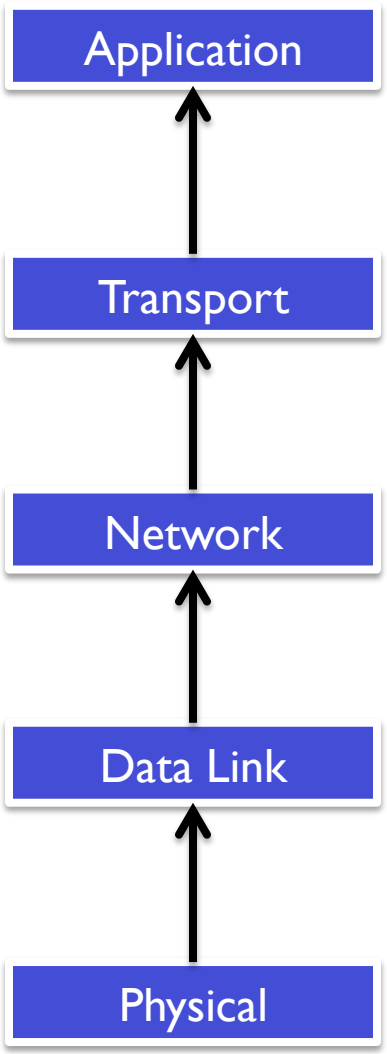
Networking 2: The Lecture

CS 241

April 16, 2014

University of Illinois

The Internet's Protocol Stack



Anything you want...

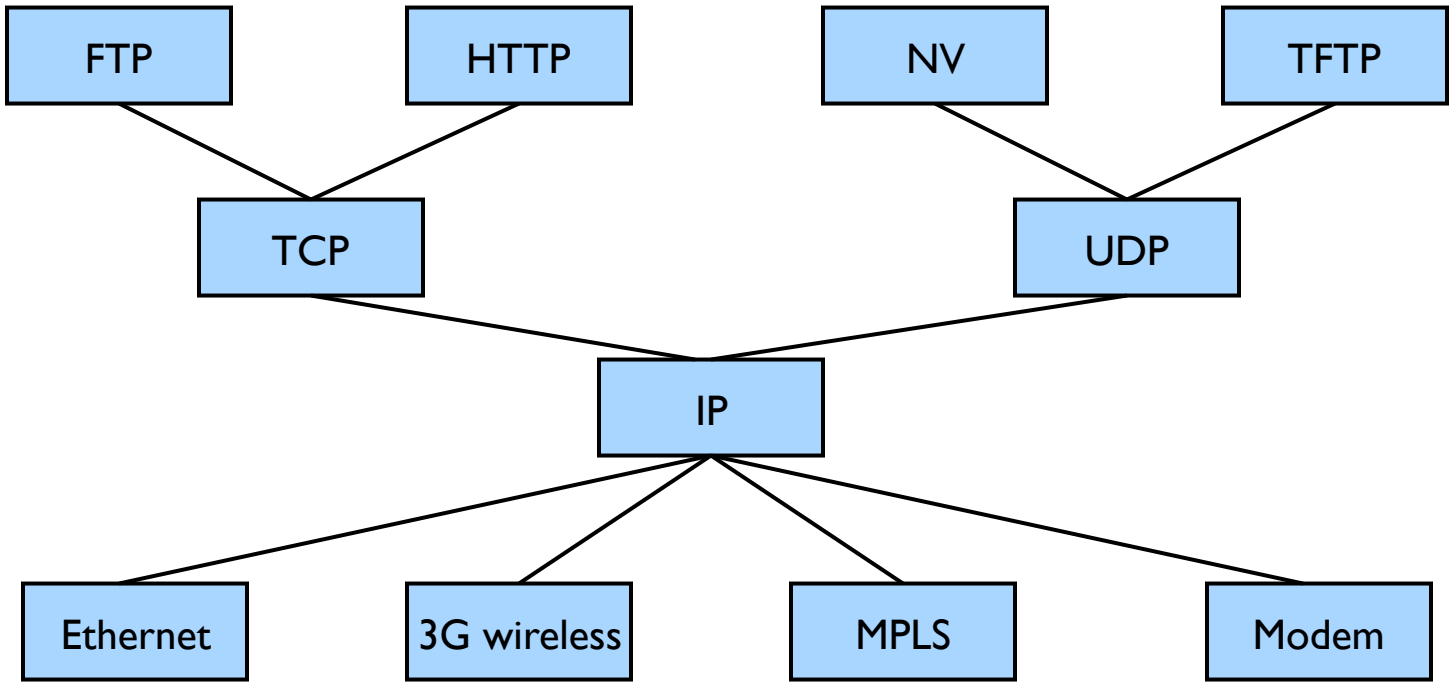
Process-to-process communication

Host-to-host packet delivery across the Internet

Host-to-host packet delivery across a link

Host-to-host bit delivery across a link

Internet Architecture: The "Hourglass" Design



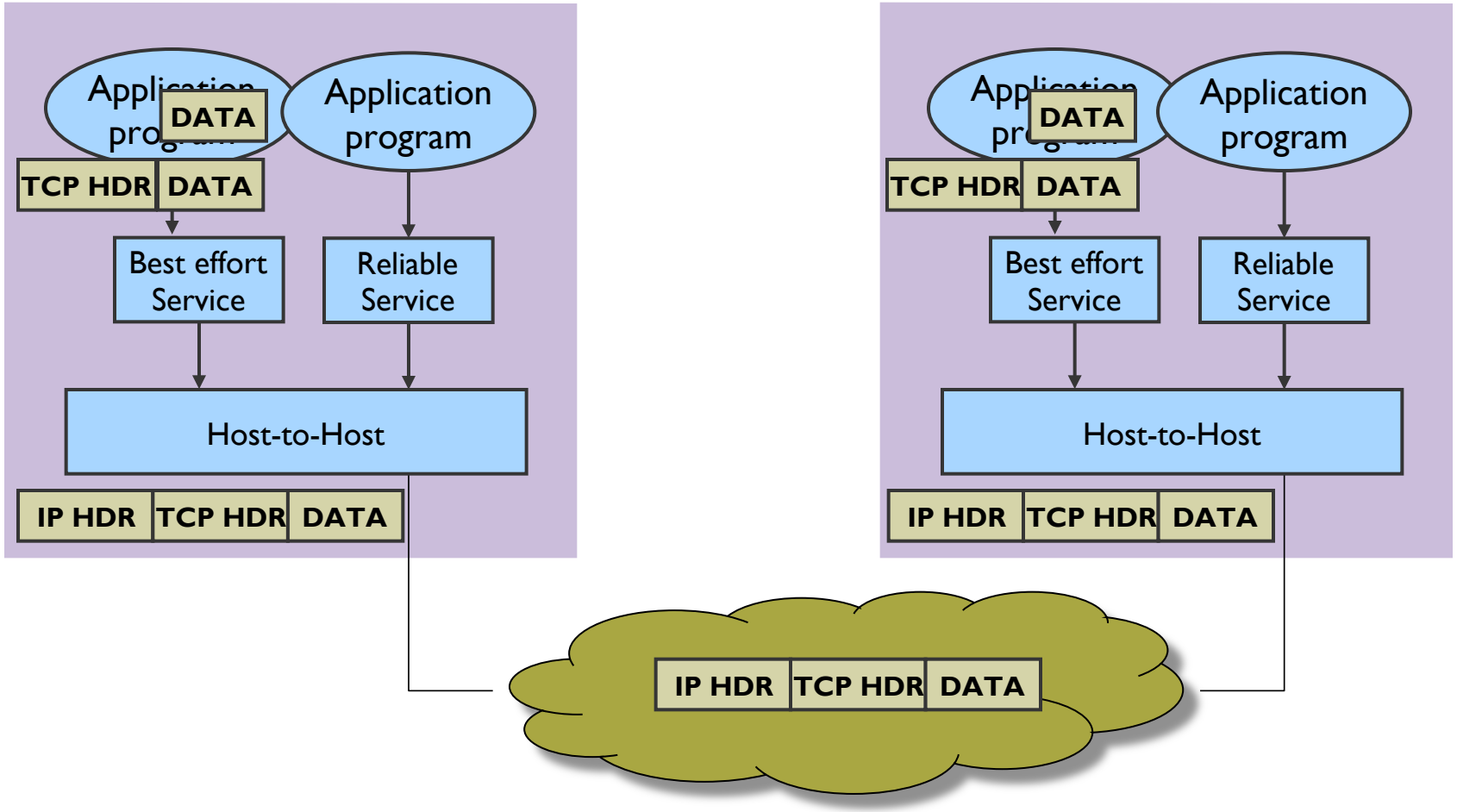
Why layering?

It's all about modularity

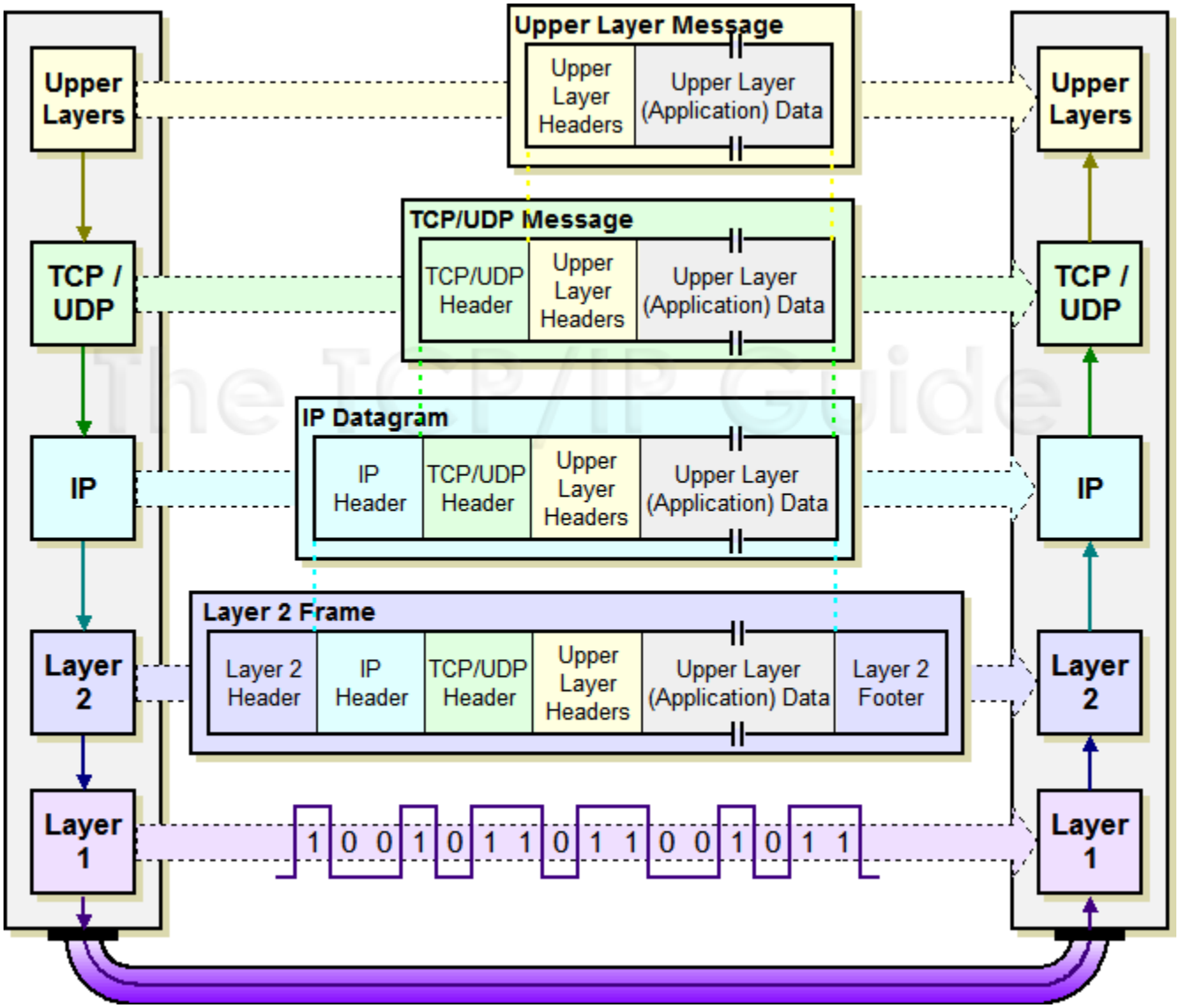
- Eases maintenance, updating of system
- Change of implementation of layer's service transparent to rest of system
- e.g., change in transmission medium (Layer 0) has no effect on network protocol or applications

What other examples of layering have we seen?

Encapsulation: Traveling through the layers



Network Packet Encapsulation



Source: http://www.tcpipguide.com/free/t_IPDatagramEncapsulation.htm

Understanding IP

The network layer provides "host-to-host" connectivity.

- In IP, done via **IP Addresses**
  - Globally unique 32-bit numbers
  - Usually written as four 8-bit integers: **127.0.0.1**
  - **IPv6:** 128-bits, written as eight sets of 16-bit hexadecimal numbers (ex: 2001:0DBB:AC10:FE01:0000:0000:0000:C3D4 == 2001:0DBB:AC10:FE01::C3D4)

- IP addresses are hard to remember!
  - **Domain names** associate easy-to-remember names that can be translated to IP addresses via the DNS protocol.

Understanding TCP

TCP provides:

- Port number to identify a process
- Reliable delivery of packets
- Check data integrity via checksums
- Pipe abstraction (stream)
- Congestion control
- Flow control

TCP doesn't provide:

- Structure to data
- Security / encryption

*…while the **session** is active.*

Understanding UDP

UDP provides:

- Port number for process-to-process communication
- Lower-level access to the network via discrete packets
  - Greater speed and flexibility

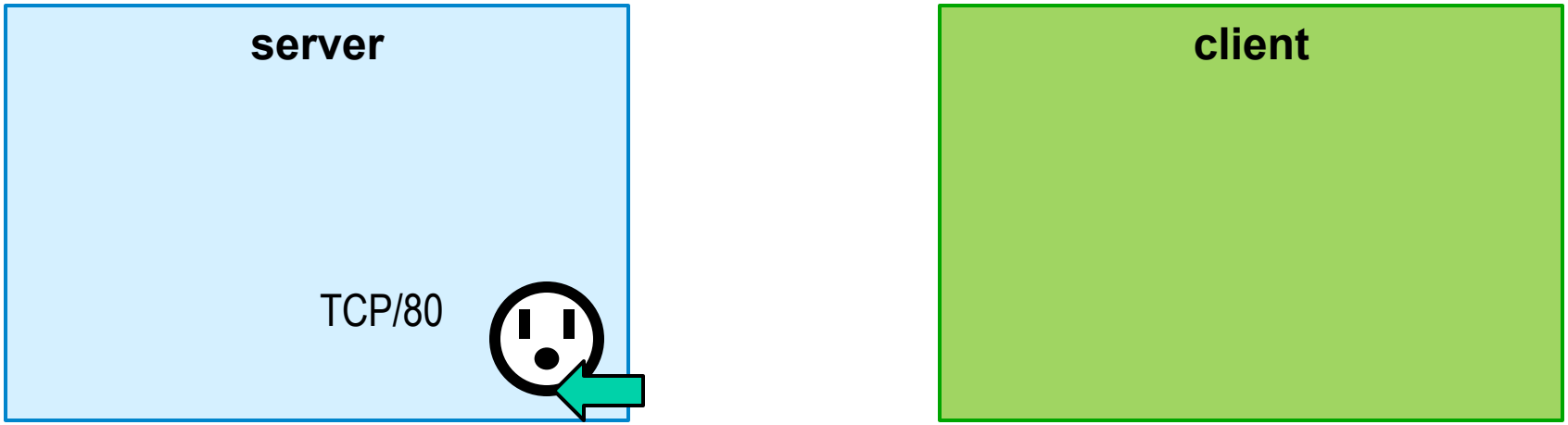
UDP doesn't provide:

- Everything else

Creating a TCP session

Server:

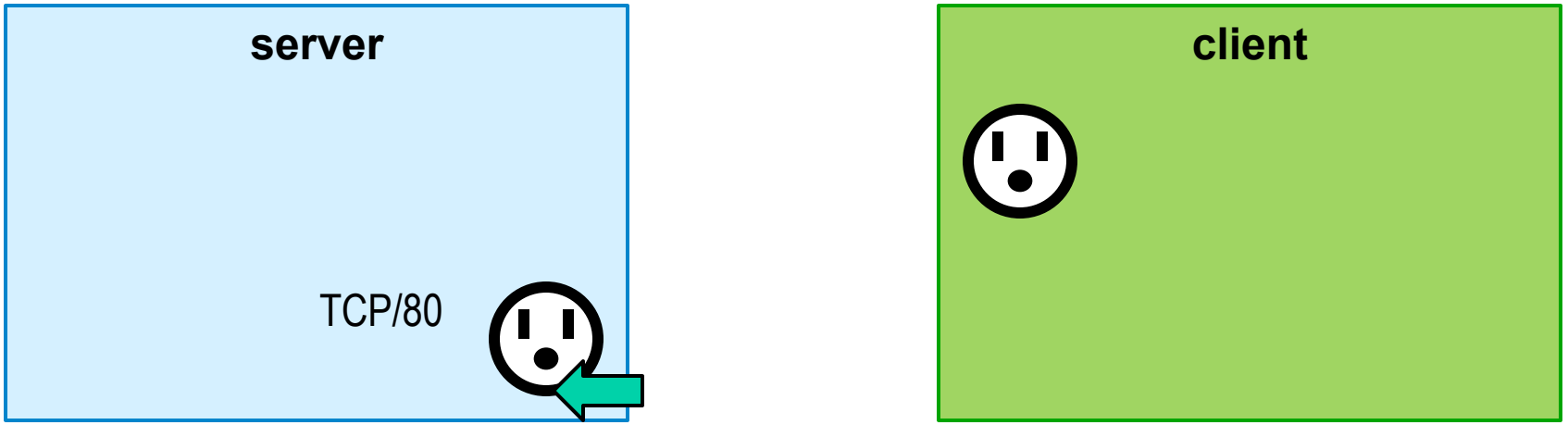
- Creates a socket to listen for incoming connections.
- Must listen on a specific protocol/port.



Creating a TCP session

Client:

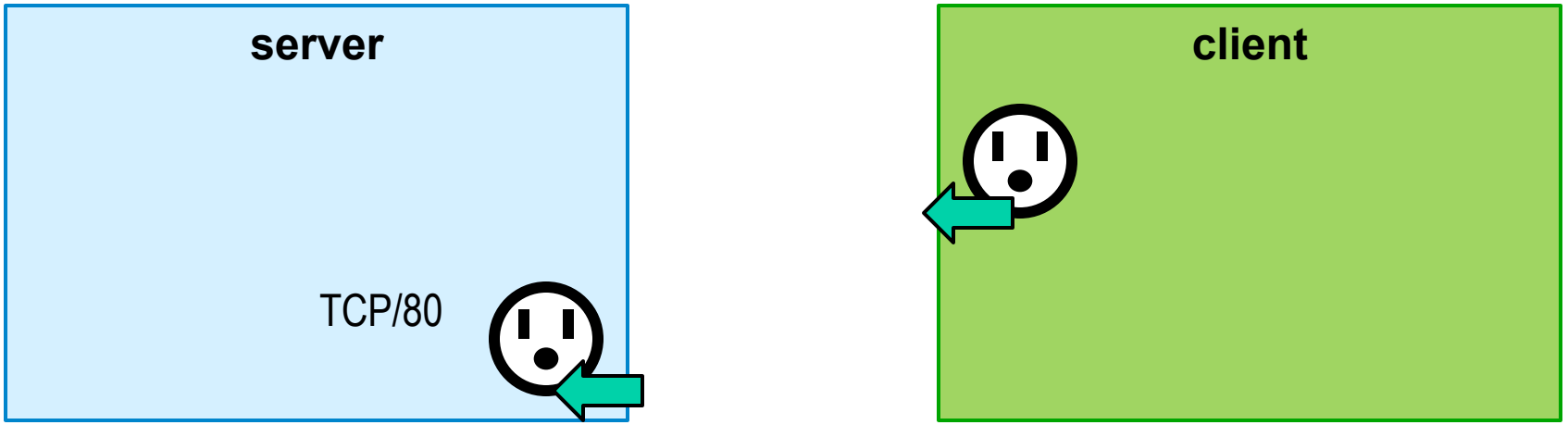
- Creates a socket to connect to a remote computer.



Creating a TCP session

Client:

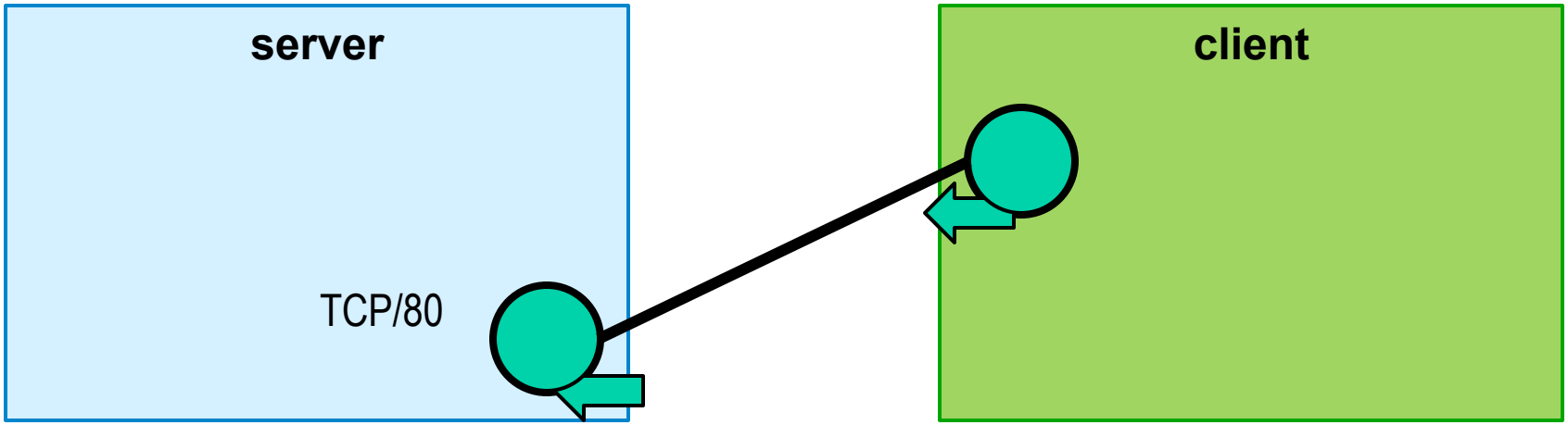
- Requests a connection to TCP port 80 on 74.125.225.70.



Creating a TCP session

Server:

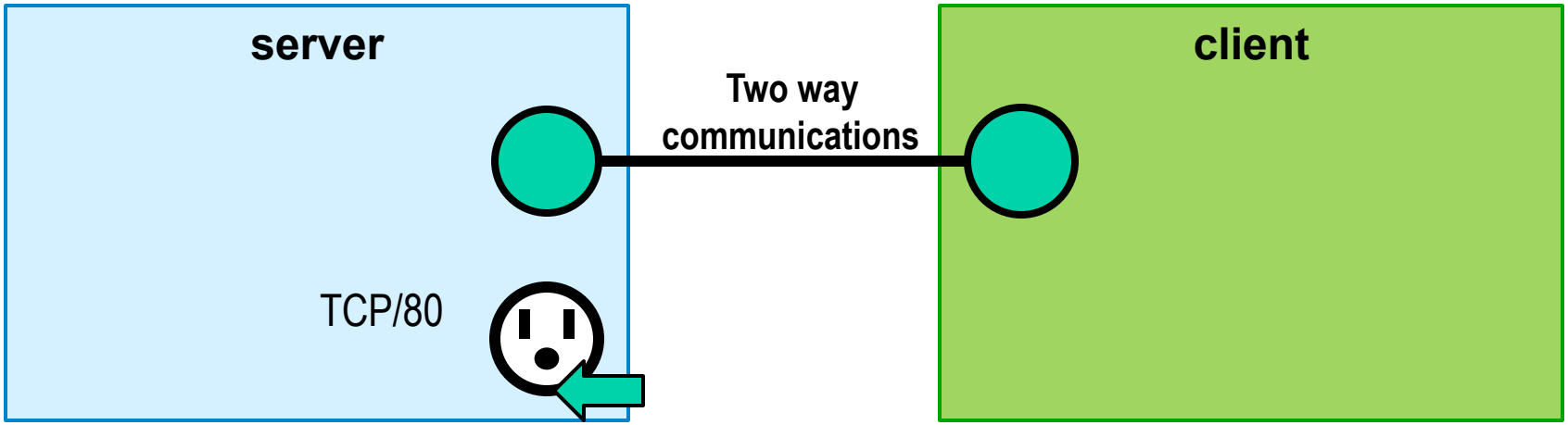
- Accepts the connection.



Creating a TCP session

Server:

- Spawns a new socket to communicate directly with the newly connected client.
- Allows other clients to connect.



Network Vocabulary

Socket Address

- Complete identification of the socket you are connecting to. Made up of three pieces:
  - Protocol (ex: TCP)
  - Network Address (ex: 127.0.0.1)
  - Port Number (ex: 80)

Port Number

- Globally shared system resource, 16-bit integer (0 to 65,535)
- A port number can only be used by one process at a time on the entire system
- Ports below 1024 are "special"
  - Associated with particular applicaitons
  - Use often requires elevated privileges (e.g. root)

Network socket

A network socket is stream-based IPC.

Similar to a pipe:

- Uses the file descriptor interface
- Stream-based, not segment- or message-based

Different from a pipe:

- The file descriptor is bi-directional (read and write)
- Reliability based on the transport protocol used
- Special type of “server socket” that listens for incoming connections from remove hosts and does not transmit any application data!

Creating a network socket (client and server)

`socket()`: Create an endpoint for communication

```
int socket(int network_protocol,
           int transport_protocol,
           int sub_protocol)
```

```
IP: AF_INET          IPv6: AF_INET6
TCP: SOCK_STREAM     UDP: SOCK_DGRAM
```

Setting up a server socket

`getaddrinfo()`: network address translation

- Translates a hostname (IP address or domain name), port, and protocol into a socket address struct.

`bind()`: binds an socket address to a socket

- Required in order to know what port number your socket will be listening for new connections

`listen()`: places the socket in a listening state

`accept()`: accept a communication on a socket

- ```
  int accept(int sockfd,
             struct sockaddr *addr,
             socklen_t *addrlen);
  ```
  ```

Setting up a client socket

connect(): initiate a connection on a socket

- `int connect(int sockfd,`
  `struct sockaddr *addr,`
  `socklen_t *addrlen);`