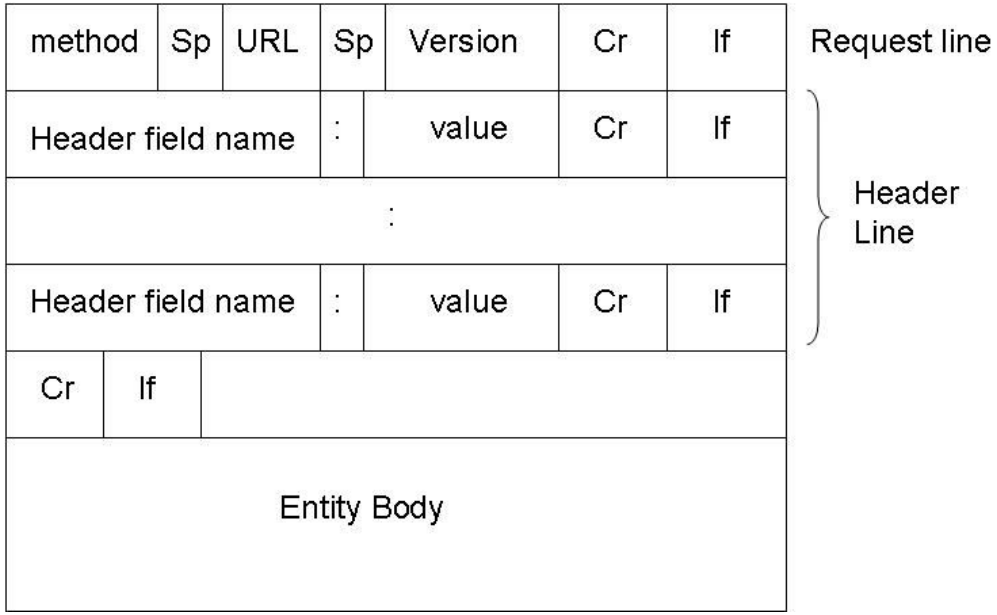
Request line

Header lines

Double carriage return + line feed

HTTP message format

`GET:` Download resource
`HEAD:` Get resource metadata
`POST:` Upload form contents
`PUT:` Upload object to URL
`DELETE:` Delete object from URL

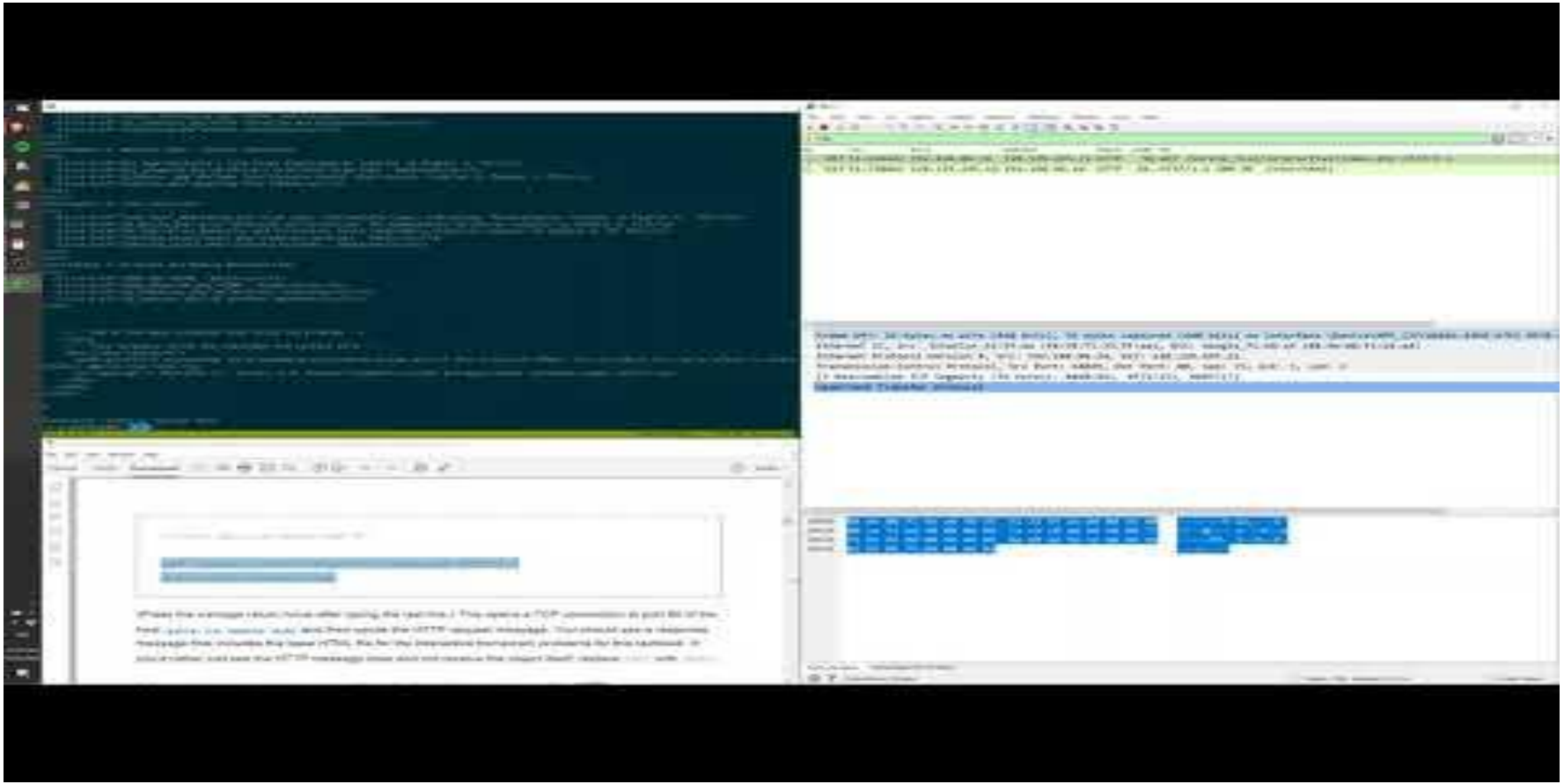



```
HTTP/1.1 200 OK\r\n
Connection: Close\r\n
\r\n
```

Some HTTP responses:

`200 OK:` Data included
`400 Bad Request:` Bad formatting
`404 Not Found:` Object not on server
`505 HTTP Version Not Supported:` Data included

HTTP Example



HTTP API

- HTTP relies on TCP
  - Not a transport protocol itself
  - Relies on E2E reliability and congestion control mechanisms of TCP

- Application Programming Interfaces (APIs) in many High Level Languages (HLAs)
  - Ex. Python

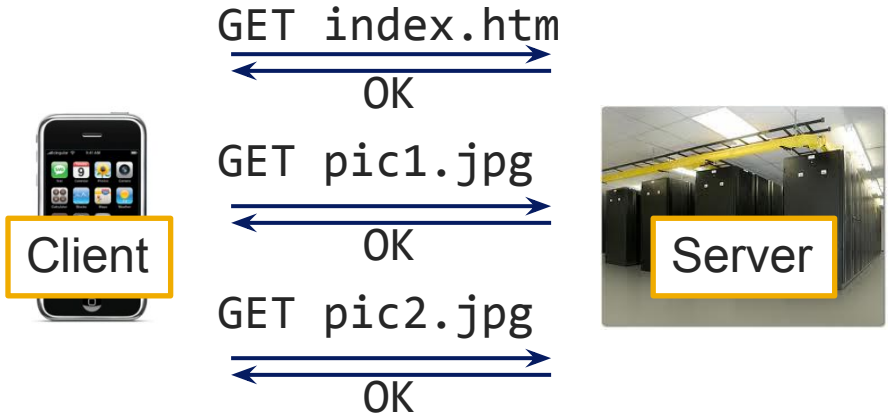
What is the format of data returned by the read() functions?

```
import httplib
conn=httplib.HTTPConnection(“srvr.com")
conn.request("GET", "/pathto/object.jpg")
r = conn.getresponse()
data = r.read()
conn.close()
```

```
import urllib
params = urllib.urlencode(
    {'spam': 1, 'eggs': 2, 'bacon': 0})
f = urllib.urlopen("http://srvr.com/
            cgi-bin/query?%s" % params)
data = f.read()
```

HTTP transfer modes

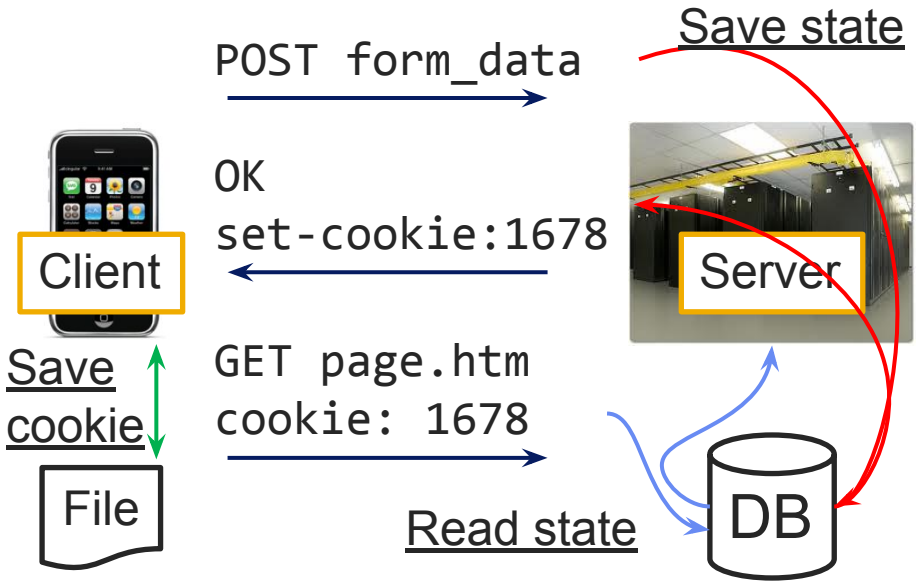
```
<!DOCTYPE html>
<html>
 <body>
  <h1>Some objects</h1>
  <img src="pic1.jpg" />
  <img src="pic2.jpg" />
 </body>
</html>
```



- Non-persistent connections (HTTP/1.0)
  - Each request over separate TCP connection
    - – Slow ramp up of TCP transfer rate
  - Parallel connections for multiple objects
    - – Wasted server resources
- Persistent connections (HTTP/1.1)
  - One TCP connection for all objects
    - + Low per client server overhead
  - Pipelining: objects requested in bulk
    - + Allows TCP to reach fast transfer rates
- Parallel transfers (HTTP/2.0)
  - Multiple elements loaded in parallel on same connection
  - Header compression
  - Eliminate head of line blocking
  - Server push

Client side data: Cookies

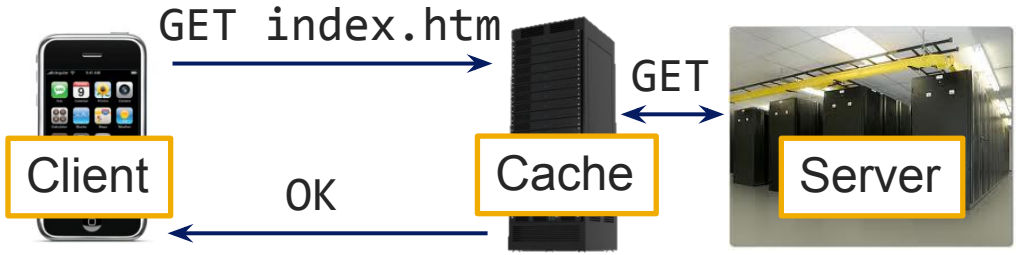
- Problem
  - HTTP is *stateless*
  - Resending client data wastes network and server resources
- Cookies
  - Associate client state on server with unique id
  - Clients refer to their state through id
- Applications:
  - Authentication
  - Shopping carts



- Cookies and privacy
  - Can store your personal information (forms, ad history, browsing history, shopping cart contents)
  - Can be exploited by other sites

Client side data: Caching

- Problem
  - HTTP is stateless
  - Resending server data wastes bandwidth
- Caching
  - Save previously delivered data
  - Subsequent requests served from cache on the browser, or in the access network
- Applications
  - Reduce response time for client request
  - Reduce ISP traffic costs
- Content distribution networks
  - Distributed caches
  - Web objects addressed to CDN server
  - CDN server fetches from content provider on first access



- Conditional GET
  - Cache: specify date of cached copy in HTTP request
    `If-modified-since: <date>`
  - Server: response contains no object if cached copy is up-to-date:
    `HTTP/1.0 304 Not Modified`

Why is sending a conditional GET all the way to the server still faster, than a non-conditional GET?