CLIENT SERVER ARCHITECTURE

EN.600.444/644

Spring 2019

Dr. Seth James Nielson

COMPUTING 1960-1980 (ISH)

NETWORK

"DUMB" TERMINAL

MAINFRAME

COMPUTING 1980-2000 (ISH)

COMPUTING 2000 – PRESENT

NETWORK

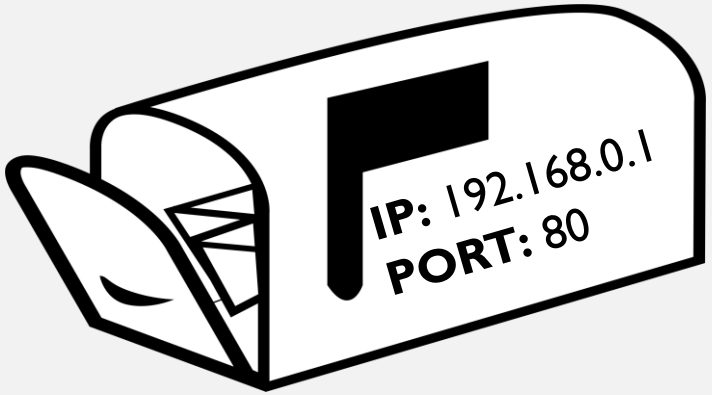
GENERAL IDEAS BEHIND CLIENT-SERVER

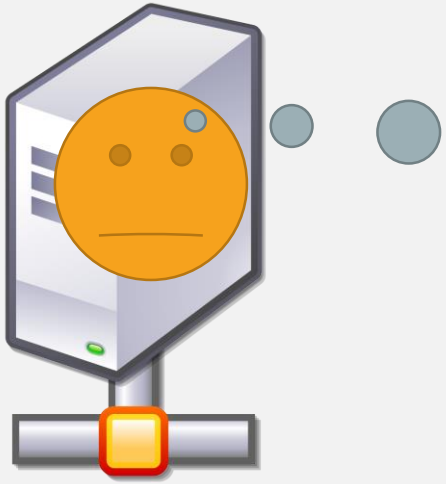
- Put a bunch of resources in a high-performance, centralized machine
- Clients can be much "dumber" *by comparison*
- Much more efficient
  - Sharing data between devices, applications, and people (and marketing)
  - Access from multiple locations (including hackers!)
  - Time-sharing a central machine is more scalable and cost-effective

SERVER ABSTRACTION

SERVER ***LISTENS*** FOR INCOMING REQUESTS

PREVIEW OF TCP/IP





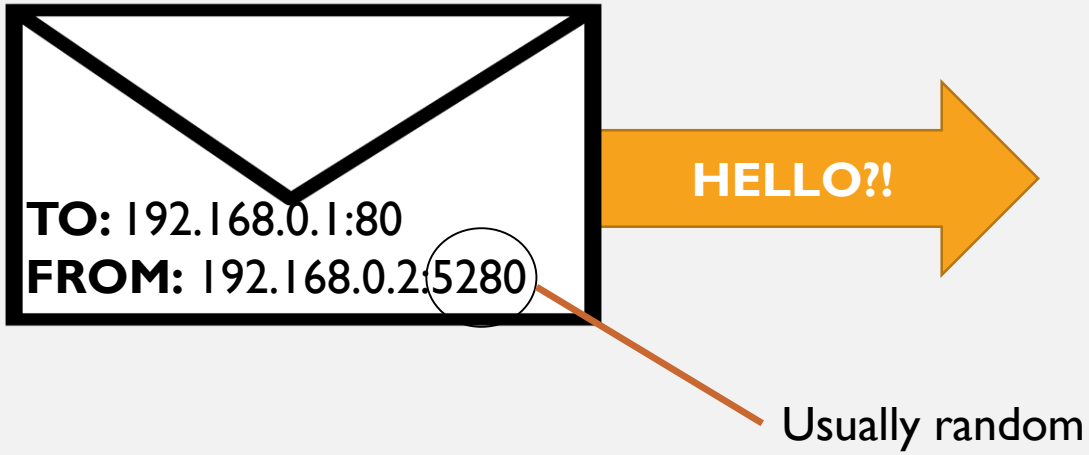
Now I have an Address/Port! Maybe I'll get Requests!

SERVER HAS AN **IP ADDRESS** AND **TCP PORT**

MEANWHILE, CLIENT ABSTACTION

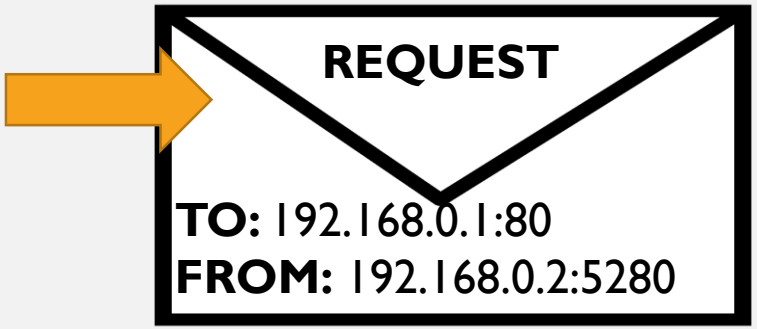
CLIENT ***CONNECTS*** TO MAKE OUTBOUND REQUESTS

TCP/IP AGAIN



CLIENT ***CONNECTS*** TO MAKE OUTBOUND REQUESTS

INCOMING REQUEST



SERVER RECEIVES REQUEST

REQUEST RESPONSE

SERVER **INVERTS TO/FROM** FOR RESPONSE

SERVER LISTENS TO MANY REQUESTS

SERVER USES **(SRC IP, SRC PORT, DST IP, DST PORT)*** TO MULTIPLEX

This is how one server on one port (e.g., webserver) handles many clients

SOCKETS

- Sockets are a simple abstraction of client-server
- Thus, there is a “client” socket and “server” socket
- The server socket *listens* for incoming connections
- The client socket makes an outbound connection to server
- The server *accepts* the incoming connection and spawns a connected socket

PYTHON SERVER SOCKET

```python
# Echo server program
import socket

HOST = ''                    # Symbolic name meaning all available interfaces
PORT = 50007                 # Arbitrary non-privileged port
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen(1)
    conn, addr = s.accept()
    with conn:
        print('Connected by', addr)
        while True:
            data = conn.recv(1024)
            if not data: break
            conn.sendall(data)
```

Basically, local IP address

PYTHON SERVER SOCKET

```python
# Echo server program
import socket

HOST = ''                    # Symbolic name meaning all available interfaces
PORT = 50007                 # Arbitrary non-privileged port
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen(1)
    conn, addr = s.accept()
    with conn:
        print('Connected by', addr)
        while True:
            data = conn.recv(1024)
            if not data: break
            conn.sendall(data)
```

IP network, Streaming is TCP

PYTHON SERVER SOCKET

```python
# Echo server program
import socket

HOST = ''                    # Symbolic name meaning all available interfaces
PORT = 50007                 # Arbitrary non-privileged port
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen(1)
    conn, addr = s.accept()
    with conn:
        print('Connected by', addr)
        while True:
            data = conn.recv(1024)
            if not data: break
            conn.sendall(data)
```

Listen with backlog of one

PYTHON SERVER SOCKET

```
# Echo server program
import socket

HOST = ''                    # Symbolic name meaning all available interfaces
PORT = 50007                 # Arbitrary non-privileged port
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen(1)
    conn, addr = s.accept()
    with conn:
        print('Connected by', addr)
        while True:
            data = conn.recv(1024)
            if not data: break
            conn.sendall(data)
```

Accept the new incoming connection

PYTHON SERVER SOCKET

```python
# Echo server program
import socket

HOST = ''                    # Symbolic name meaning all available interfaces
PORT = 50007                 # Arbitrary non-privileged port
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen(1)
    conn, addr = s.accept()
    with conn:
        print('Connected by', addr)
        while True:
            data = conn.recv(1024)
            if not data: break
            conn.sendall(data)
```

This is the spawned socket for the specific (SRC IP, SRC PORT, DST IP, DST PORT)
ALSO OF TYPE SOCKET!!!!!

PYTHON SERVER SOCKET

```python
# Echo server program
import socket

HOST = ''                    # Symbolic name meaning all available interfaces
PORT = 50007                 # Arbitrary non-privileged port
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen(1)
    conn, addr = s.accept()
    with conn:
        print('Connected by', addr)
        while True:
            data = conn.recv(1024)
            if not data: break
            conn.sendall(data)
```

With threads or non-blocking I/O, could have two+ *conn*'s at the same time

PYTHON CLIENT SOCKET

```
# Echo client program
import socket

HOST = 'daring.cwi.nl'          # The remote host
PORT = 50007                    # The same port as used by the server
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((HOST, PORT))
    s.sendall(b'Hello, world')
    data = s.recv(1024)
print('Received', repr(data))
```

Server name (could also be IP addr)

PYTHON CLIENT SOCKET

```python
# Echo client program
import socket

HOST = 'daring.cwi.nl'    # The remote host
PORT = 50007              # The same port as used by the server
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((HOST, PORT))
    s.sendall(b'Hello, world')
    data = s.recv(1024)
print('Received', repr(data))
```

Server port; client's host/port determined automatically

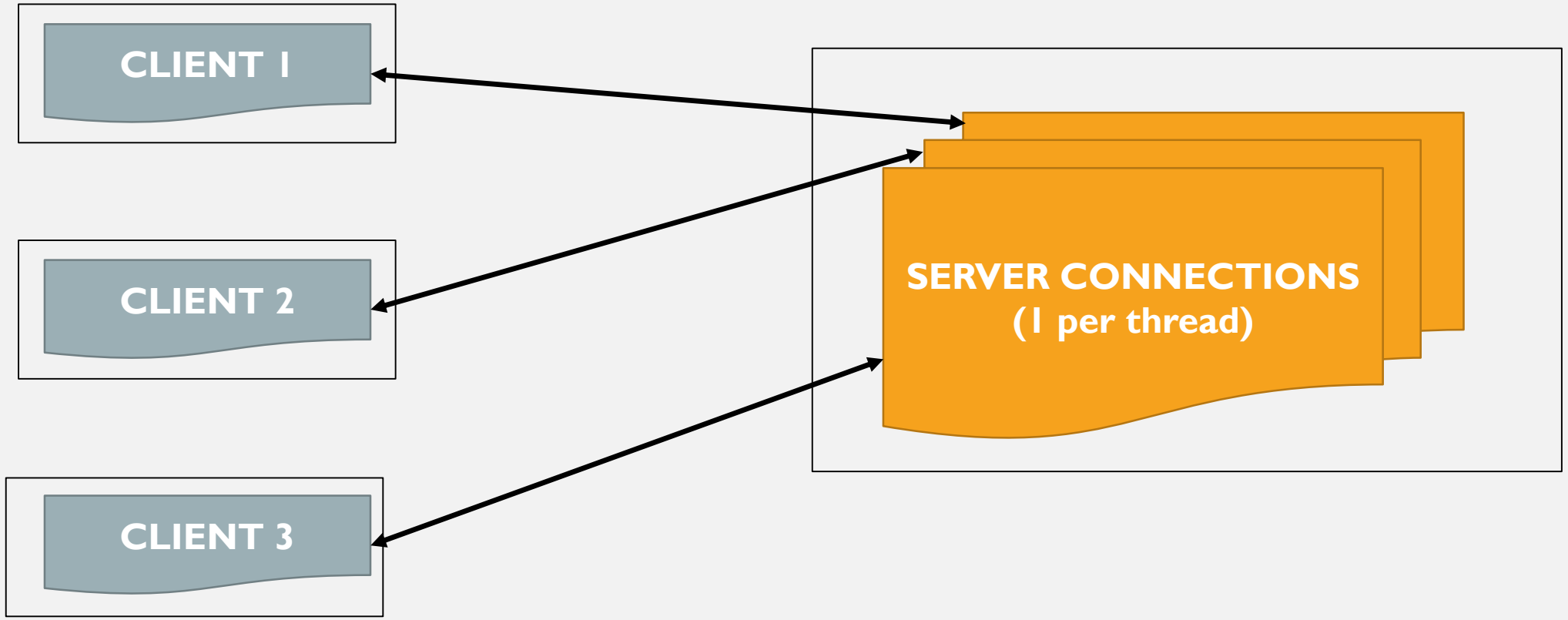
PYTHON CLIENT SOCKET

```python
# Echo client program
import socket

HOST = 'daring.cwi.nl'    # The remote host
PORT = 50007              # The same port as used by the server
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((HOST, PORT))
    s.sendall(b'Hello, world')
    data = s.recv(1024)
print('Received', repr(data))
```

Blocking call; will raise exception on failure

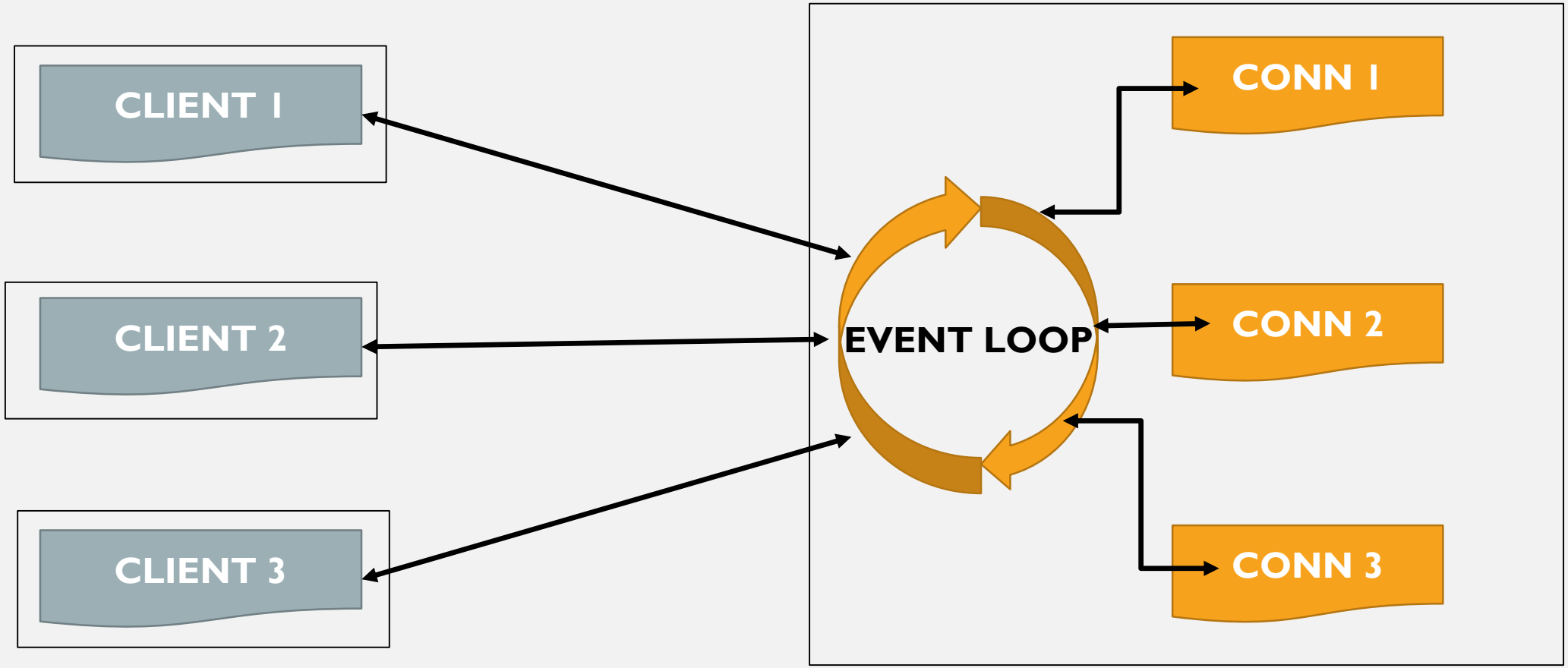
CLASSIC SOCKETS DESIGN (THREADS)



PSEUDO CODE

```
server_socket.listen(backlog)
LOOP forever;
        conn, addr = server_socket.accept()
        run_thread(server_handling_function, conn, addr)
```

CLASSIC SOCKETS DESIGN (EVENTS)



PSEUDO CODE

```
server_socket.listen(backlog)
all_sockets = [server_socket]
LOOP forever
    BLOCK until any socket in all_sockets is ready to read or write
    ready_socket = socket in all_sockets ready to read or write
    IF ready_socket == server_socket:
            conn, addr = ready_socket.accept()
            all_sockets.append(conn)
    ELSE: handle client data
```