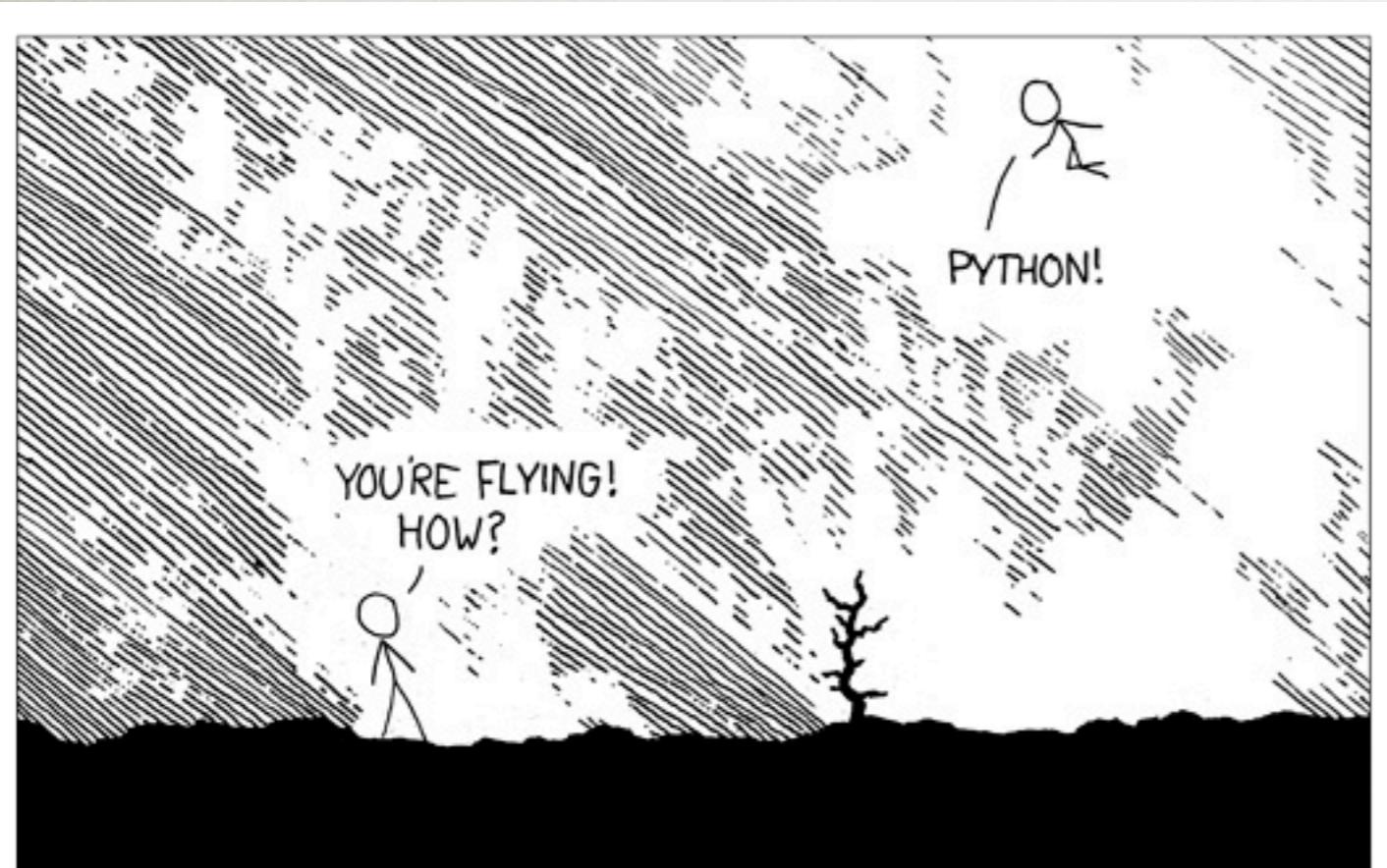


# BUILD A DYNAMIC HTTP SERVER FROM SCRATCH IN PYTHON



<http://xkcd.com/353/>

Brian Dorsey  
[brian@dorseys.org](mailto:brian@dorseys.org)  
<http://briandorsey.info>

# OVERVIEW & INTRODUCTIONS (5)

introductions

# some thoughts...

(not directly related to HTTP...)

Magic

Magic  
(or at least, indistinguishable from)

Embarrassment of riches

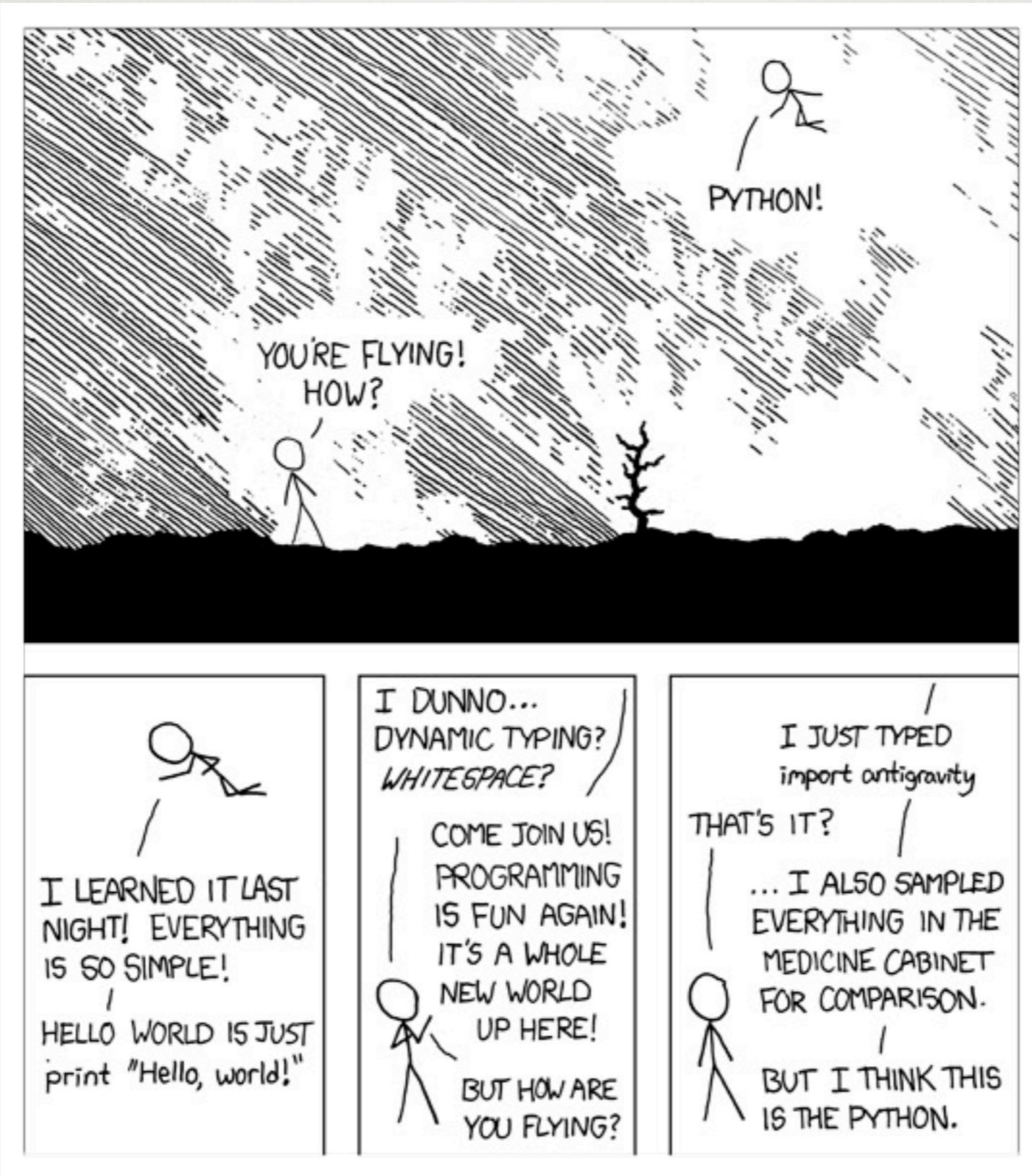
Beware the trappings  
of authority.

# Collaboration

"One of the things I find  
that's remarkable about  
Python is that it has a very  
even learning curve. Maybe  
it's not even a curve,  
It's kind of a straight line."  
- Bruce Eckel

<http://www.artima.com/intv/prodperfP.html>

(imagine graph here)



<http://xkcd.com/353/>

going to try to do the  
opposite to HTTP

skipping a lot of details  
just enough to build our  
HTTP server

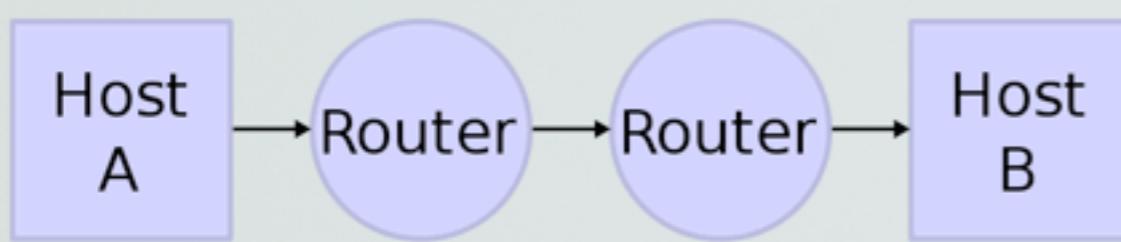
# UNIT A (25)

NETWORKING:  
FROM HARDWARE TO SOCKETS

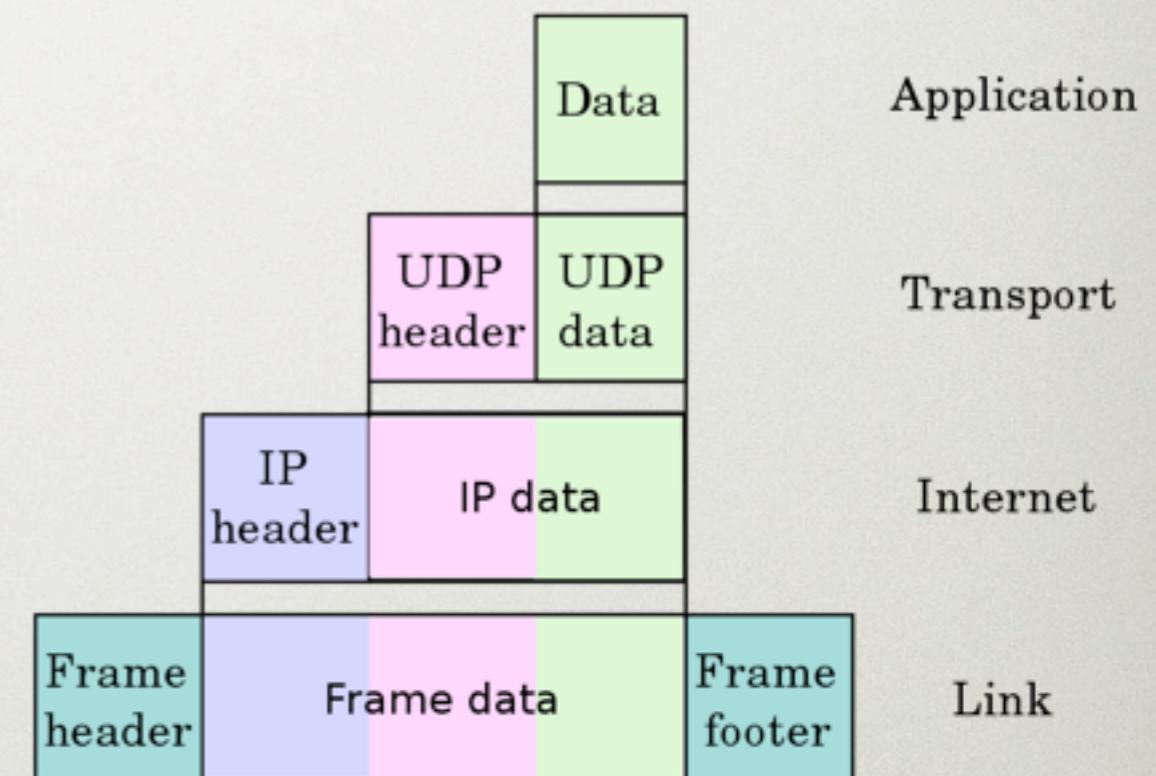
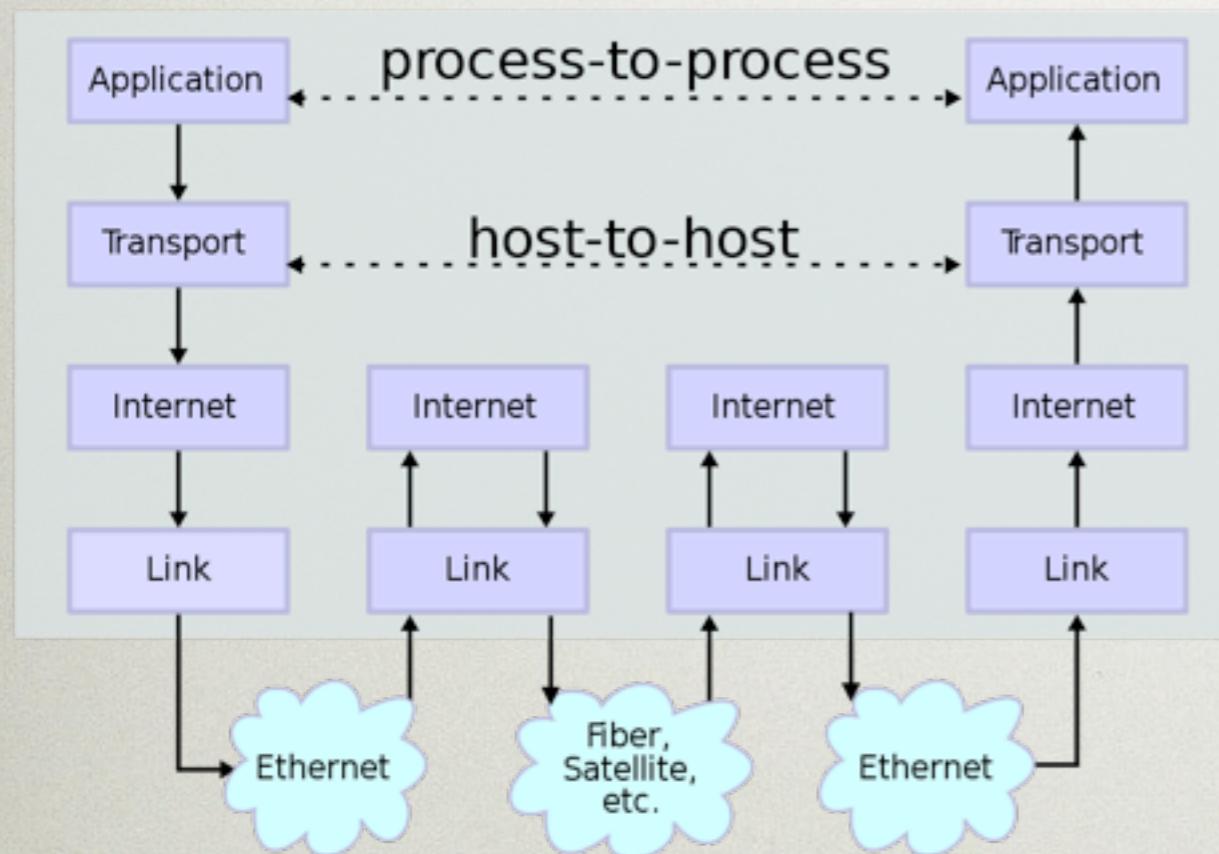
level setting

questions from the readings

# Network Topology



## Data Flow

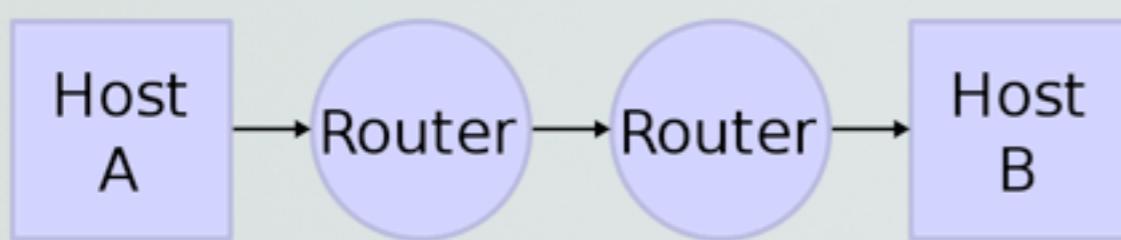


# IP addresses

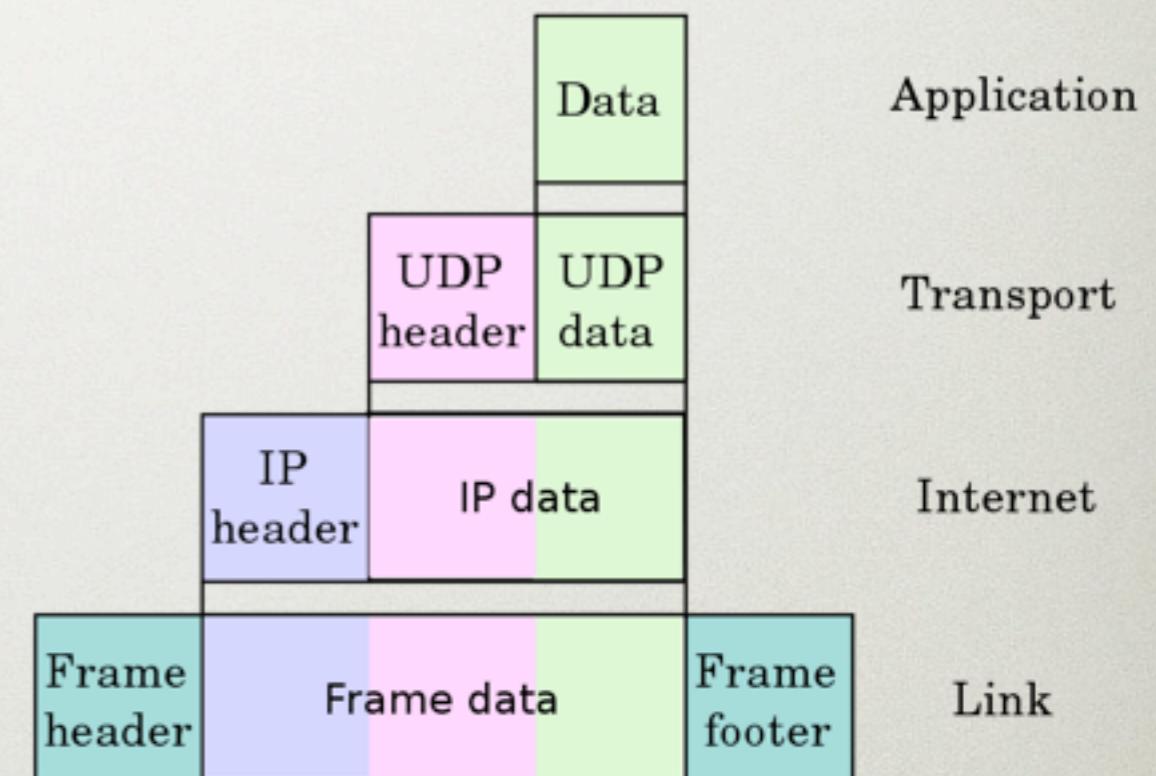
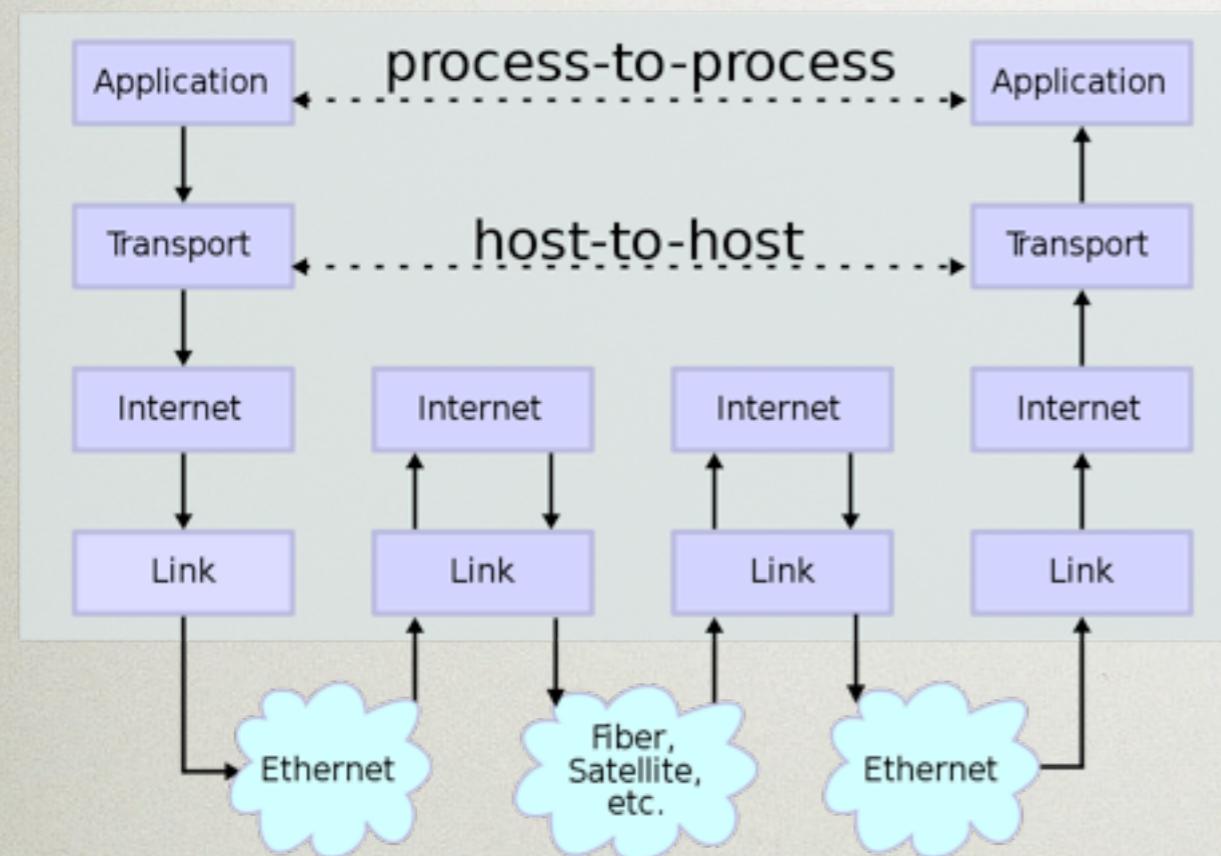
ports

transports  
TCP / UDP

# Network Topology



## Data Flow



# Socket abstraction (IP address, port, transport)

## TCP

Transmission Control Protocol

- guaranteed delivery
- connection setup (3 packets)
- error checking
- ordered delivery
- stream of data

HTTP, FTP, SMTP, telnet  
most control data

## UDP

User Datagram Protocol

- no guarantees
- no connection
- no error checking
- no order guarantees
- packets of data

DNS, DHCP, SNMP, VOIP,  
most game data  
& streaming media

text vs. binary

strings vs. bytes

unicode vs. encodings

server vs. client

# (TCP echo client)

```
import socket

host = 'localhost'
port = 50000
size = 1024

s = socket.socket(socket.AF_INET,
                  socket.SOCK_STREAM)

s.connect((host, port))
s.send('Hello, world')
data = s.recv(size)
s.close()
print 'Received:', data
```

```
import socket
```

(TCP echo server)

```
host = ''
```

```
port = 50000
```

```
backlog = 5
```

```
size = 1024
```

```
s = socket.socket(socket.AF_INET,  
                  socket.SOCK_STREAM)
```

```
s.bind((host, port))
```

```
s.listen(backlog)
```

```
while True:
```

```
    client, address = s.accept()
```

```
    data = client.recv(size)
```

```
    if data:
```

```
        client.send(data)
```

```
    client.close()
```

```
import socket

host = ''
port = 50000
backlog = 5
size = 1024
s = socket.socket(socket.AF_INET,
                  socket.SOCK_STREAM)
s.bind((host, port))
s.listen(backlog)
while True:

    client, address = s.accept()
    data = client.recv(size)
    if data:
        client.send(data)
    client.close()

    import socket
    host = 'localhost'
    port = 50000
    size = 1024
    s = socket.socket(socket.AF_INET,
                      socket.SOCK_STREAM)
    s.connect((host, port))
    s.send('Hello, world')
    data = s.recv(size)
    print 'Received:', data
    s.close()
```

**LAB A**  
**(25)**

# LAB A

**[HTTPS://GITHUB.COM/BRIANDORSEY/HTTP\\_TUTORIAL](https://github.com/briandorsey/http_tutorial)**

---

- run echo\_server.py
- run echo\_client.py a few times
- telnet localhost 50000
- open two python interpreters and manually input both programs, starting with the server side until it blocks, then just enough of the client to unblock the server.
- modify echo\_server.py to ignore what the client sends, and always reply with the time
- Bonus: raw socket version of whatismyip.com - tell the client what its' IP address is.

how'd that go?

# interactive

```
conn, addr = s.accept()
print 'Connected by', addr
data = 'Enter a Number:'
conn.send(data)

number1 = conn.recv(1024)
data = 'Enter another Number:'
conn.send(data)

number2 = conn.recv(1024)
total = int(number1) + int(number2)
data = "Total = " + str(total)
conn.send(data)
conn.close()
```

```
s.connect((HOST, PORT))
```

```
data = s.recv(1024)
print repr(data)
number = raw_input()
s.send(number)
```

```
data = s.recv(1024)
print repr(data)
number = raw_input()
s.send(number)
```

```
data = s.recv(1024)
print repr(data)
s.close()
```

# UNIT B

## (25)

HTTP

# HTTP Made Really Easy

did it work?

**GET /path/to/index.html HTTP/1.0**

## HTTP Requests

## HTTP Responses

stateless

**GET /path/to/index.html HTTP/1.0**

resources  
(nouns)

required first line  
optional headers  
“blank” line (CRLF)  
body (data)

GET /path/file.html HTTP/1.0  
User-Agent: HTTPTool/1.0  
[blank line here]

request

---

HTTP/1.0 200 OK

Date: Fri, 31 Dec 1999 23:59:59 GMT

Content-Type: text/html

Content-Length: 1354

response

<html>  
<body>  
<h1>Happy New Millennium!</h1>  
(more file contents)

• • •

</body>  
</html>

**GET /path/to/index.html HTTP/1.0**

# HTTP Requests

**GET** /path/to/index.html HTTP/1.0

methods  
(verbs)

**GET** /path/to/index.html HTTP/1.0

GET

POST

PUT

DELETE

**GET** /path/to/index.html HTTP/1.0

POST = Create

GET = Read

PUT = Update

DELETE = Delete

**GET** /path/to/index.html HTTP/1.0

safe

GET

side  
effects

POST

PUT

DELETE

**GET** /path/to/index.html HTTP/1.0

idempotent

GET

PUT

DELETE

changes  
after each  
request

POST

**GET /path/to/index.html HTTP/1.0**

**resources != files**

**HTTP/1.0 200 OK**

# HTTP Responses

**HTTP/1.0 200 OK**

version

status code

reason phrase

## HTTP/1.0 200 OK

- 1xx indicates an informational message
- 2xx indicates success of some kind
- 3xx redirects the client to another URL
- 4xx indicates an error on the client's part
- 5xx indicates an error on the server's part

## HTTP/1.0 **200** OK

- 200 OK
- 404 Not Found
- 301 Moved Permanently
- 302 Moved Temporarily
- 303 See Other (HTTP 1.1 only)
- 500 Server Error

GET /path/file.html HTTP/1.0  
User-Agent: HTTPTool/1.0  
[blank line here]

request

---

HTTP/1.0 200 OK

Date: Fri, 31 Dec 1999 23:59:59 GMT

Content-Type: text/html

Content-Length: 1354

response

<html>  
<body>  
<h1>Happy New Millennium!</h1>  
(more file contents)

• • •

</body>  
</html>

Header-Name: value

Quick reference to  
HTTP headers

[http://www.cs.tut.fi/  
~jkorpela/http.html](http://www.cs.tut.fi/~jkorpela/http.html)

body data

Content-Type: xyz

(mime types)

# DEBUGGING TOOLS

---

- windows:

<http://www.fiddler2.com/fiddler2/>

- windows & mac:

<http://www.charlesproxy.com/>

- firefox:

<http://getfirebug.com/>

- Safari, Chrome and IE: built in

questions?

# HTTP telnet demo

# HTTP curl demo

LAB B  
(20)

# LAB B

---

- experiment with making manual requests using telnet:  
**GET / HTTP/1.0\r\n\r\n**
- look at some headers with your browser's developer tools
- turn the echo client into an HTTP client
- bonus: accept a URL argument

# UNIT C (25)

30 MINUTE WEB SERVER

# The Thirty Minute Web Server

(presented in about  
thirty minutes)

demo

story

review

triple quotes  
string interpolation

codewalk

code

git clone git://github.com/briandorsey/http\_tutorial.git

or

[https://github.com/briandorsey/http\\_tutorial](https://github.com/briandorsey/http_tutorial)

[https://github.com/briandorsey/http\\_tutorial/downloads](https://github.com/briandorsey/http_tutorial/downloads)

# LAB C

## (25)

safe place to fail

# LAB C

---

git clone git://github.com/briandorsey/http\_tutorial.git

- run `thirty_minute_webserver.py`
- point your web browser at it, experiment. Break code, add prints, etc
- add `/time` url which returns HTML with the actual current time

# review

```
python -m SimpleHTTPServer 8000
```

# httplib

```
import httplib
```

```
conn = httplib.HTTPConnection("www.python.org")  
conn.request("GET", "/index.html")
```

```
r1 = conn.getresponse()  
print r1.status, r1.reason
```

```
data1 = r1.read()
```

```
conn.close()
```

# urllib2

```
import urllib2
```

```
f = urllib2.urlopen('http://www.python.org/')
print f.read()
```

# requests

```
import requests
```

```
r = requests.get('http://www.python.org/')
print r.text
```

# WRAPUP

# ASSIGNMENT

---

- Write me a note with feedback on this tutorial. brian@dorseys.org
- Did you have an HTTP epiphany?  
Is HTTP mysterious?
- What worked well, what worked poorly.
- Additional info which should be included?
- Comments

# ASSIGNMENT?

---

- Update `thirty_minute_webserver.py`
- Instead of showing the content of .py files, run the script and return its' output instead.  
(the stdlib subprocess.Popen() will be useful)
- test with script which outputs the time  
(`print_time.py`)
- update the `print_time.py` script to output html
- You've now written a dynamic webserver!  
(and re-invented CGI! )

# BONUS (25)

MORE PROTOCOLS

EHR

BREAKING! WAVES

SEA OF PROTOCOL  
CONFUSION

GOOGLE TALK  
INVASION  
FLEET

USENET  
(STILL HERE!)

\*E2

ICQ

S: 220 foo.com Simple Mail Transfer Service Ready  
C: EHLO bar.com  
S: 250-foo.com greets bar.com  
S: 250-8BITMIME  
S: 250-SIZE  
S: 250-DSN  
S: 250 HELP  
C: MAIL FROM:<Smith@bar.com>  
S: 250 OK  
C: RCPT TO:<Jones@foo.com>  
S: 250 OK  
C: RCPT TO:<Green@foo.com>  
S: 550 No such user here  
C: RCPT TO:<Brown@foo.com>  
S: 250 OK  
C: DATA  
S: 354 Start mail input; end with <CRLF>.<CRLF>  
C: Blah blah blah...  
C: ...etc. etc. etc.  
C: .  
S: 250 OK  
C: QUIT  
S: 221 foo.com Service closing transmission channel

SMTP

# POP3

```
C: <client connects to service port 110>
S: +OK POP3 server ready <1896.6971@mailgate.dobbs.org>
C: USER bob
S: +OK bob
C: PASS redqueen
S: +OK bob's maildrop has 2 messages (320 octets)
C: STAT
S: +OK 2 320
C: LIST
S: +OK 2 messages (320 octets)
S: 1 120
S: 2 200
S: .
C: RETR 1
S: +OK 120 octets
S: <the POP3 server sends the text of message 1>
S: .
C: DELE 1
S: +OK message 1 deleted
C: RETR 2
S: +OK 200 octets
S: <the POP3 server sends the text of message 2>
S: .
C: DELE 2
S: +OK message 2 deleted
C: QUIT
S: +OK dewey POP3 server signing off (maildrop empty)
C: <client hangs up>
```

# IMAP

```
C: <client connects to service port 143>
S: * OK example.com IMAP4rev1 v12.264 server ready
C: A0001 USER "frobozz" "xyzzy"
S: * OK User frobozz authenticated
C: A0002 SELECT INBOX
S: * 1 EXISTS
S: * 1 RECENT
S: * FLAGS (\Answered \Flagged \Deleted \Draft \Seen)
S: * OK [UNSEEN 1] first unseen message in /var/spool/mail/esr
S: A0002 OK [READ-WRITE] SELECT completed
C: A0003 FETCH 1 RFC822.SIZE           Get message sizes
S: * 1 FETCH (RFC822.SIZE 2545)
S: A0003 OK FETCH completed
C: A0004 FETCH 1 BODY[HEADER]          Get first message header
S: * 1 FETCH (RFC822.HEADER {1425}
<server sends 1425 octets of message payload>
S: )
S: A0004 OK FETCH completed
C: A0005 FETCH 1 BODY[TEXT]            Get first message body
S: * 1 FETCH (BODY[TEXT] {1120}
<server sends 1120 octets of message payload>
S: )
S: * 1 FETCH (FLAGS (\Recent \Seen))
S: A0005 OK FETCH completed
C: A0006 LOGOUT
S: * BYE example.com IMAP4rev1 server terminating connection
S: A0006 OK LOGOUT completed
C: <client hangs up>
```

# IMAP (reordering)

```
C: <client connects to service port 143>
S: * OK example.com IMAP4rev1 v12.264 server ready
C: A0001 USER "frobozz" "xyzzy"
S: * OK User frobozz authenticated
C: A0002 SELECT INBOX
S: * 1 EXISTS
S: * 1 RECENT
S: * FLAGS (\Answered \Flagged \Deleted \Draft \Seen)
S: * OK [UNSEEN 1] first unseen message in /var/spool/mail/esr
S: A0002 OK [READ-WRITE] SELECT completed
C: A0003 FETCH 1 RFC822.SIZE
C: A0004 FETCH 1 BODY[HEADER]
C: A0005 FETCH 1 BODY[TEXT]
S: * 1 FETCH (RFC822.SIZE 2545)
S: A0003 OK FETCH completed
S: * 1 FETCH (RFC822.HEADER {1425})
<server sends 1425 octets of message payload>
S: )
S: A0004 OK FETCH completed
S: * 1 FETCH (BODY[TEXT] {1120})
<server sends 1120 octets of message payload>
S: )
S: * 1 FETCH (FLAGS (\Recent \Seen))
S: A0005 OK FETCH completed
C: A0006 LOGOUT
S: * BYE example.com IMAP4rev1 server terminating connection
S: A0006 OK LOGOUT completed
C: <client hangs up>
```

```
1 boardsize 7  
=1  
  
2 clear_board  
=2  
  
3 play black D5  
=3  
  
4 genmove white  
=4 C3  
  
5 play black C3  
?5 illegal move  
  
6 play black E3  
=6  
  
7 showboard  
=7  
A B C D E F G  
7 . . . . . . . 7  
6 . . . . . . . 6  
5 . . + X + . . 5  
4 . . . + . . . 4  
3 . . 0 . X . . 3  
2 . . . . . . . 2      WHITE (0) has captured 0 stones  
1 . . . . . . . 1      BLACK (X) has captured 0 stones  
A B C D E F G  
  
8 quit  
=8
```

# GNU Go Text Protocol (GTP)

**GET /index.html HTTP/1.1**  
**Host: www.example.com**

**HTTP**

**HTTP/1.1 200 OK**

**Date: Mon, 23 May 2005 22:38:34 GMT**

**Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)**

**Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT**

**Etag: "3f80f-1b6-3e1cb03b"**

**Accept-Ranges: bytes**

**Content-Length: 438**

**Connection: close**

**Content-Type: text/html; charset=UTF-8**

**<data>**

many others only run  
or optionally run  
over HTTP

now, in Python

```
import smtplib  
  
template = """From: %s  
To: %s  
Subject: %s  
  
"""  
  
from_addr = 'Darth Vader <darth@deathstar.com>'  
to_addrs = 'briandorsey@gmail.com'  
subject = "I'm your father."  
message = 'message body'  
headers = template % (from_addr, to_addrs, subject)  
  
s = smtplib.SMTP('mail.blueboxgrid.com')  
s.ehlo()  
s.sendmail(from_addr, to_addrs, headers + message)  
s.close()
```

```
import poplib  
import string, random
```

poplib

```
SERVER = "pop.spam.egg"
```

```
USER = "mulder"
```

```
PASSWORD = "trustno1"
```

```
# connect to server
```

```
server = poplib.POP3(SERVER)
```

```
# login
```

```
server.user(USER)
```

```
server.pass_(PASSWORD)
```

```
# list items on server
```

```
resp, items, octets = server.list()
```

```
# download a random message
```

```
id, size = string.split(random.choice(items))
```

```
resp, text, octets = server.retr(id)
```

# imaplib

```
import imaplib
import string, random

SERVER = "imap.spam.egg"
USER   = "mulder"
PASSWORD = "trustno1"

# connect to server
server = imaplib.IMAP4(SERVER)

# login
server.login(USER, PASSWORD)
server.select()

# list items on server
resp, items = server.search(None, "ALL")
items = string.split(items[0])

# fetch a random item
id = random.choice(items)
resp, data = server.fetch(id, "(RFC822)")
text = data[0][1]
```

<http://effbot.org/librarybook/imaplib.htm>

# httplib

```
import httplib
```

```
conn = httplib.HTTPConnection("www.python.org")  
conn.request("GET", "/index.html")
```

```
r1 = conn.getresponse()  
print r1.status, r1.reason
```

```
data1 = r1.read()
```

```
conn.close()
```

# urllib2

```
import urllib2
```

```
f = urllib2.urlopen('http://www.python.org/')
print f.read()
```

what's included in the  
standard library?

non-stdlib libraries?

a library exists for  
pretty much every protocol

for SSH / SFTP

Paramikio

<http://www.lag.net/paramiko/>

```
url = 'http://www.python.org/  
webbrowser.open_new_tab(url)
```