CS 43: Computer Networks

05: Sockets & Concurrency

September 16, 2026

Slides adapted from Kurose & Ross, Kevin Webb, Vasanta Chaganti

Announcements

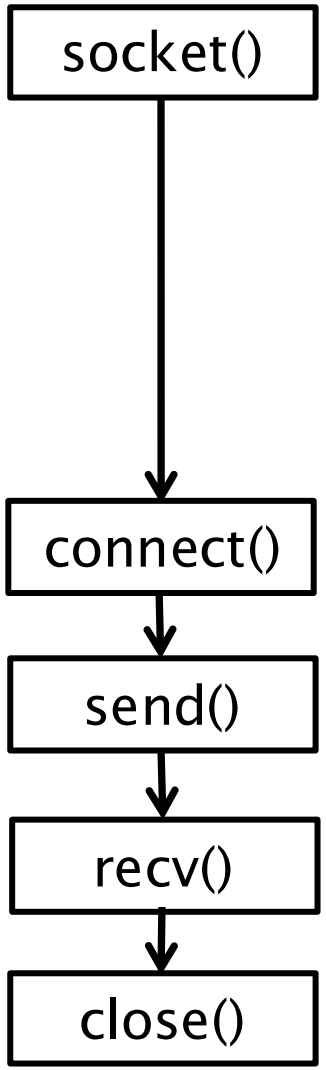
- Grades for Quiz 1 are out
  - I use positive scoring rubrics. The elements that are clicked on the rubric are the points that you received.
  - Median grade was an 8.93.
  - Feel free to come talk to me about your feedback (especially if your grade is significantly below the median). If you can't make my OH, message me on Ed and we can find a time.
- Study guide for Quiz 2 out by Thursday evening

Last time we didn’t get to a worksheet…feel free to work it on it on your own, especially if you continue to feel confused about sockets

Today: Sockets & Concurrency

🤔 How can we build web servers that can accept thousands of connections efficiently?

TCP Socket Procedures: for a Web Client



socket: create a new communication endpoint

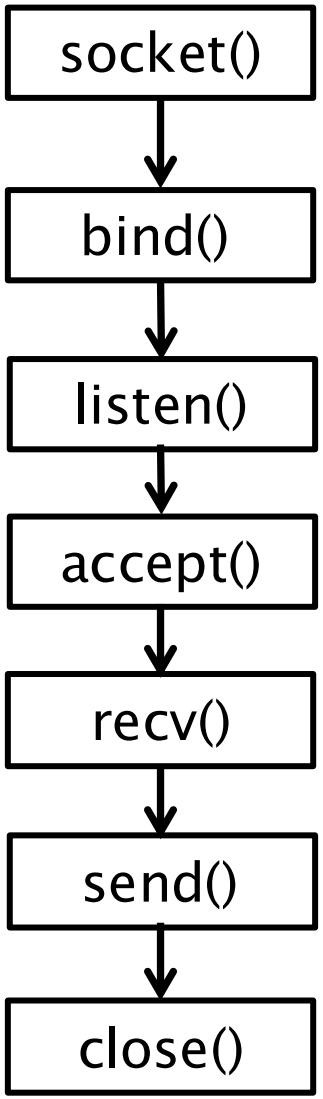
connect: actively attempt to establish a connection

send: receive some data over a connection

receive: send some data over a connection

close: release the connection

TCP socket procedures for a web server



socket: create a new communication endpoint

bind: attach a local address to a socket

listen: announce willingness to accept connections

accept: block caller until a connection request arrives

recv: receive some data over a connection

send: send some data over a connection

close: release the connection

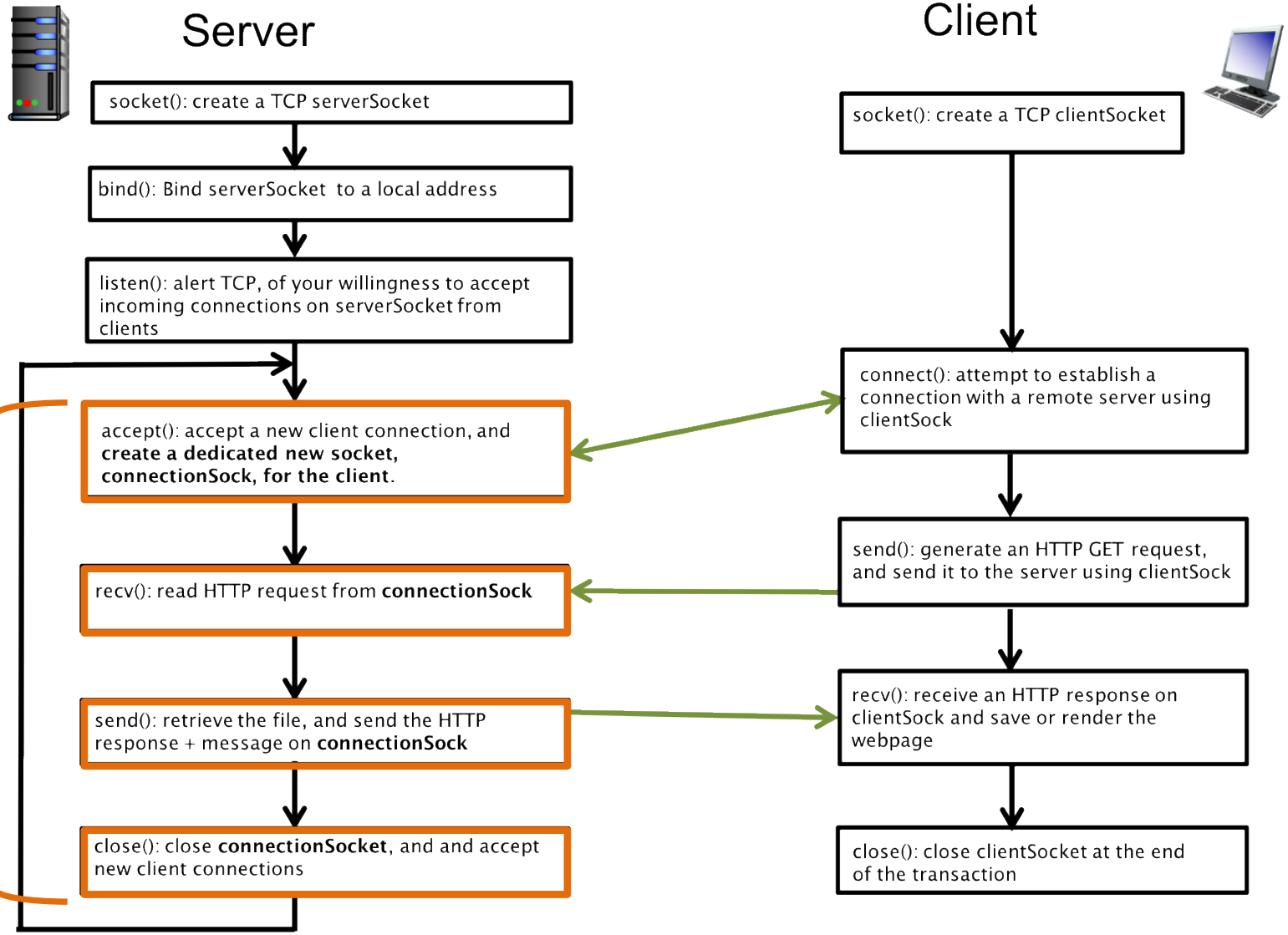
A web server is listening on port 80. Three clients connect and are accepted. How many sockets does the server process now have open?

A. One. All three clients share the socket bound to port 80
B. Three, one per client
C. Four: the one listening on port 80, plus one per client
D. Three, and port 80 is released once the first client connects
E. I don't know

A web server is listening on port 80. Three clients connect and are accepted. How many sockets does the server process now have open?

A. One. All three clients share the socket bound to port 80
B. Three, one per client
C. Four: the one listening on port 80, plus one per client
D. Three, and port 80 is released once the first client connects
E. I don't know

Running a Web Server

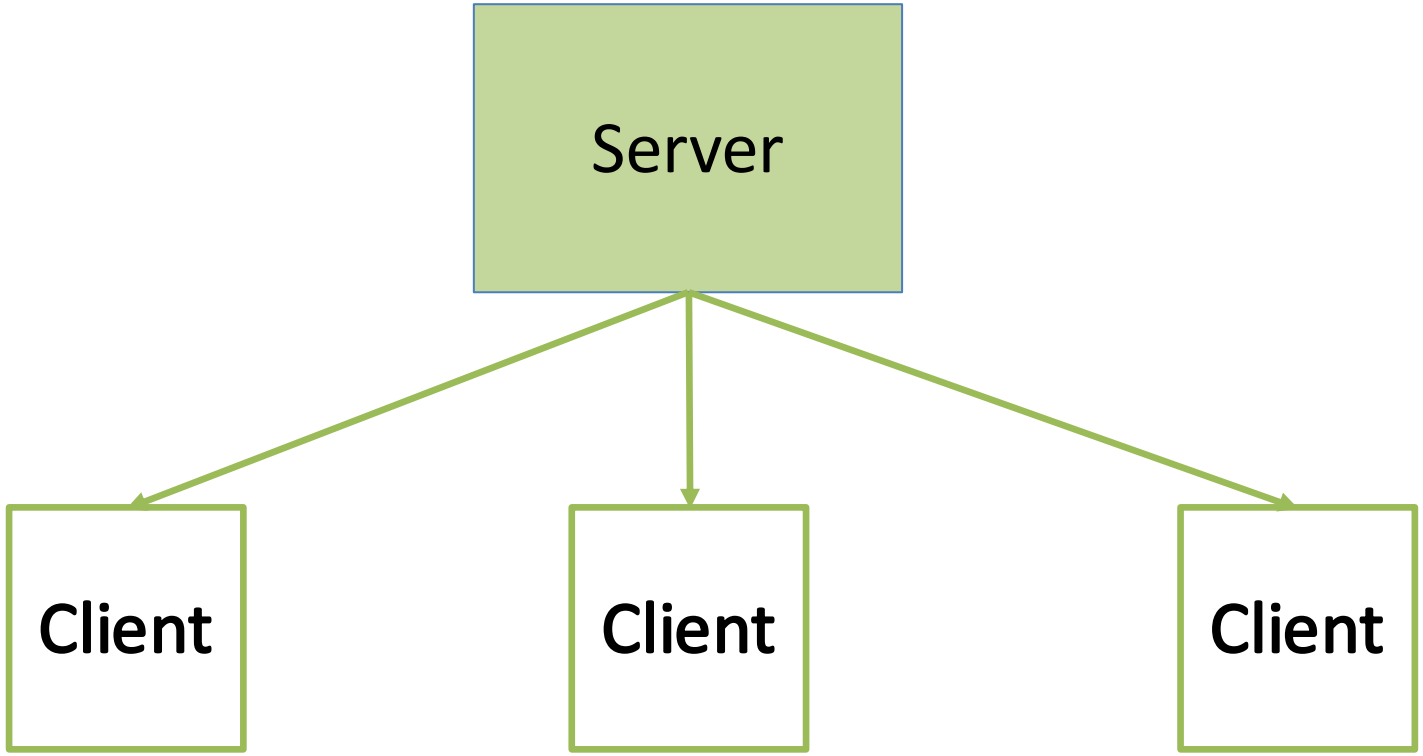


Dedicated Socket Per Client

connectionSock

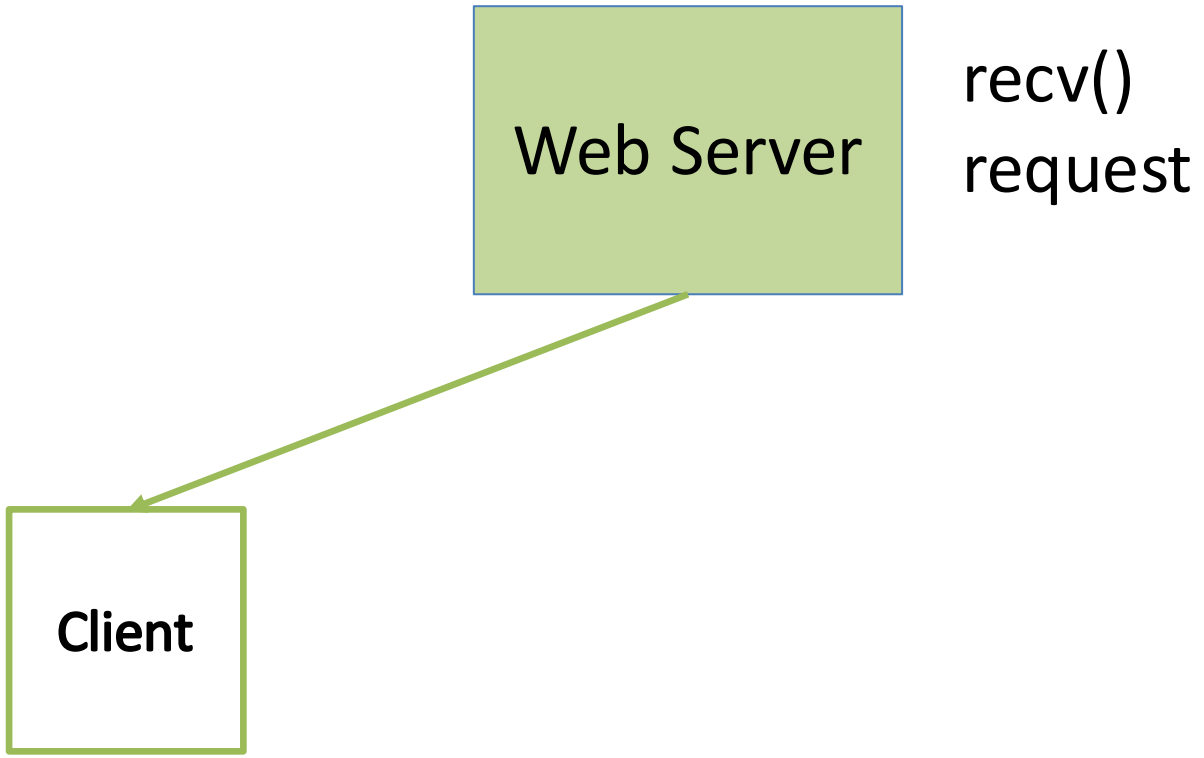
Concurrency

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



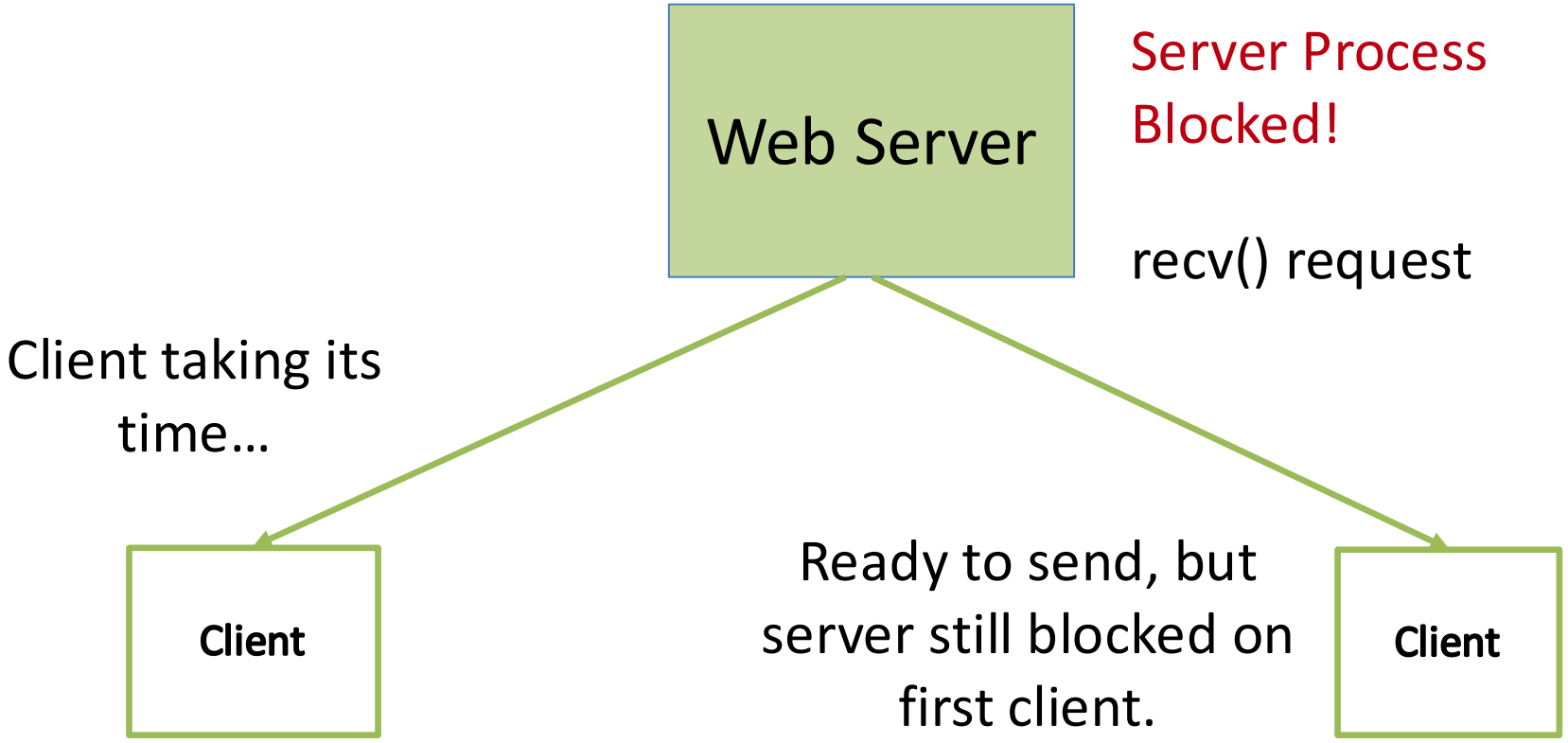
Without Concurrency

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



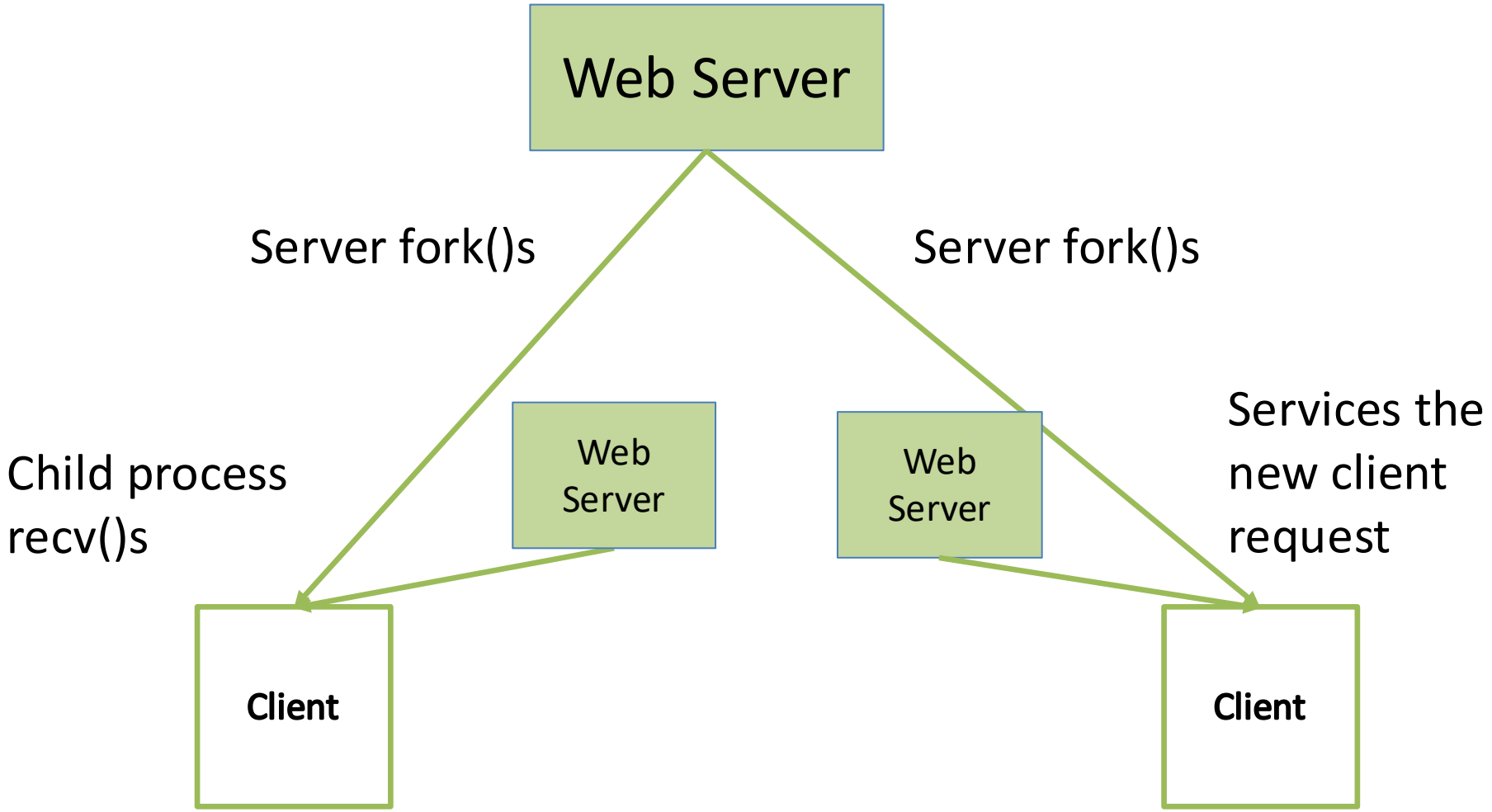
Without Concurrency

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



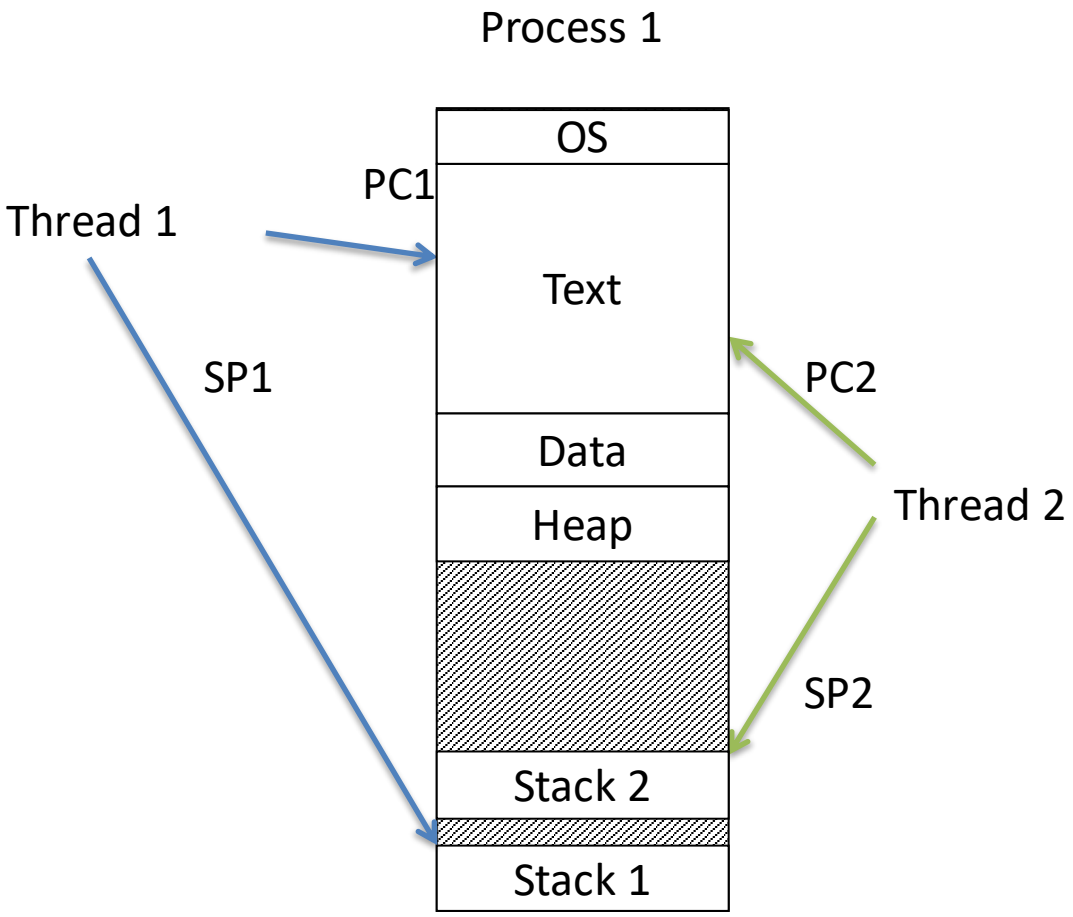
If only we could handle these connections separately…

Multiple processes

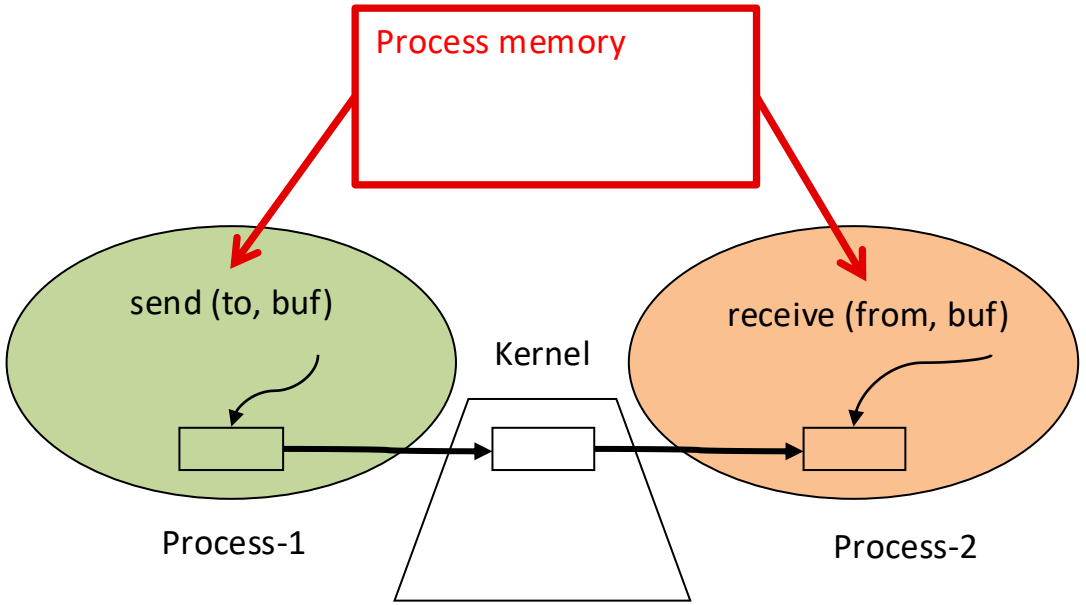


Concurrent Web-servers with multiple threads/processes

- Recall from CS 31: 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.

Recall from CS31: 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:

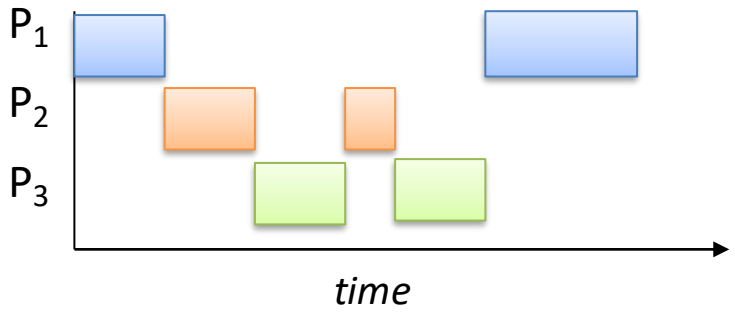
Several benefits

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

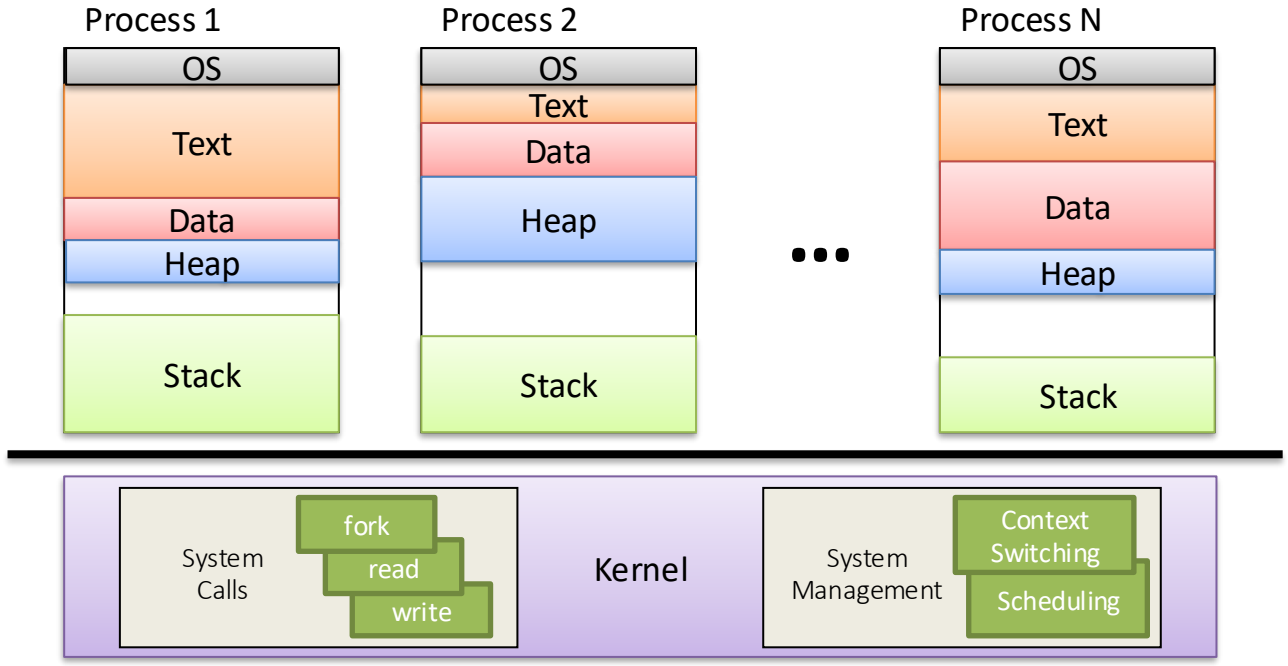
Both processes and threads

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 and errno=EAGAIN or EWOULDBLOCK
- On send(), if socket buffer is full:
  - returns -1, and errno=EAGAIN or EWOLDBLOCK

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
```

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 and errno=EAGAIN

- On send(), if socket buffer is full:
  - returns -1, and errno=EAGAIN

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

Today: Sockets & Concurrency

🤔 How can we build web servers that can accept thousands of connections efficiently?

Two options:

- Use threads or forking processes
- Use non-blocking I/O and event-based concurrency