

CS 43: Computer Networks

05: HTTP Concurrency and Performance

September 17, 2024

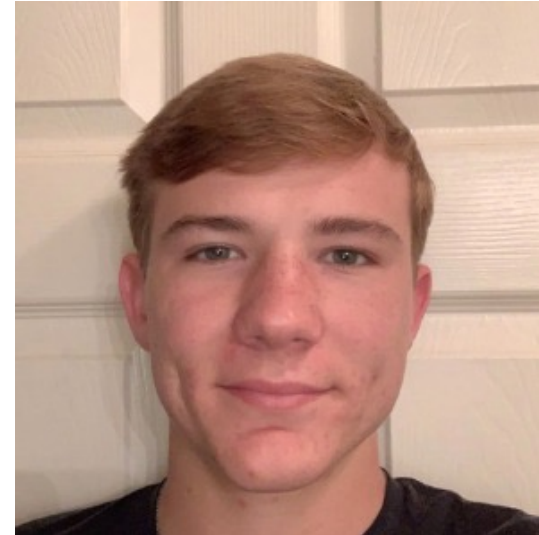


Slides adapted from Kurose & Ross, Kevin Webb

Reading Quiz

Announcements

- TA for the course: Marcus Wright
 - Office Hours: 2 – 4pm in Overflow.
- Regarding missed classes/labs
 - three free misses on classes
 - lab attendance is mandatory



Midterm Scheduling: Monday Oct 21st 7 – 8.30 PM

Can you make this time?

A. Yes

B. No

Client-Server communication

- Client:
 - initiates communication
 - must know the address and port of the server
 - active socket
- Server:
 - passively waits for and responds to clients
 - passive socket

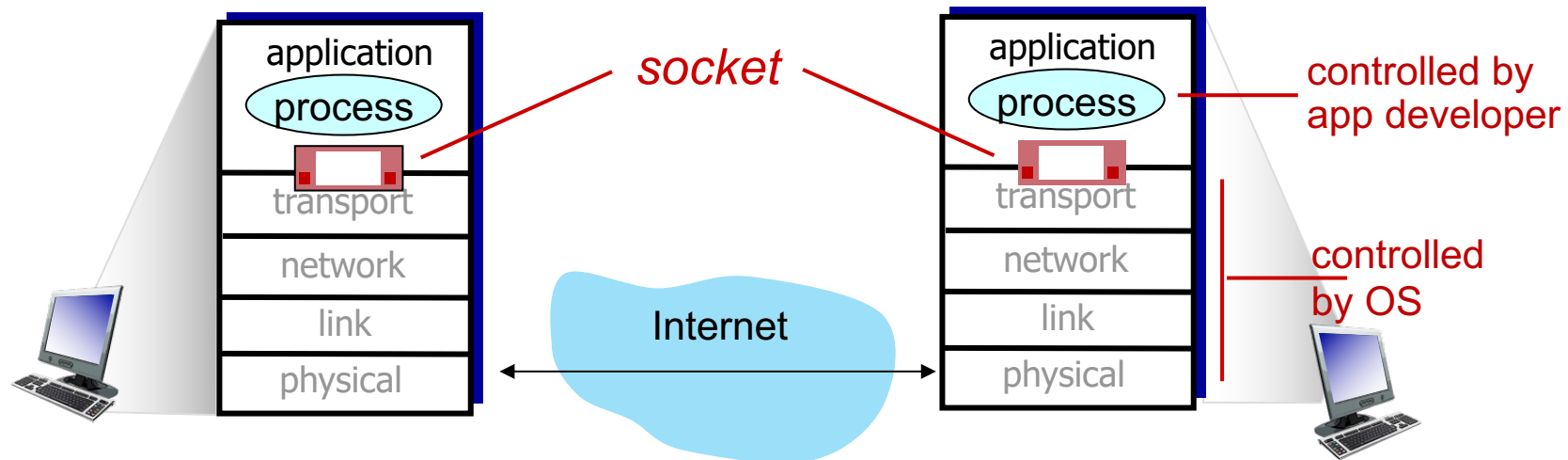
What is a socket?

An abstraction through which an application may send and receive data,

in the same way as a open-file handle or file pointer allows an application to read and write data to storage.

Sockets

- process sends/receives messages to/from its **socket**
- socket analogous to door
 - sending process shoves message out door
 - sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process
 - two sockets involved: one on each side



Blocking Summary

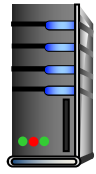
send()

- Blocks when socket buffer for sending is full
- Returns less than requested size when buffer cannot hold full size

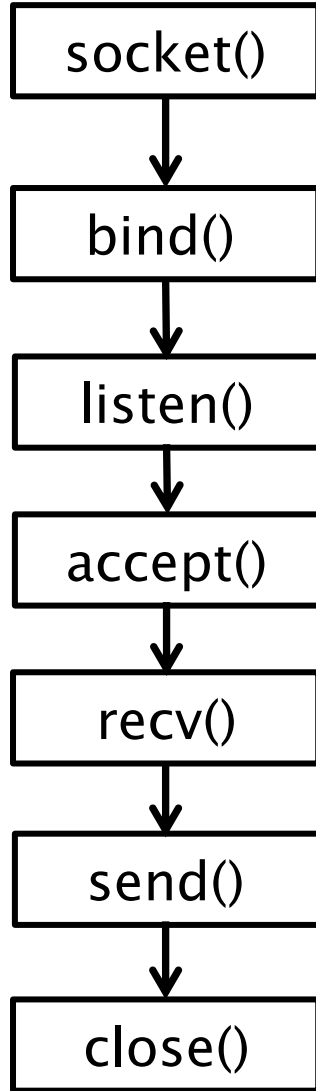
recv()

- Blocks when socket buffer for receiving is empty
- Returns less than requested size when buffer has less than full size

Always check the return value!



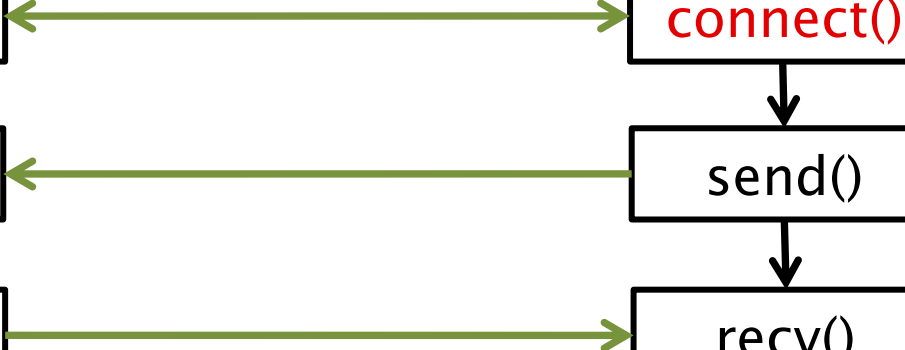
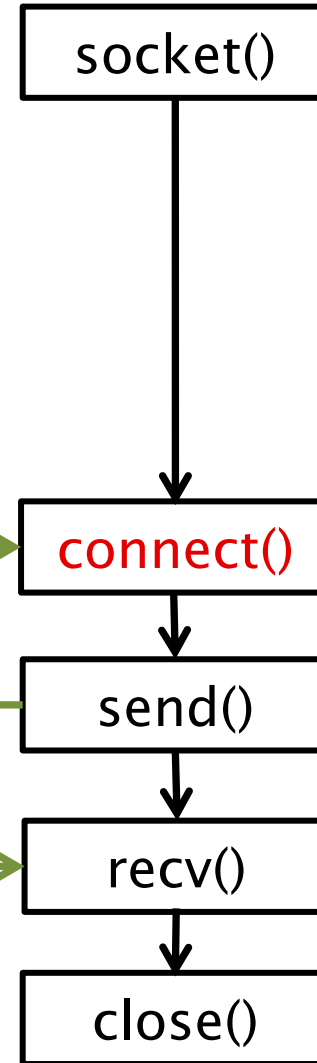
Server



connect()

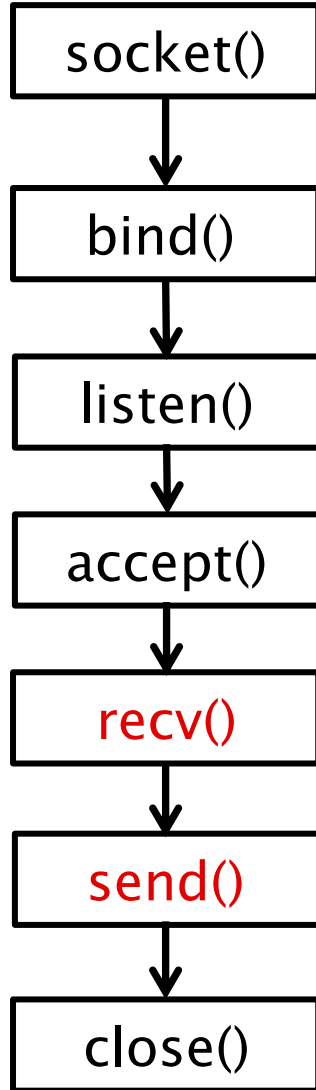


Client





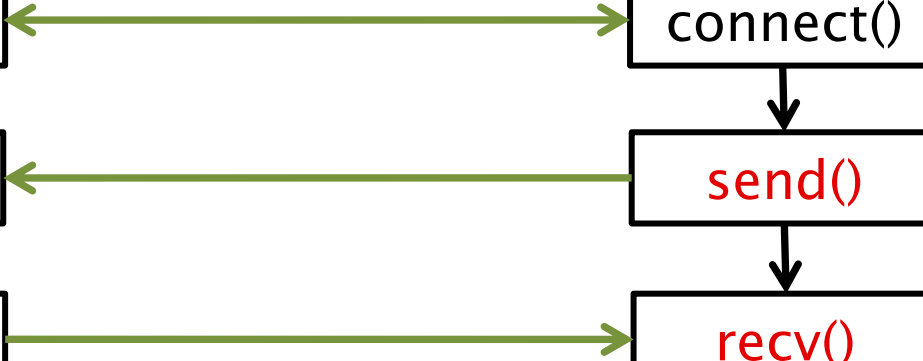
Server



send(), recv()



Client



Request Method Types (“verbs”)

HTTP/1.0 (1996):

- GET:
 - Requests page.
- POST:
 - Uploads user response to a form.
- HEAD:
 - asks server to leave requested object out of response

HTTP/1.1 (1997 & 1999):

- GET, POST, HEAD
- PUT
 - uploads file in entity body to path specified in URL field
- DELETE
 - deletes file specified in the URL field
- TRACE, OPTIONS, CONNECT, PATCH
- Persistent connections

Uploading form input

GET (in-URL) method:

- uses GET method
- input is uploaded in URL field of request line:

`www.somesite.com/animalsearch?monkeys&banana`

POST method:

- web page often includes form input
- input is uploaded to server in request entity body

GET vs. POST

GET can be used for **idempotent** requests

- Idempotence: an operation can be applied multiple times without changing the result (the final state is the same)

GET vs. POST

GET can be used for **idempotent** requests

- Idempotence: an operation can be applied multiple times without changing the result (the final state is the same)

Q: How many of the following operations are idempotent?

- | | |
|-------------------------------------|-------------------------|
| I. Incrementing a variable | III. Allocating Memory |
| II. Assigning a value to a variable | IV. Compiling a program |

- | | |
|-----------------|------------------|
| A. None of them | D. Three of them |
| B. One of them | E. All of them |
| C. Two of them | |

GET vs. POST

GET can be used for **idempotent** requests

- Idempotence: an operation can be applied multiple times without changing the result (the final state is the same)

Q: How many of the following operations are idempotent?

- | | |
|--|--------------------------------|
| I. Incrementing a variable | III. Allocating Memory |
| II. Assigning a value to a variable | IV. Compiling a program |

- | | |
|-----------------------|------------------|
| A. None of them | D. Three of them |
| B. One of them | E. All of them |
| C. Two of them | |

GET vs. POST

GET can be used for **idempotent** requests.

- Idempotence: an operation can be applied multiple times without changing the result (the final state is the same)

GET vs. POST

POST should be when:

- A request **changes the state of the server** or DB
- Sending a request twice would be harmful: (Some) browsers warn about sending multiple post requests
- Users are inputting **non-ASCII** characters
- Input may be very large
 - You want to hide how the form works/user input

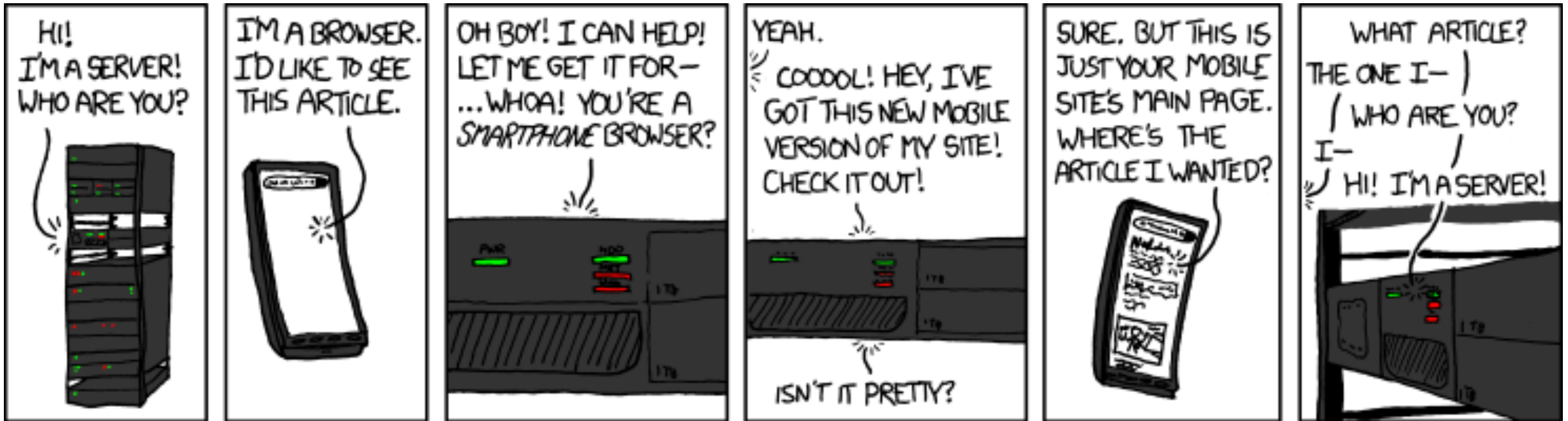
When might you use GET vs. POST?

	GET	POST
A.	Forum post	Search terms, Pizza order
B.	Search terms, Pizza order	Forum post
C.	Search terms	Forum post, Pizza order
D.	Forum post, Search terms, Pizza Order	
E.		Forum post, Search terms, Pizza Order

When might you use GET vs. POST?

	GET	POST
A.	Forum post	Search terms, Pizza order
B.	Search terms, Pizza order	Forum post
C.	Search terms	Forum post, Pizza order
D.	Forum post, Search terms, Pizza Order	
E.		Forum post, Search terms, Pizza Order

State(less)



(XKCD #869, "Server Attention Span")

HTTP State

Does the HTTP protocol, allow for a server to keep track of every client?

- A. Yes, it's required to
- B. No, it would not scale
- C. That's against privacy rules!
- D. Something else

State(less)

- Original web: simple document retrieval
- **Maintain State?** Server is not required to keep state between connections
...often it might want to though
- **Authentication:** Client is not required to identify itself
– server might refuse to talk otherwise though

User-server state: cookies

What cookies can be used for:

- authorization
- shopping carts
- recommendations
- user session state (Web e-mail)

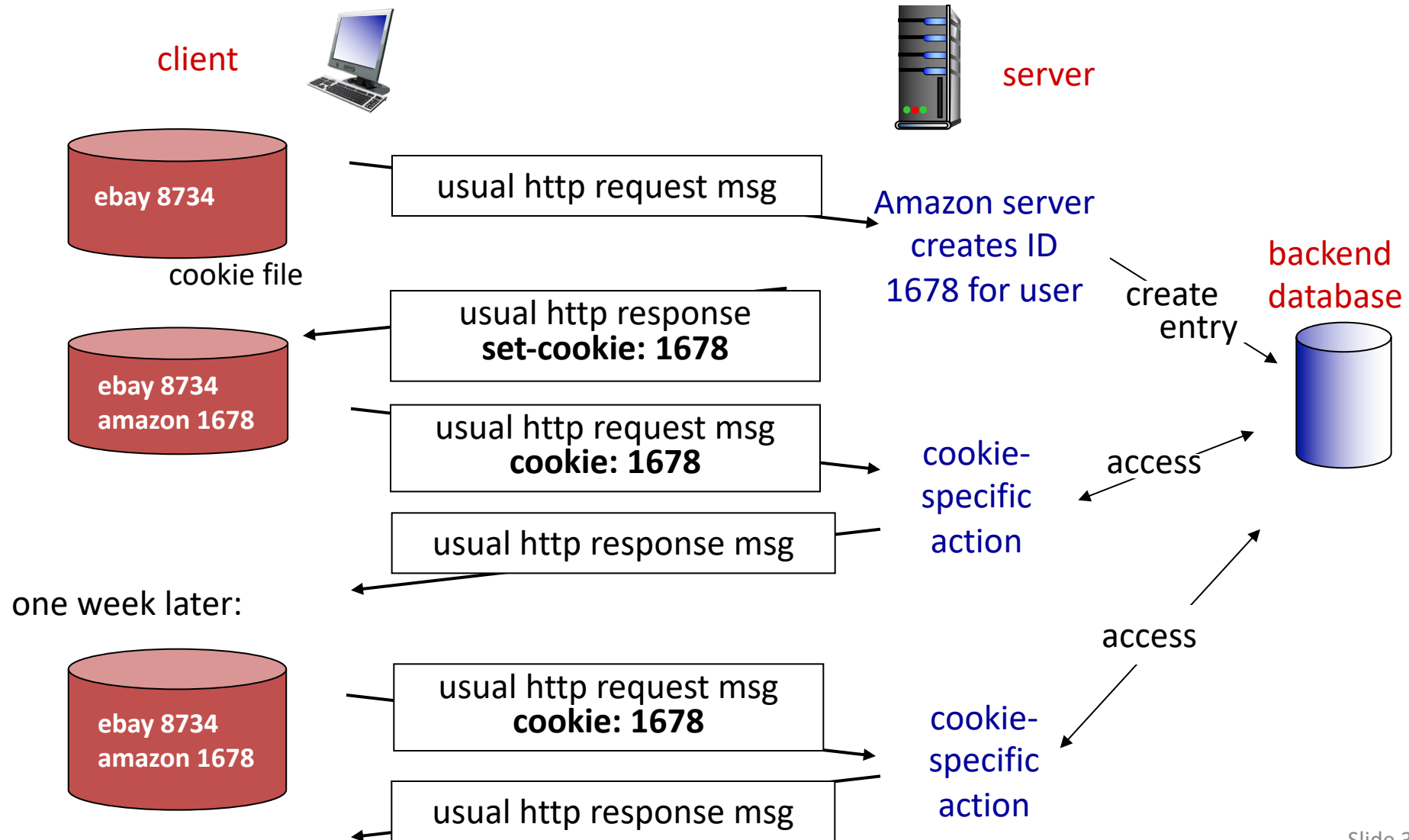
How to keep “state”:

- protocol endpoints: maintain state at sender/receiver over multiple transactions
- cookies: http messages carry state

What Are Cookies Used For?

- Authentication
 - The cookie proves to the website that the client previously authenticated correctly
- Personalization
 - Helps the website recognize the user from a previous visit
- Tracking
 - Follow the user from site to site;
 - Read about iPads on CNN and see ads on Amazon 🤖
 - How can an advertiser (A) know what you did on another site (S)?

Cookies: keeping "state" (cont.)



User-server state: cookies

Many web sites use cookies

Four components:

- 1) cookie header line of **HTTP response** message
- 2) cookie header line in **next HTTP request** message
- 3) cookie file kept on user's host, managed by user's browser
- 4) back-end database at Web site

Cookies and Privacy

Cookies permit sites to learn a lot about you

supply name and e-mail to sites (and more!)

third-party cookies (ad networks) follow you across multiple sites.

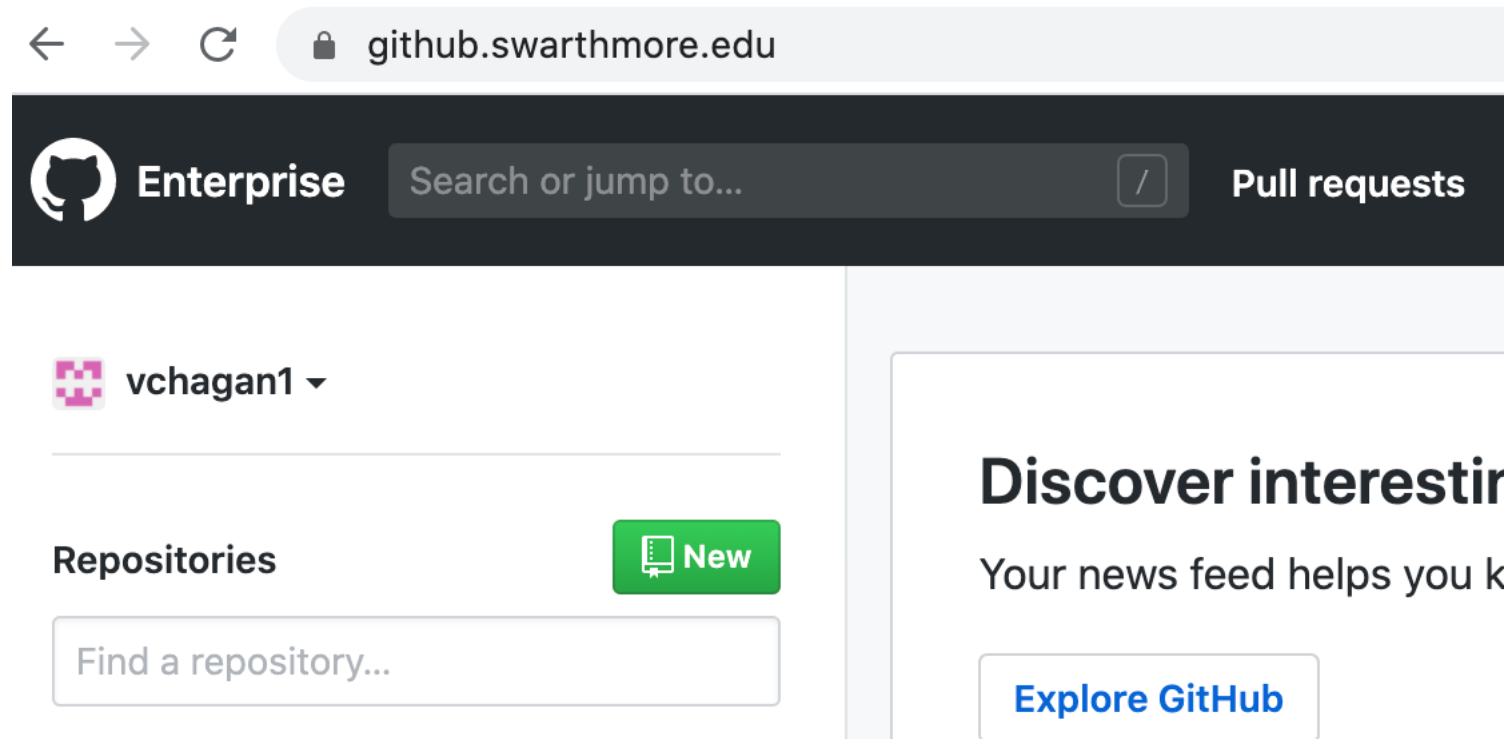


The screenshot shows the top section of the New York Times website. At the top, there is a large banner for WeWork. The banner is split into three parts: on the left, a modern office interior with a woman sitting at a table; in the center, a black background with the 'wework' logo in white, the text 'Hello office of tomorrow' and 'Designed for the new ways you work', and a 'LEARN MORE' button; on the right, a woman sitting on a blue sofa by a large window overlooking a city. Below the banner, there is a navigation bar with a menu icon, a search icon, and the text 'ENGLISH ESPAÑOL 中文'. To the right of the navigation bar are two buttons: 'SUBSCRIBE NOW' and 'LOG IN'. Below the navigation bar is the New York Times logo in its signature font. At the bottom left of the banner area, the date 'Monday, September 14, 2020' is displayed. At the bottom right, the text 'Today's Paper' is visible.

Cookies and Privacy

Cookies permit sites to learn a lot about you

You could turn them off ...but good luck doing anything on the internet!



Login Session

GET /loginform HTTP/1.1

cookies: []

HTTP/1.1 200 OK

cookies: []

POST /login HTTP/1.1

cookies: []

<html><form>...</form></html>

username: chaganti

password: swarthmore

HTTP/1.0 200 OK

cookies: [session: e82a7b92]

<html><h1>Login Success</h1></html>

GET /account HTTP/1.1

cookies: [session: e82a7b92]

GET /img/user.jpg HTTP/1.1

cookies: [session: e82a7b92]

HTTP connections

Non-persistent HTTP

- at most one object sent over TCP connection
 - connection then closed
- downloading multiple objects requires multiple connections

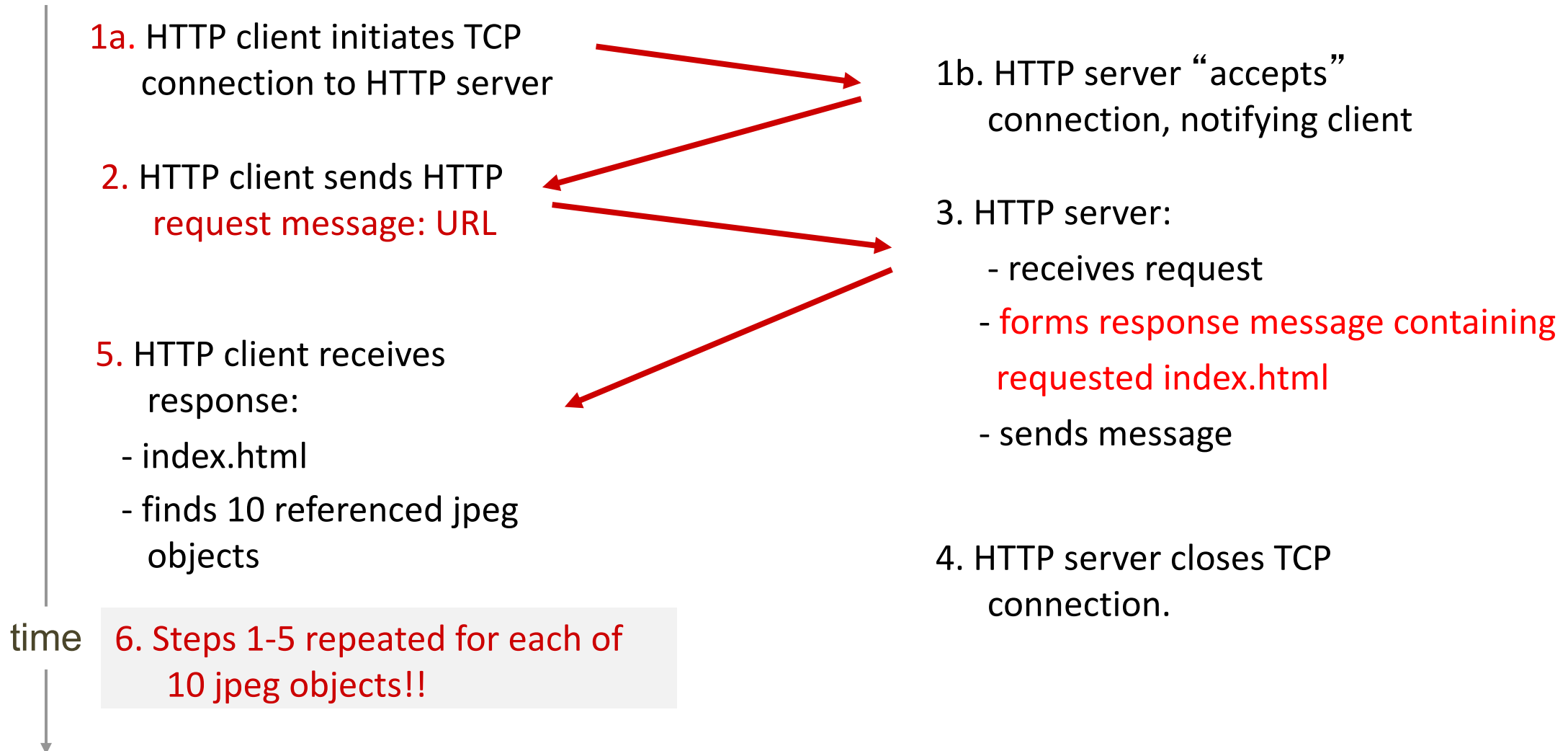
Persistent HTTP

- multiple objects can be sent over single TCP connection between client, server

object: image, script, stylesheet, etc.

Non-persistent HTTP

suppose user enters URL: contains references to 10 jpeg images



Pseudocode Example

non-persistent HTTP

for object on web page:

connect to server

request object

receive object

close connection

persistent HTTP

connect to server

for object on web page:

request object

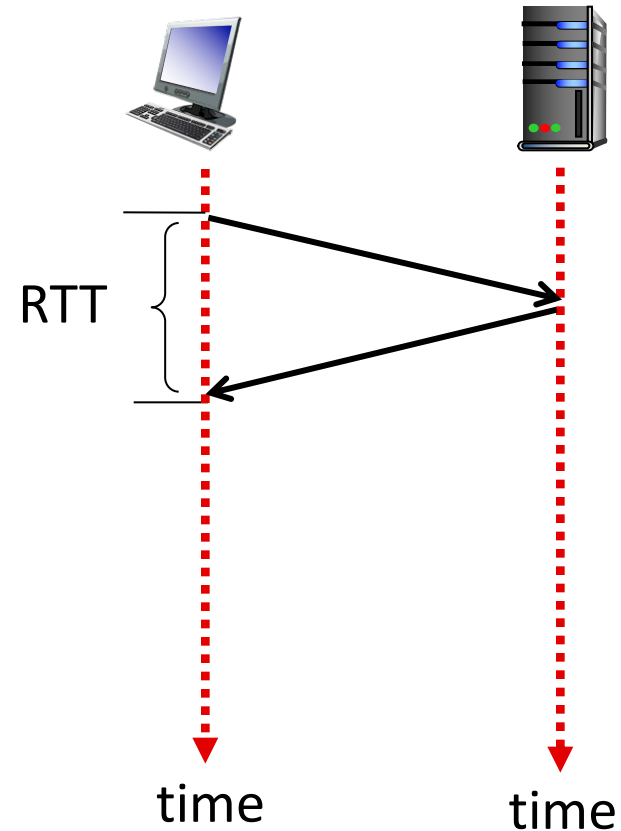
receive object

close connection

Round Trip Time

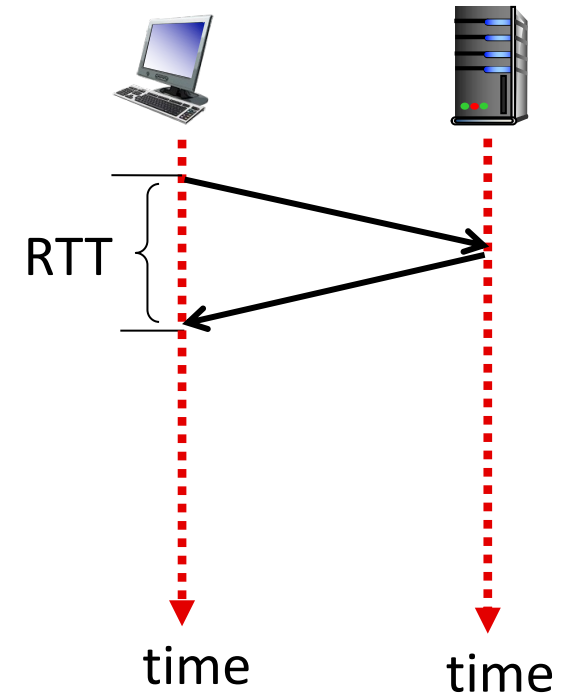
Round Trip Time (RTT):

- time for a small packet to travel from client to server and response to come back.
- Connection establishment (via TCP) requires **one RTT**.

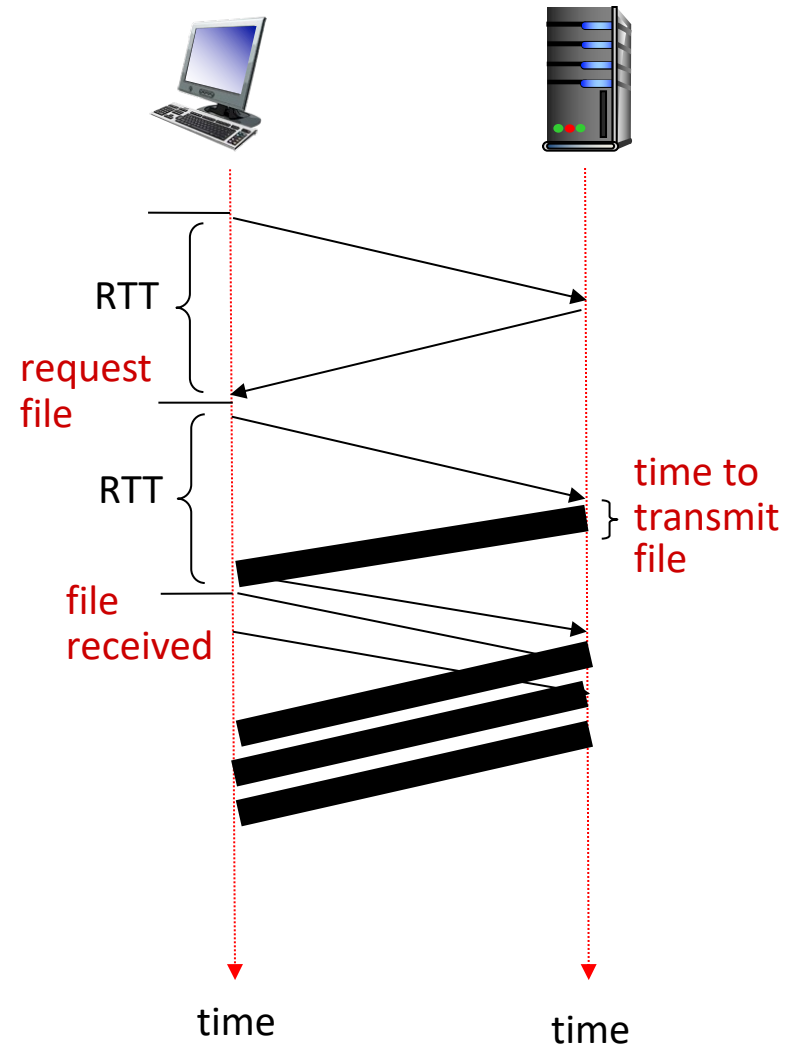


Non-Persistent HTTP Connections can download a website with several objects in...

- A. One RTT + (File transfer time per object)
- B. (One RTT + File transfer time) per object
- C. Two RTTs
- D. Two RTTs + (File transfer time per object)
- E. (Two RTTs + File transfer time) per object



Persistent Connection



Persistent HTTP

Non-persistent HTTP issues:

- requires **2 RTTs** per object
- OS overhead for **each** TCP connection
- browsers often open parallel TCP connections to fetch referenced objects

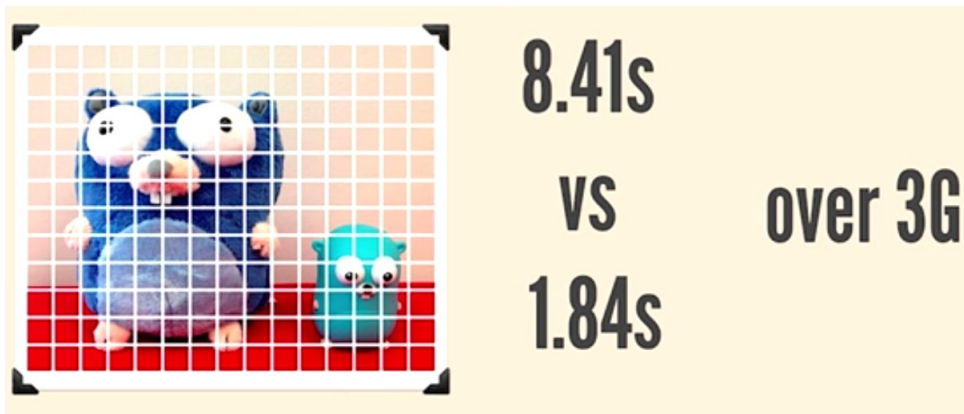
Persistent HTTP:

- server leaves connection open after sending response
- subsequent HTTP messages between same client/server sent over open connection
- client sends requests as soon as it encounters a referenced object
- **as little as one RTT for all the referenced objects**

HTTP 1.x vs HTTP 2.0 vs. HTTP 3.0



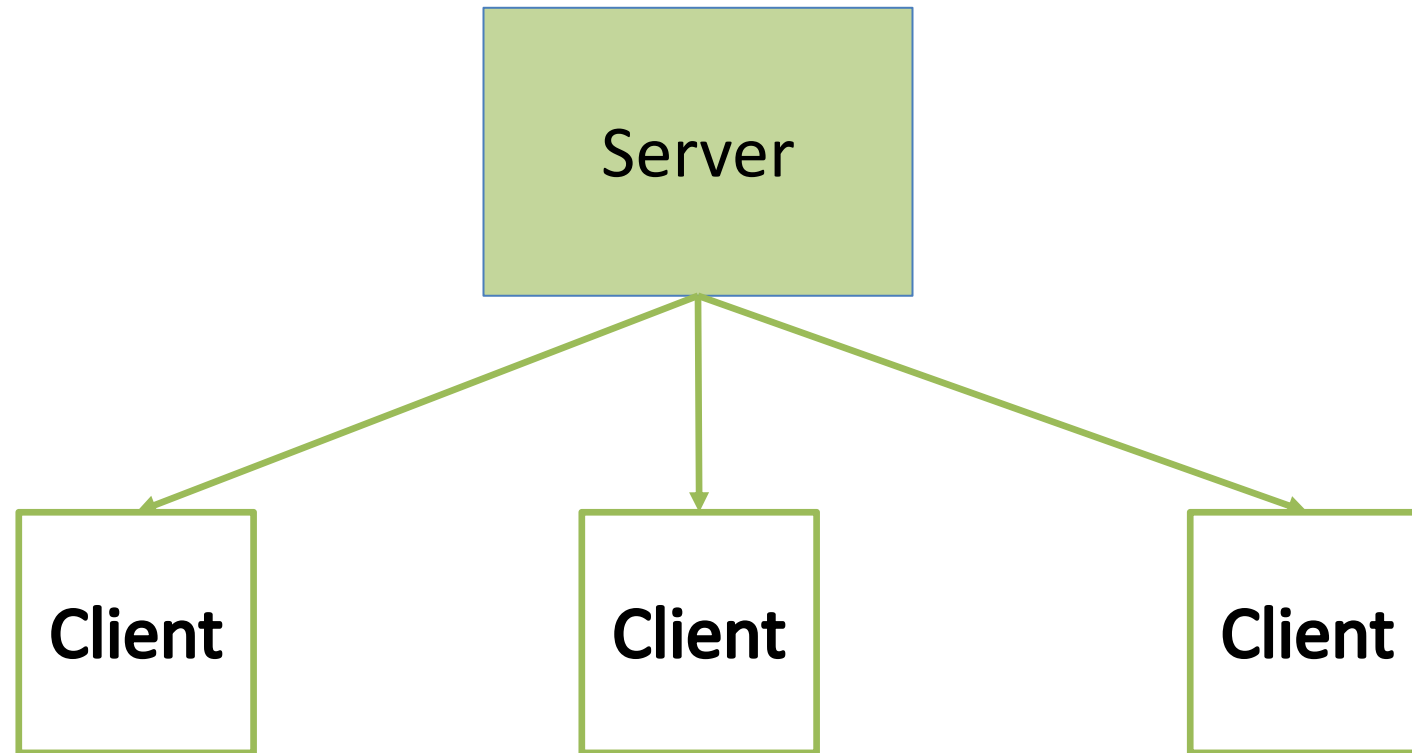
- SPDY: protocol to speed up the web: Basis for HTTP 2.0
- Request pipelining
- Compress header metadata



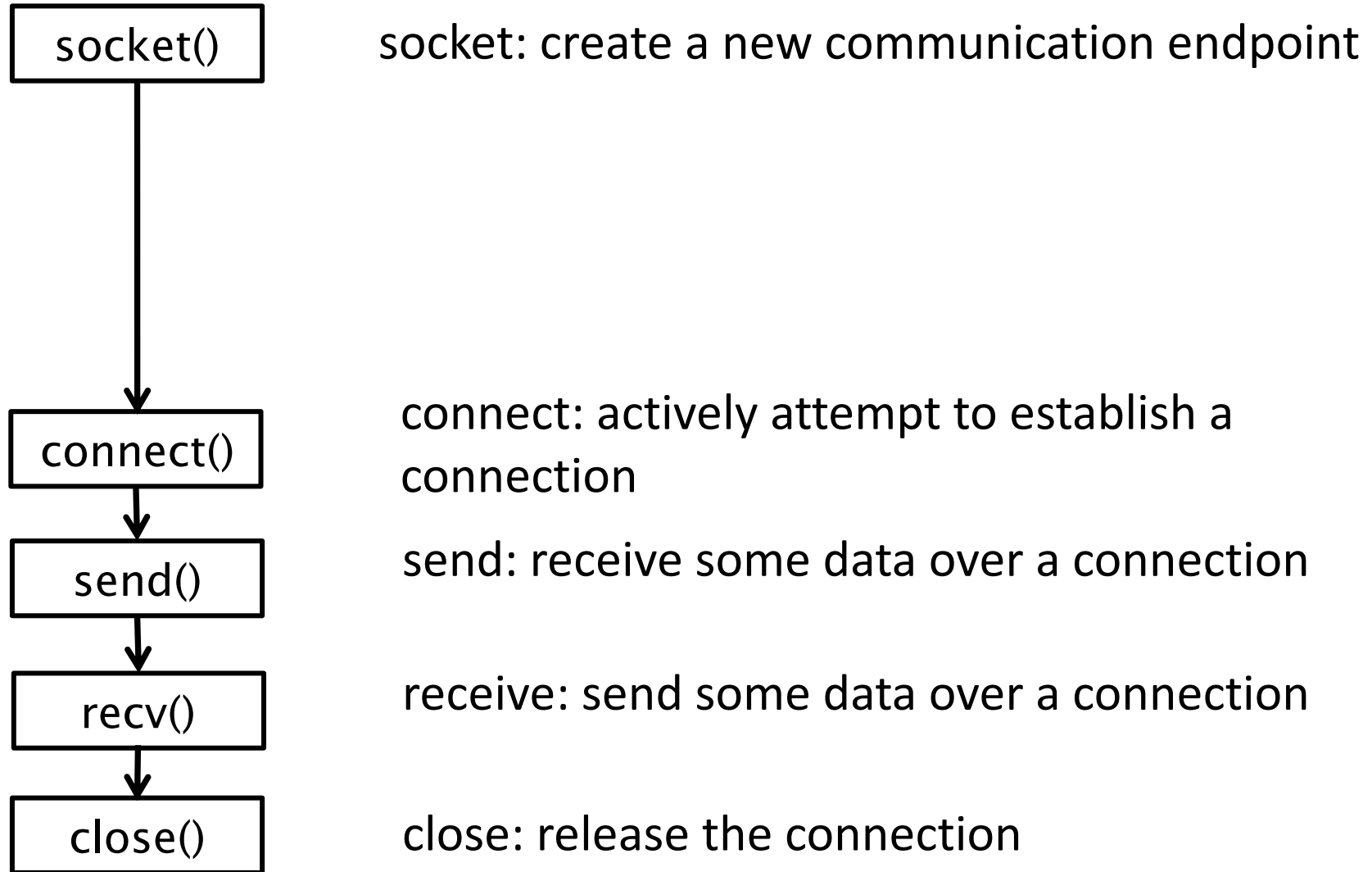
Learn more: <https://http2.github.io/>

Concurrency

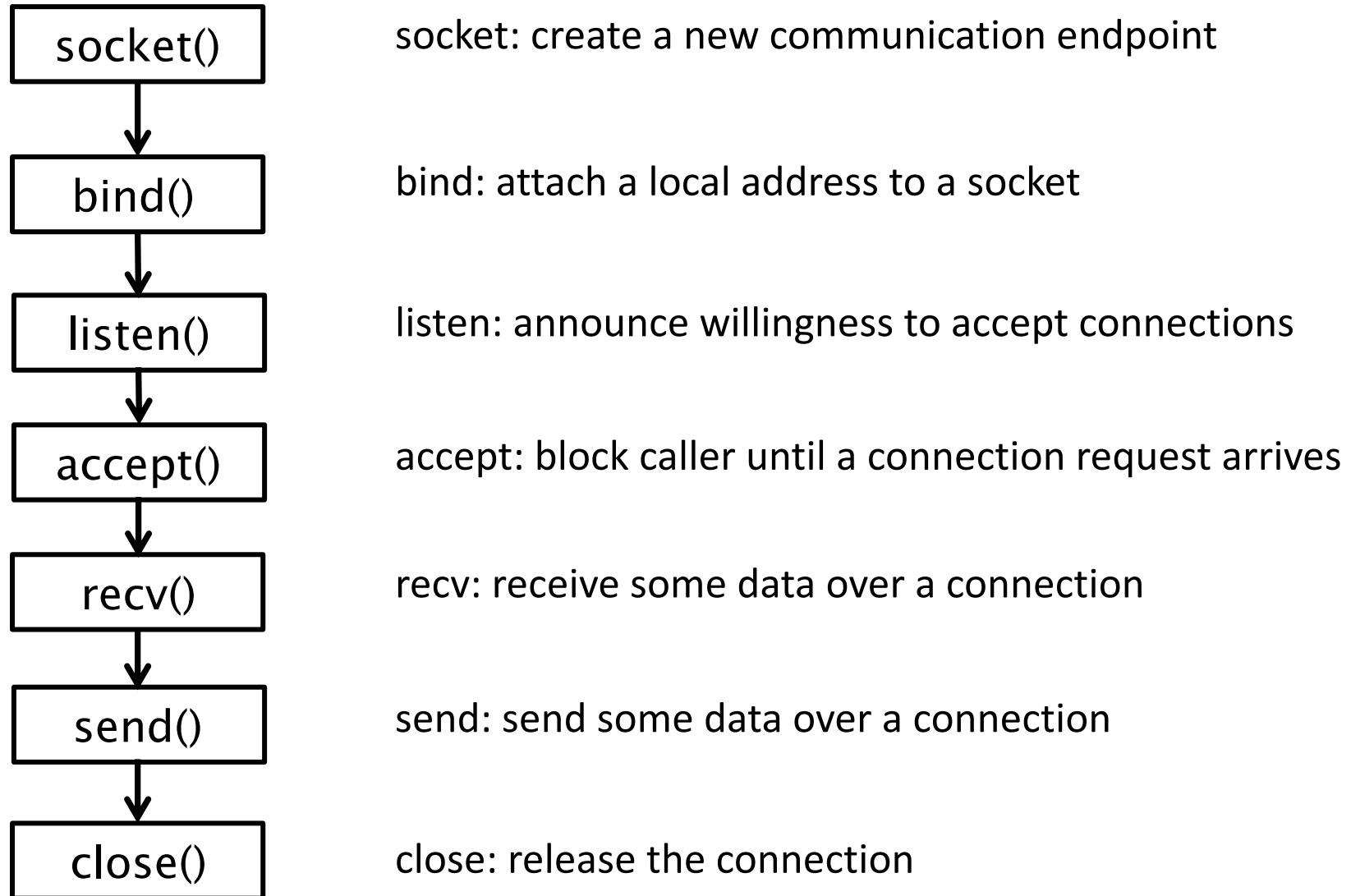
- Think you're the only one talking to that server?



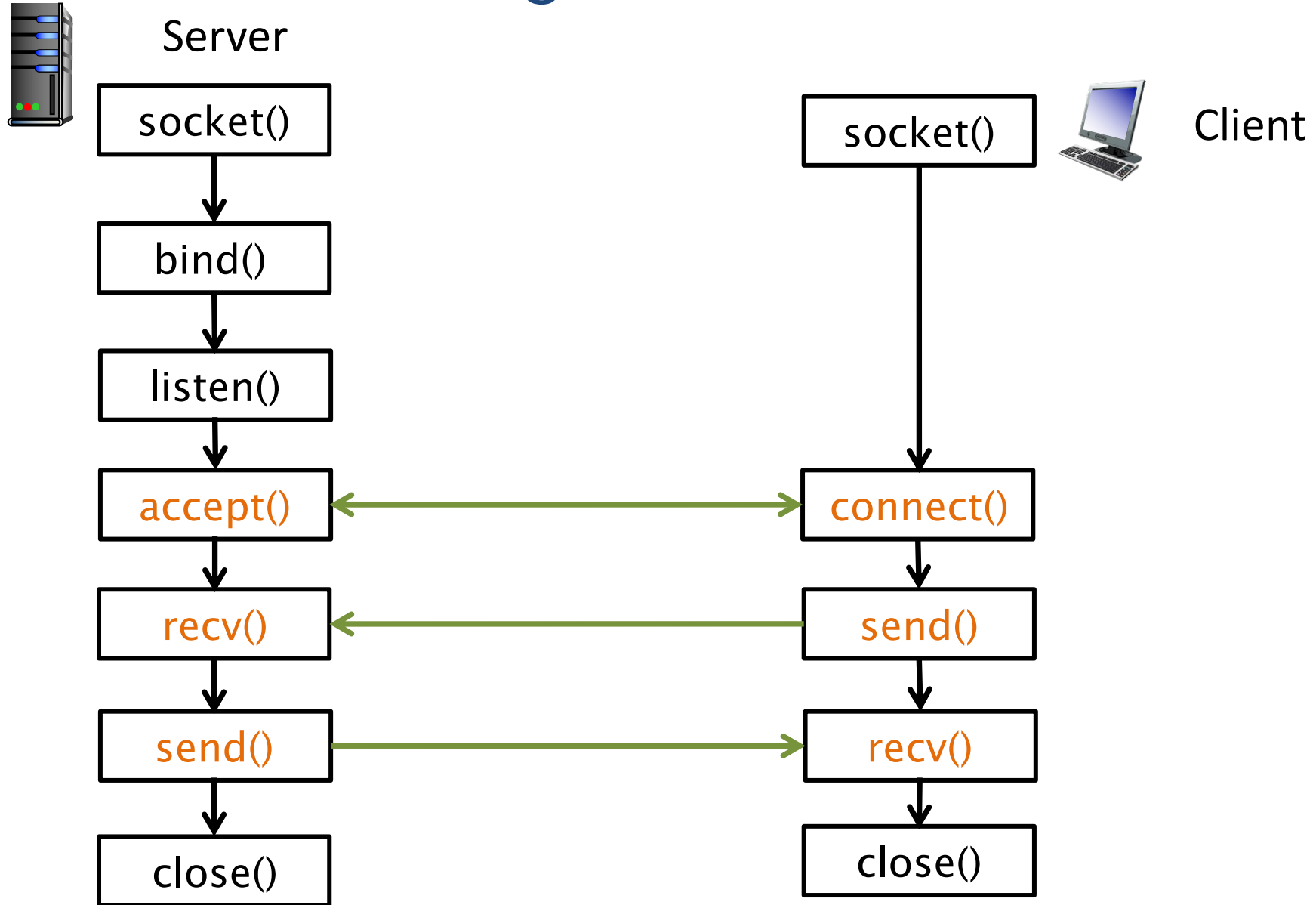
TCP Socket Procedures: for a Web Client



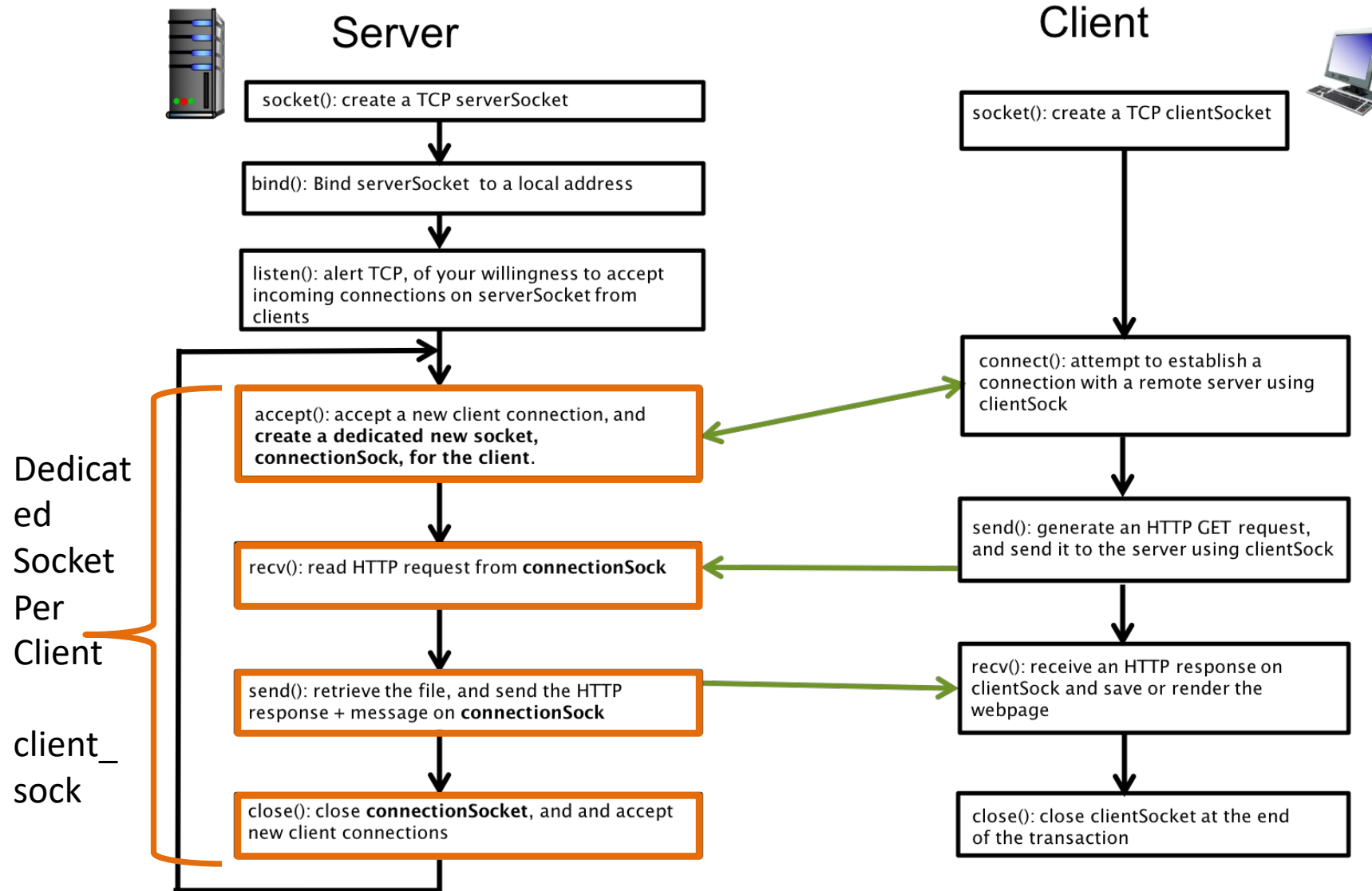
TCP socket procedures for a web server



Running a Web Server over TCP

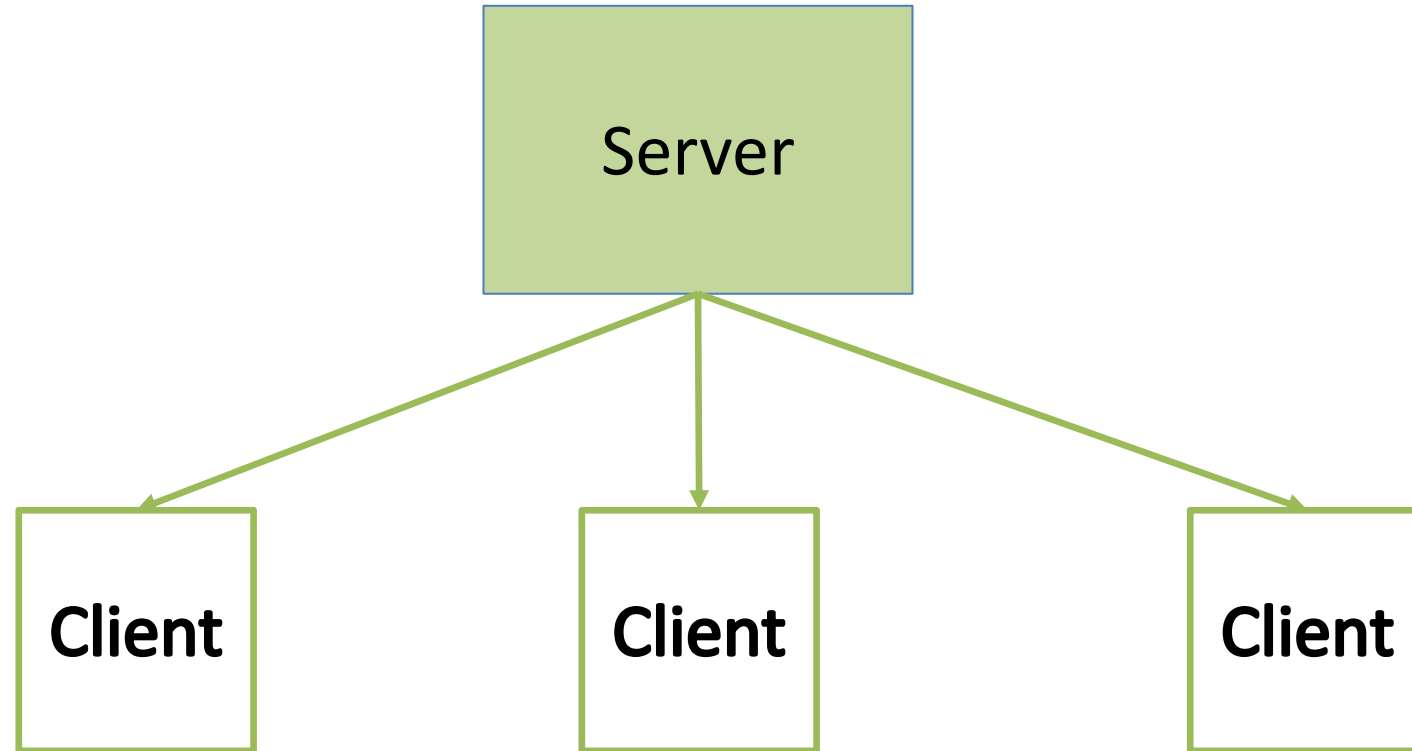


Running a Web Server



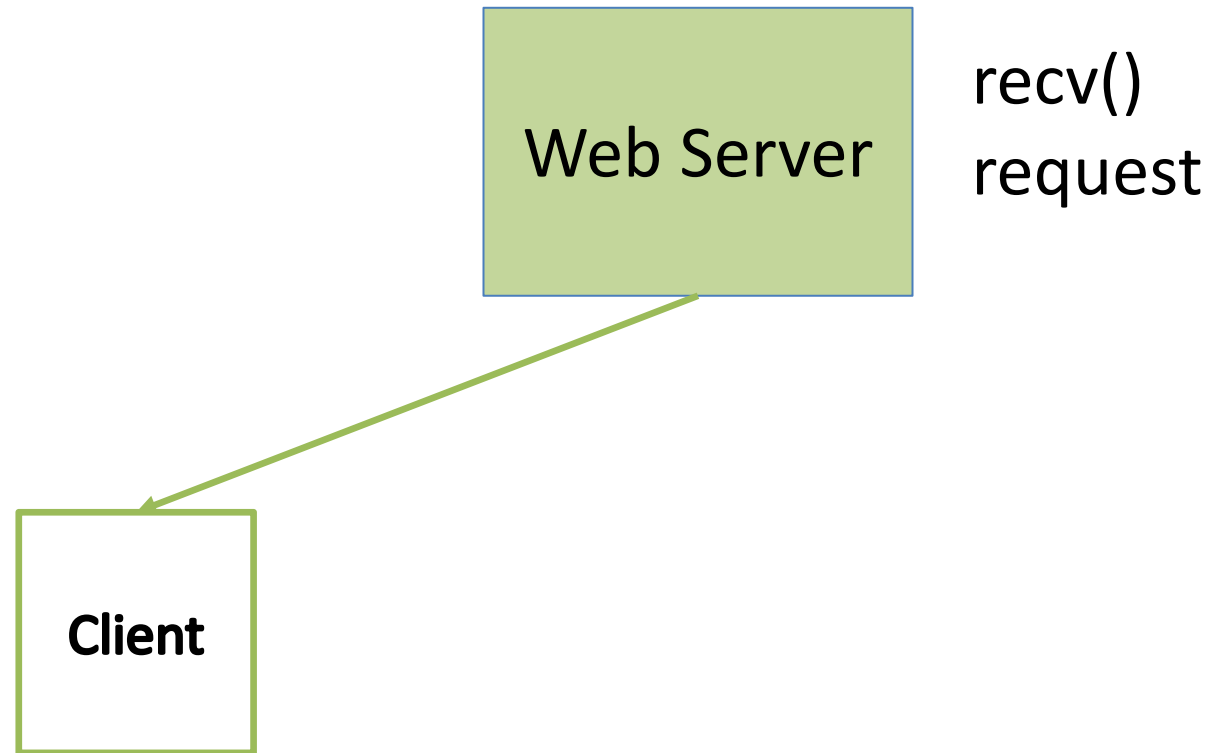
Concurrency

- Think you're the only one talking to that server?



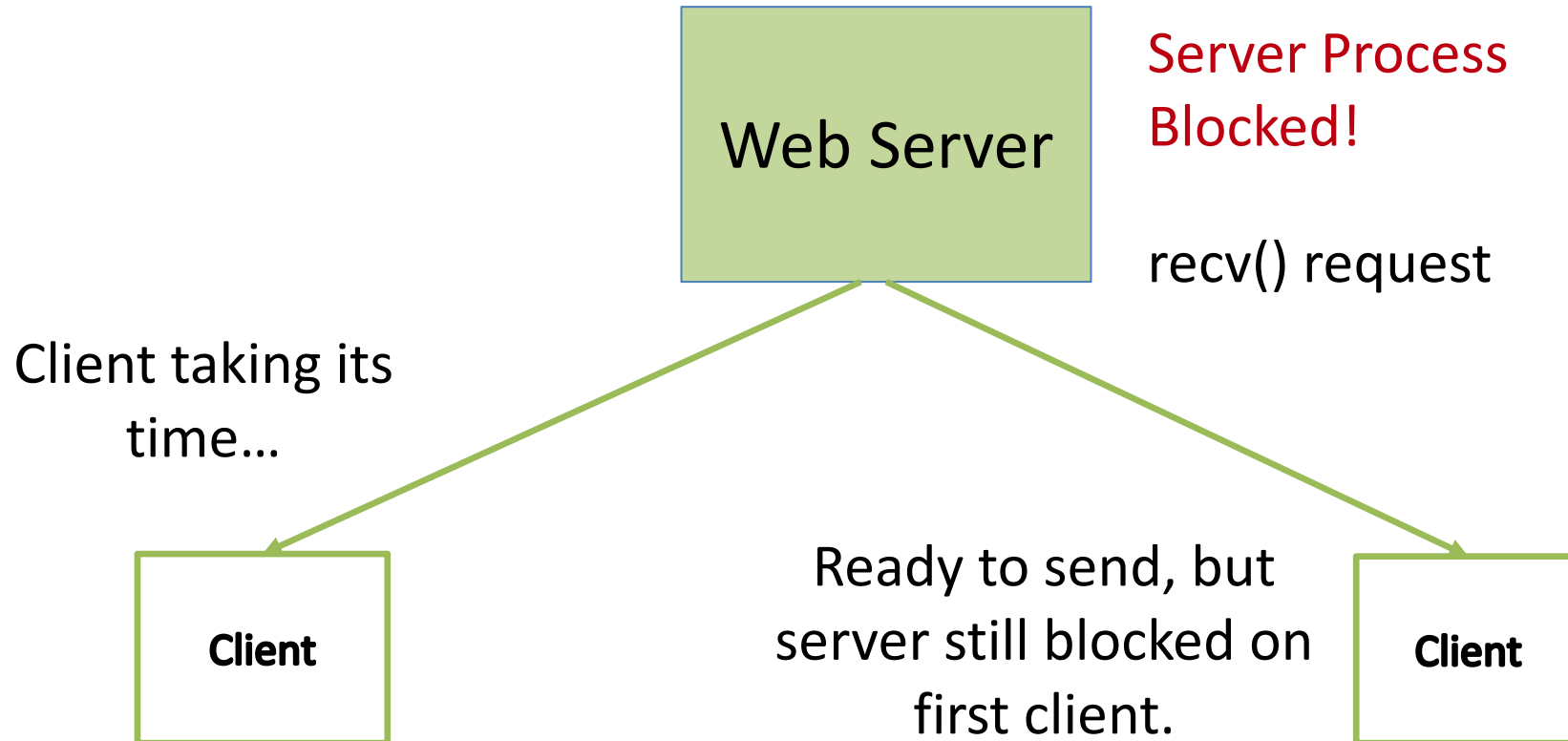
Without Concurrency

- Think you're the only one talking to that server?



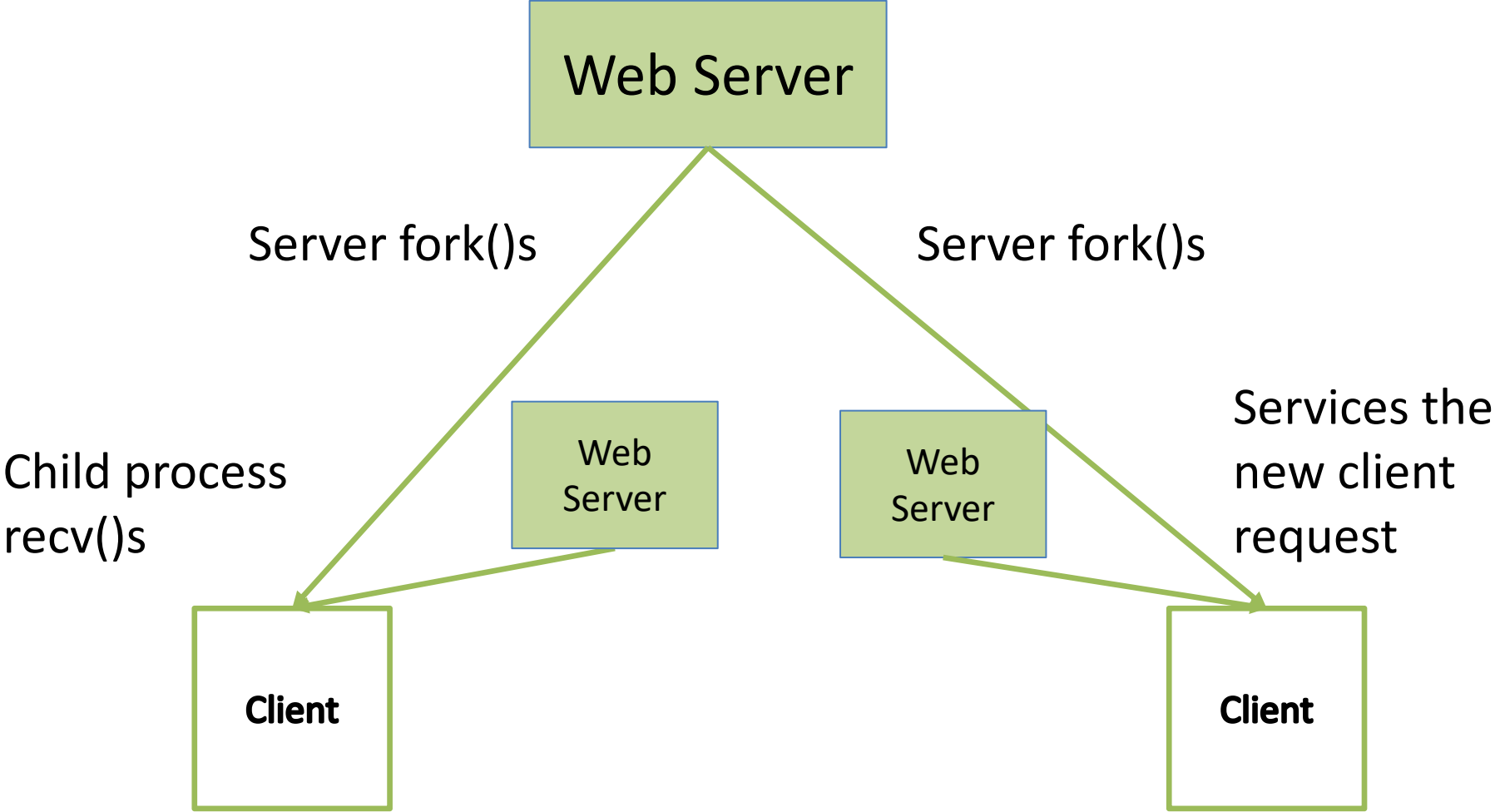
Without Concurrency

- Think you're the only one talking to that server?



If only we could handle these connections separately...

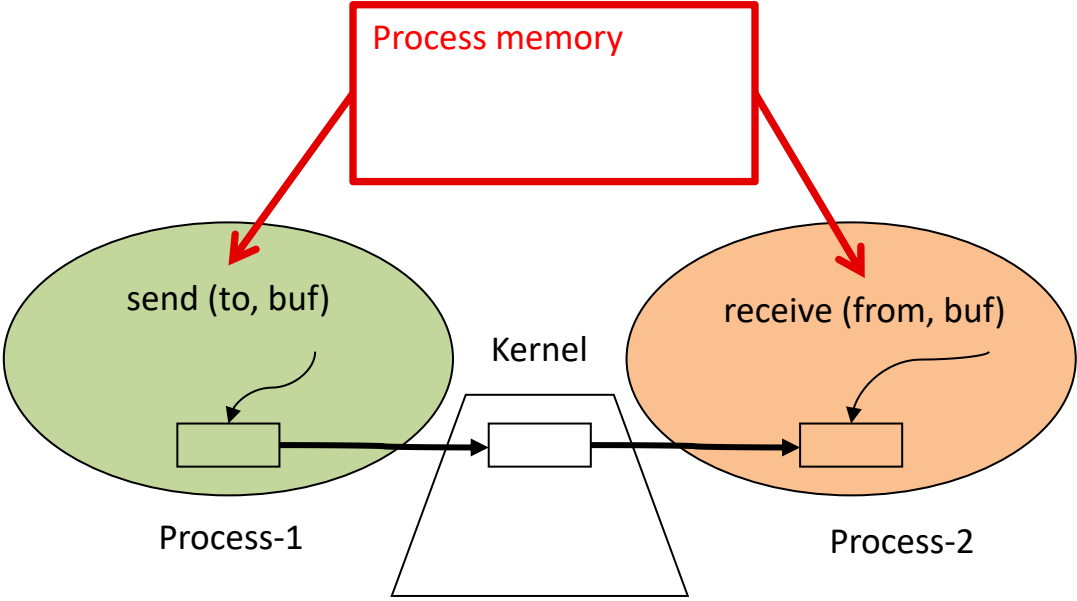
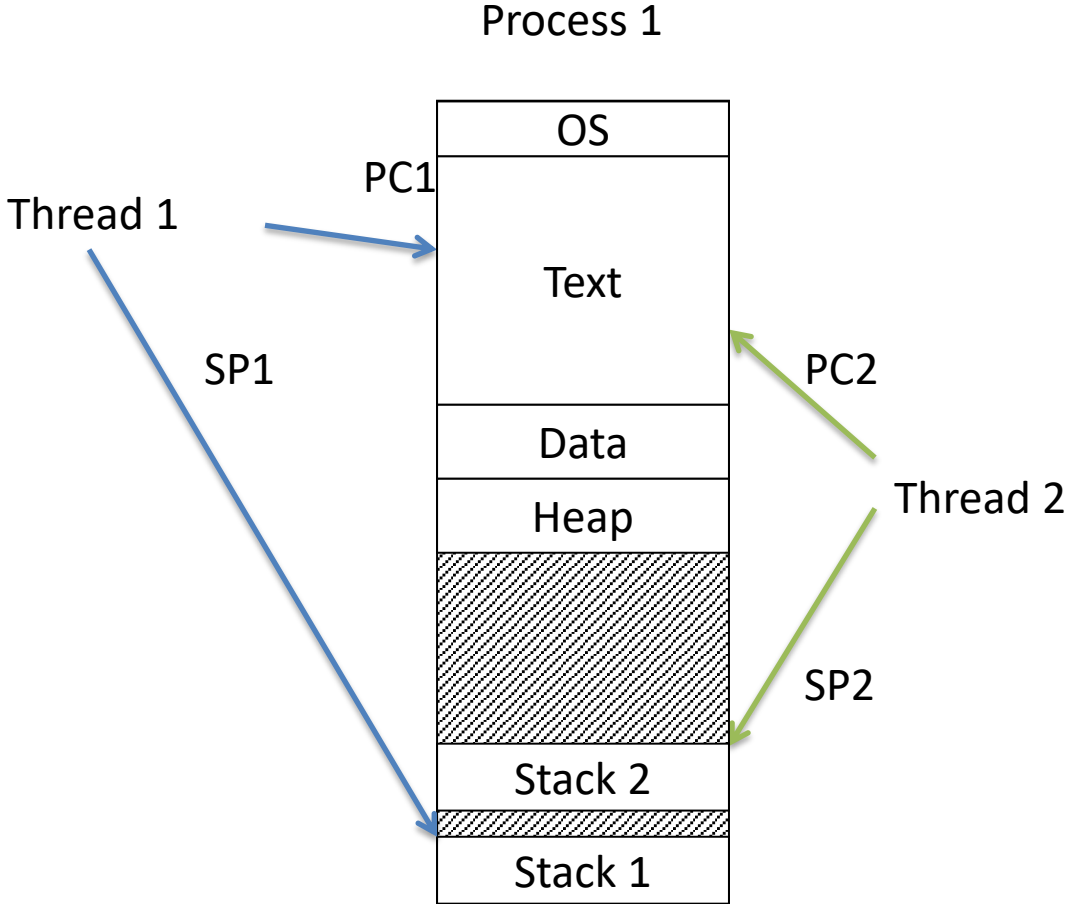
Multiple processes



Concurrent Web-servers with multiple threads/processes

- Threads (shared memory)

- Message Passing (locally)



Processes/Threads vs. Parent

Spawned Process

- Inherits descriptor table
- Does not share memory
 - New memory address space
- Scheduled independently
 - Separate execution context
 - Can block independently

Spawned Thread

- Shares descriptor table
- Shares memory
 - Uses parent's address space
- Scheduled independently
 - Separate execution context
 - Can block independently

Processes/Threads vs. Parent (More details in an OS class...)

Spawned Process

- Inherits descriptor table
- Does not share memory
 - New memory address space
- Scheduled independently
 - Separate execution context
 - Can block independently

Spawned Thread

- Shares descriptor table
- Shares memory
 - Uses parent's address space
- Scheduled independently
 - Separate execution context
 - Can block independently

Often, we don't need the extra isolation of a separate address space. Faster to skip creating it and share with parent – threading.

Threads & Sharing

- Global variables and static objects are shared
 - Stored in the static data segment, accessible by any thread
- Dynamic objects and other heap objects are shared
 - Allocated from heap with malloc/free or new/delete
- Local variables are not shared
 - Refer to data on the stack
 - Each thread has its own stack
 - Never pass/share/store a pointer to a local variable on another thread's stack

Which benefit of threads most critical in the context of running a web server?

- A. Modular code/separation of concerns.
- B. Multiple CPU/core parallelism.
- C. I/O overlapping.
- D. Some other benefit.

Both processes and threads:

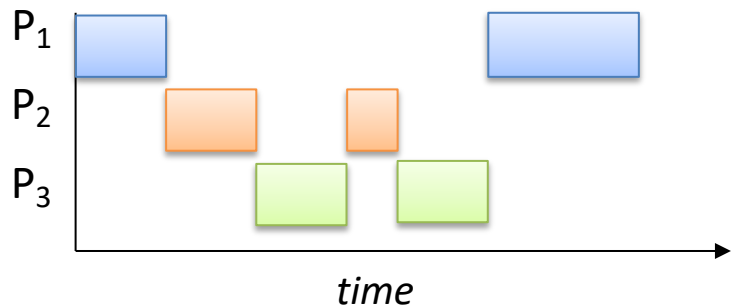
Several benefits

- Modularizes code: one piece accepts connections, another services them
- Each can be scheduled on a separate CPU
- Blocking I/O can be overlapped

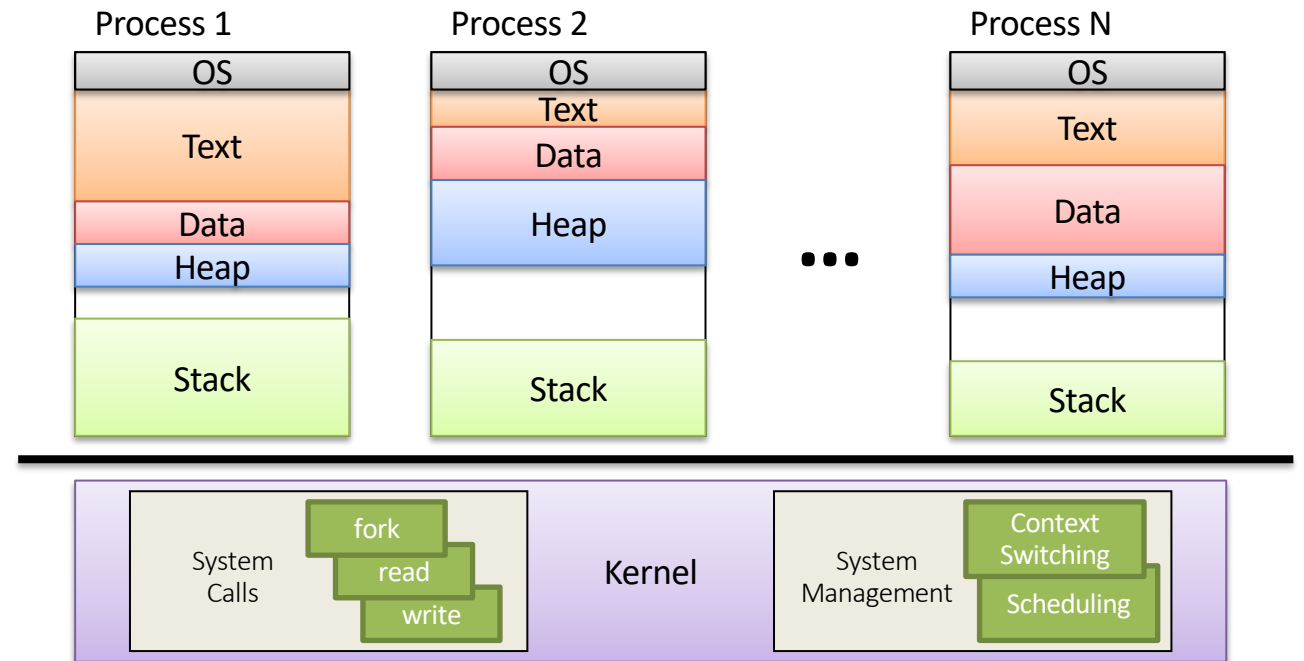
Both processes and threads

Still not maximum efficiency...

- Creating/destroying threads takes time
- Requires memory to store thread execution state
- Lots of context switching overhead



CPU: Time
Single core



Context Switching

Event-based concurrency

- Blocking: synchronous programming
 - wait for I/O to complete before proceeding
 - control does not return to the program
- Non-blocking: asynchronous programming
 - control returns immediately to the program
 - perform other tasks while I/O is being completed.
 - notified upon I/O completion

Non-blocking I/O

Event Driven I/O processing!

- Permanently for socket flag `O_NONBLOCK`
- With `O_NONBLOCK` set on a socket: No operations will block!

Non-blocking I/O

- With `O_NONBLOCK` set on a socket
 - No operations will block!
- On `recv()`, if socket buffer is empty:
 - returns -1
- On `send()`, if socket buffer is full:
 - returns -1

Will this work?

```
server_socket = socket(), bind(), listen() //non-blocking
connections = []
while (1)
    new_connection = accept(server_socket)
    if new_connection != -1,
        add it to connections
    for connection in connections:
        recv(connection, ...) // Try to receive
        send(connection, ...) // Try to send, if needed
```

Will this work?

- A. Yes, this will work efficiently.
- B. Yes but this will execute too slowly.
- C. Yes but this will use too many resources.
- D. No, this will still block.

```
server_socket = socket(), bind(), listen() //non-blocking
connections = []
while (1)
    new_connection = accept(server_socket)
    if new_connection != -1,
        add it to connections
    for connection in connections:
        recv(connection, ...) // Try to receive
        send(connection, ...) // Try to send, if needed
```

Non-blocking I/O

- With `O_NONBLOCK` set on a socket
 - No operations will block!
- On `recv()`, if socket buffer is empty:
 - returns -1
- On `send()`, if socket buffer is full:
 - returns -1

So... keep checking `send` and `recv` until they return something – waste of CPU cycles?

Event-based concurrency: select()

- Create set of file/socket descriptors we want to send and recv
- Tell **the O.S to block the process** until at least one of those is ready for us to use.
- The OS worries about selecting which one(s).

Event-based concurrency: select()

Rather than checking over and over, let the OS tell us when data can be read/written

```
client_sockets[10];
```

```
FD_SET(client_sockets) //ask OS to watch all client sockets and select those that are
```

```
select(client_sockets) are ready to recv() or send() data
```

```
for every client in client_socket:
```

```
    FD_ISSET(client, read) //return true if this client socket has any data to be received
```

```
    FD_ISSET(client, write) //return true if this client socket has any data to be sent
```

- ✓ OS worries about selecting which sockets (s) are ready.
- ✓ Process blocks if no socket is read to send or receive data.

Event-based concurrency: advantages

- Only one process/thread (or one per core)!
 - No time wasted on context switching
 - No memory overhead for many processes/threads