TCP Sockets - Python

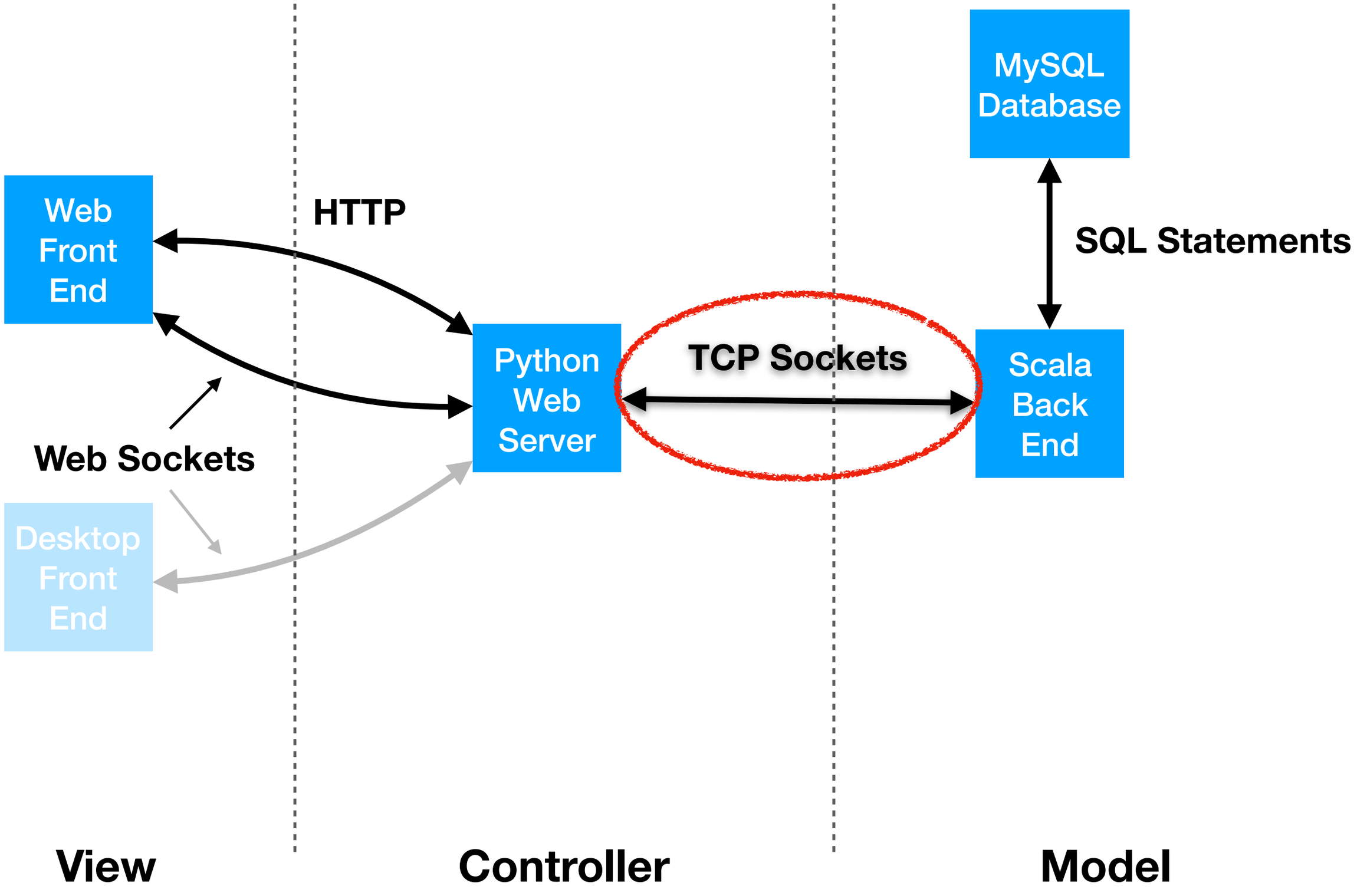
Lecture Question

Task: Write a Python program that connects to a Scala TCP socket server

- Free grader

* This question will be open until midnight

CSE116 - End Game



Socket Server

- We have a Scala TCP socket server running
- How do we connect to it in Python?

```scala
case class SendToClients(message: String)

class SocketServer extends Actor {

  import Tcp._
  import context.system

  IO(Tcp) ! Bind(self, new InetSocketAddress("localhost", 8000))

  var clients: Set[ActorRef] = Set()

  override def receive: Receive = {
    case b: Bound => println("Listening on port: " + b.localAddress.getPort)
    case c: Connected =>
      this.clients = this.clients + sender()
      sender() ! Register(self)
    case PeerClosed =>
      this.clients = this.clients - sender()
    case r: Received =>
      println("Received: " + r.data.utf8String)
    case send: SendToClients =>
      this.clients.foreach((client: ActorRef) => client ! Write(ByteString(send.message)))
  }

}
```

Python - TCP Sockets

- Assume we have a Scala TCP socket server listening on port 8000
- Connect to receive/send data
- This connection opens a *stream* of data

```
scala_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
scala_socket.connect(('localhost', 8000))

...

message = scala_socket.recv(1024).decode()

...

scala_socket.sendall(json.dumps(data).encode())
```

Python - TCP Sockets

- As data is sent to the socket it is queued in the stream
- Call recv to read any data waiting in the stream
- recv take a buffer size as parameter
  - Read at most 1024 bytes of data per call
    - Reads all data if there's < 1024 bytes waiting to be read

```
scala_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
scala_socket.connect(('localhost', 8000))

...

message = scala_socket.recv(1024).decode()

...

scala_socket.sendall(json.dumps(data).encode())
```

Python - TCP Sockets

- Send messages with sendall
- TCP Sockets do not support message types
- Can only send bytes over the stream
- Since we're communicating across programs, JSON byte strings is a good format to use

```
scala_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
scala_socket.connect(('localhost', 8000))

...

message = scala_socket.recv(1024).decode()

...

scala_socket.sendall(json.dumps(data).encode())
```

Python - TCP Sockets

- But we have a big problem
- We want to always be calling recv so we can process data as soon it's send from Scala
- How do we continuously call this while concurrently running other code?
    - Actors and Web Sockets handled this problem for us

```
scala_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
scala_socket.connect(('localhost', 8000))

...

message = scala_socket.recv(1024).decode()

...

scala_socket.sendall(json.dumps(data).encode())
```

Python - TCP Sockets

- We want to constantly call recv
- Put it in an infinite loop!
- Now we're always reading from the socket stream
- But we need to do this concurrently with the rest of our program

```
while True:
    data = the_socket.recv(1024).decode()
    # do something with data
```

Python - TCP Sockets

- One option is to listen to the socket on a separate thread
  - Concurrency managed by the OS
  - Effectively have multiple instances of the programing running
    - Threads share heap space
      - Concurrent read/writes of the same data is a concern
    - Overhead is high

```
def listen_to_scala(the_socket):
    while True:
        data = the_socket.recv(1024).decode()
        # do something with data

Thread(target=listen_to_scala, args=(scala_socket,)).start()
```

Python - TCP Sockets

- Use the eventless library to use green threads
- eventlet modifies several Python libraries including Thread
- Green threads are managed by the runtime environment (Not the OS)
    - Significantly less overhead

```
import eventlet
eventlet.monkey_patch()

def listen_to_scala(the_socket):
    while True:
        data = the_socket.recv(1024).decode()
        # do something with data

Thread(target=listen_to_scala, args=(scala_socket,)).start()
```

Python - TCP Sockets

- The TCP socket only handles streams of bytes
  - We need a way to separate the messages we receive
- Can have Scala and Python agree on a delimiter to separate them
  - Scala always sends messages followed by the delimiter
  - Python reads data until the next delimiter to process a single message

```
def listen_to_scala(the_socket):
    delimiter = "~"
    buffer = ""
    while True:
        buffer += the_socket.recv(1024).decode()
        while delimiter in buffer:
            message = buffer[:buffer.find(delimiter)]
            buffer = buffer[buffer.find(delimiter)+1:]
            # do something with message
```

Scala - TCP Sockets

- We should do this on the Scala side as well
  - Can get away with it if we don't expect concurrent messages being received
  - Not recommended (The current HW is setup without a delimiter. You may want to add one in your project)

```
var buffer = ""
val delimiter = "~"
...
case r: Received =>
 var buffer += r.data.utf8String
  while(buffer(delimiter)) {
    val curr = buffer.substring(0, buffer.indexOf(delimiter))
    buffer = buffer.substring(buffer.indexOf(delimiter)+1)
    val message: JsValue = Json.parse(curr)
    # do something with message
```

Lecture Question

Task: Write a Python program that connects to a Scala TCP socket server

- Free grader

* This question will be open until midnight