Bread and butter - you will use it for… almost all workshops after this. You may not understand everything after this lecture, or even after the workshop. And it’s ok. We’re not going to learn everything about express in this lecture.

You will learn it well, you’ll have to!

COMING UP

- **Part I: The Internet**
  - Clients & Servers
  - HTTP
  - TCP
  - IP
- **Part II: Node Servers**
- **Part III: Express**

but first, we're going to very briefly talk about the internet…

PART I: THE INTERNET

CLIENTS & SERVERS

- Client requests a resource
- Server responds with resource
- Client initiates, server responds; not other direction
- These are *roles* — not technical specs or computer types

what do the words mean?
roles a program can play.
this is a paradigm - a specific kind of relationship. There are others - sockets, peer to peer, etc.

CLIENTS & SERVERS

A bank teller is a server. Clients ask the teller for stuff and the teller responds. The teller doesn't go looking out on the street for clients, he/she just waits for requests.

CLIENTS & SERVERS

DEAR ABBY:

My Dad Objects To a Pet Monkey

DEAR ABBY: I am 13 years old and my Daddy said that when I earned enough money I could buy anything I wanted with it.

All my life, I have wanted a monkey. I have never had a pet monkey. I could buy a pet monkey and he said no, because I wouldn't know how to take care of it. My Mom is the lazy type. You know everything has to be just so. Do you know anyone who has a pet monkey, and can give me some advice? MAN'S A MONKEY

DEAR WANTS: I have had two pet monkeys (David and Dolly-Molly) and, although I love monkeys, your father is right. To quote my sister-in-law, "Monkeys should live with monkeys, and people should live with people."

ABIGAIL VAN BUREN

Dear Abby: clients write in for advice, Abby responds with advice. Abby doesn't send prospective letters to newspaper readers…

a real dear abby…

CLIENTS & SERVERS

When you hear "server" you probably think of a big special computer connected to the internet that hosts websites…

is a program running on any computer that listens for requests, and then sends a response

CLIENTS & SERVERS

…however, a *web server* is really any PROGRAM connected to the internet, which can receive requests and send responses. That means your laptop can be a server. Heck, it can be many servers!

CLIENTS & SERVERS

Heck, it can be many servers (if each is on a different port) — and also a client (for other servers).

a given process can even switch! a server that listens, then sends a request and is the client. they are relative terms

<RUNNING SERVER DEMO>

No coding demo yet — just demo RUNNING a tiny web server. E.g. tell audience to use curl to hit your IP:Port/greeting and see what they get back.

curl
curl -i
curl -v
curl --trace

Tiny HTTP server demo (scale it back to simplest case, then built up):

```
'use strict';

const http = require('http');
const EOL = require('os').EOL;

const requestResponseHandler = function (req, res) {

  // server window, does not go to client
  console.log('Received request!', req.method, req.url);

  // starting to respond to client
  res.writeHead(200, { 'Content-Type': 'text/plain'});
  res.write('Here is part of a…' + EOL);

  setTimeout(function(){

    // completing the response to the client, depending on what they asked for.
```

WEB SERVERS

- **Processes (running programs), not physical machines**
  - Might be running on a laptop,
  - or a Raspberry Pi,
  - or an enterprise-grade workstation…
- **Listening on a *port* for incoming requests**
- **Send back responses**
- **…but we are getting ahead of ourselves.**

INTERNET COMMUNICATION PROTOCOLS

TCP
BitTorrent
AFP
HTTPS
POP
9P
IMAP
BGP
SSL
UDP
HTTP
SCP
ICMP
FTP
SSH
IP
PPP
NFS
SMTP

PROTOCOL

- Rules for interaction / communication
- Specification, not implementation

not the message itself, and not the transmission

concept, not implementation

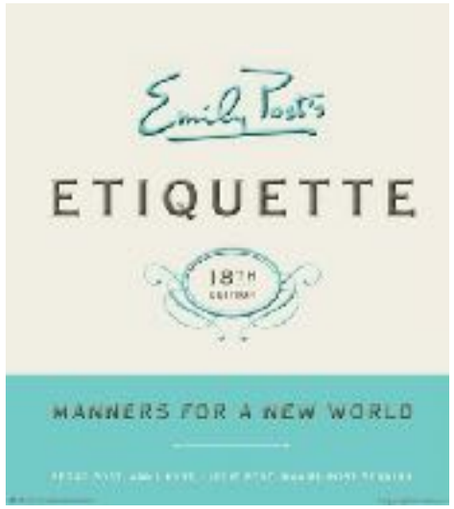
PROTOCOL

john adams meeting a king george,
must walk, bow, two paces, bow, walk forward, bow, and wait for king to speak

PROTOCOL

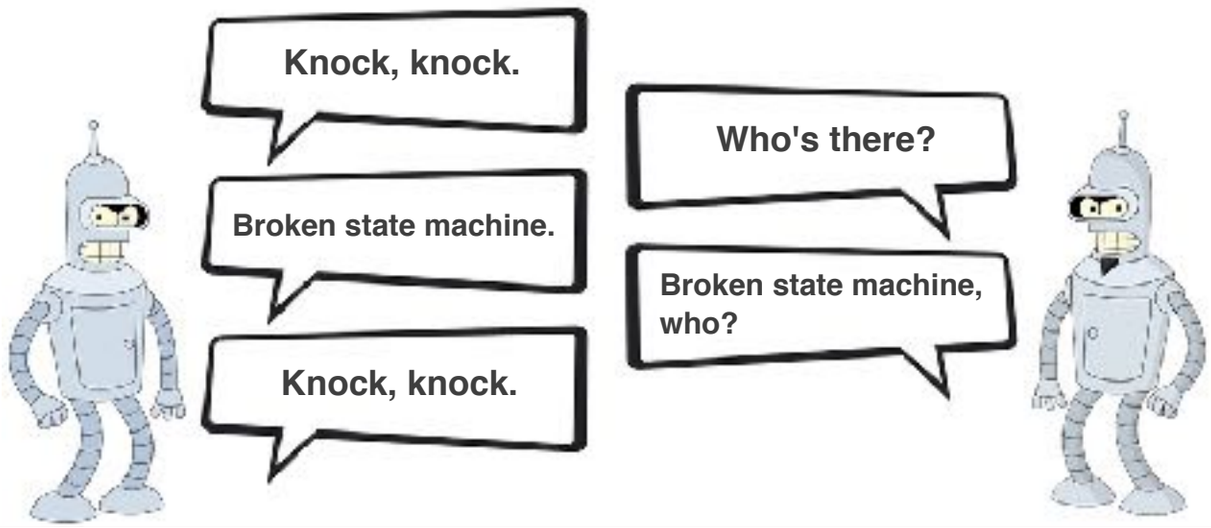
he’s a protocol droid!
facilitates communication between various parties.

PROTOCOL



proper formatting of messages - no mail system, or words on the paper. Just what info should be there, what are the rules for you and for them to respond.

PROTOCOL


Knock, knock.
Broken state machine.
Knock, knock.
Who's there?
Broken state machine, who?

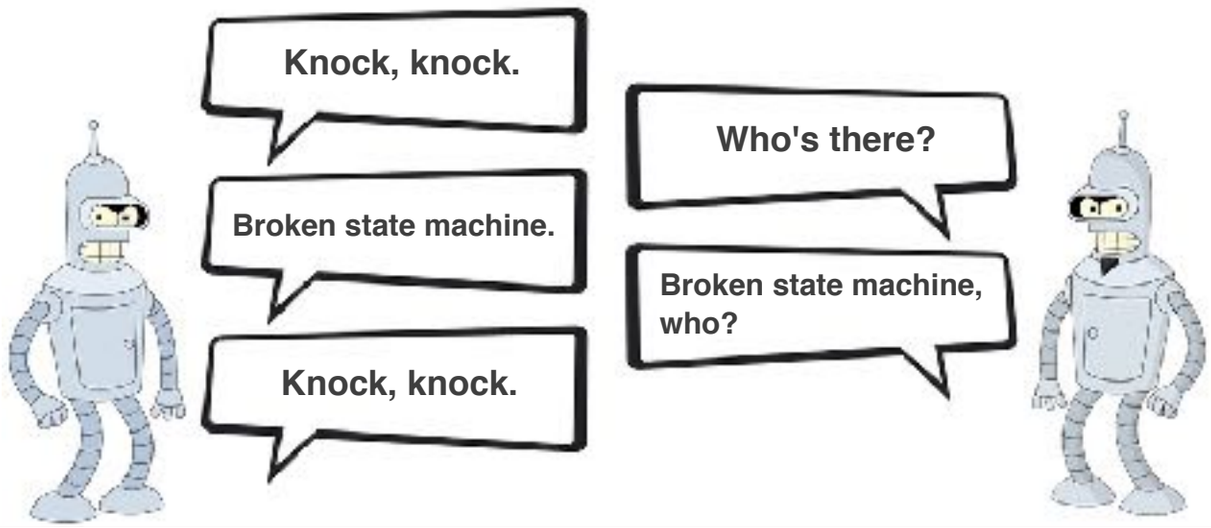
THE KNOCK-KNOCK MESSAGE PROTOCOL

- Joker opens connection with "knock, knock."
- Victim completes handshake with "who's there?"
- Joker transmits identity label: "<IDENTITY>"
- Victim requests clarification: "<IDENTITY> who?"
- Joker delivers payload: "<PUNCHLINE>"
- Joke is now delivered, close connection. Participants may optionally laugh and/or dodge fists.

MESSAGING / APP VS. TRANSMISSION

- KnockKnock is an *application level* protocol
- It specifies the sequence and content of messages
- It does NOT specify how those messages are transmitted

KNOCK KNOCK OVER VOX



KNOCK KNOCK OVER TEXT

KNOCK KNOCK OVER BLACKBOARD

Knock, knock...

who's there?

HTTP

Hypertext Transfer Protocol

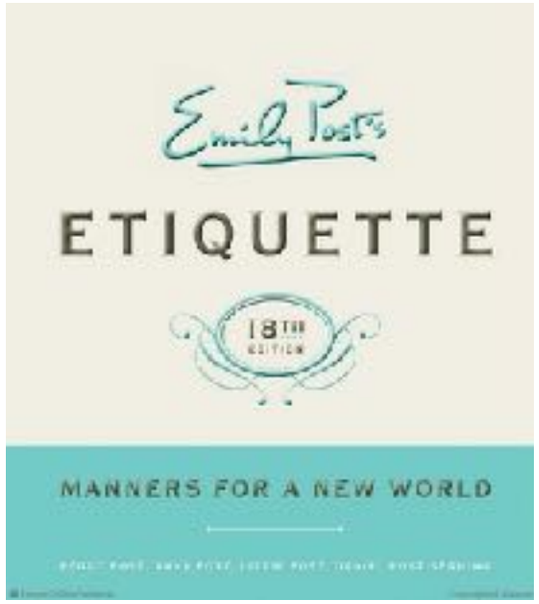
Hypertext Transfer Protocol

HTTP

- An *application-level* communications protocol. You might call it a *messaging* protocol.
- Specifies allowable *metadata* and *content* of messages.
- Does *NOT* specify *how* messages are transmitted!
- STATELESS: does *not* need to remember previous req-res!

like knock knock - application level protocol - formatting and content, not implementation
req, res cycle. then it's done. no cycles know of any other cycles. Server doesn't need to remember anything. Don't depend upon eachother.

each request can be individually responded to just with the info in the req.



HTTP is like the rules for proper wedding invitations / business letters / letters of condolence / etc. — "you must include the date in such-and-such format in such-and-such location," etc.

HTTP PROTOCOL

- RFC (Request For Comments) 7230 (link)
- By the IETF (Internet Engineering Task Force)
- But a *generic* messaging protocol
  - *"HTTP is a generic interface protocol for information systems. It is designed to hide the details of how a service is implemented... independent of the types of resources provided."*

first of 7 documents that document the http protocol…
very strict and long, and detailed
So… we don't write it ourselves.
so we use a library - built into node - http. Not express!

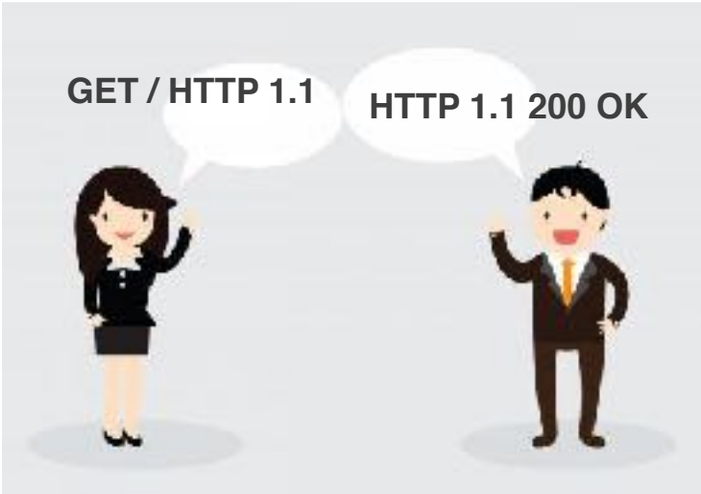
HTTP CLIENTS & SERVERS

- **Example Clients**
  - web browsers
  - household appliances
  - stereos
  - firmware update scripts
  - command-line programs
  - mobile apps
  - communication devices
- **Example Servers**
  - web servers
  - home automation units
  - networking components
  - office machines
  - autonomous robots
  - news feeds
  - traffic cameras

again, it’s not specific to the internet, even though it was designed with the internet in mind.

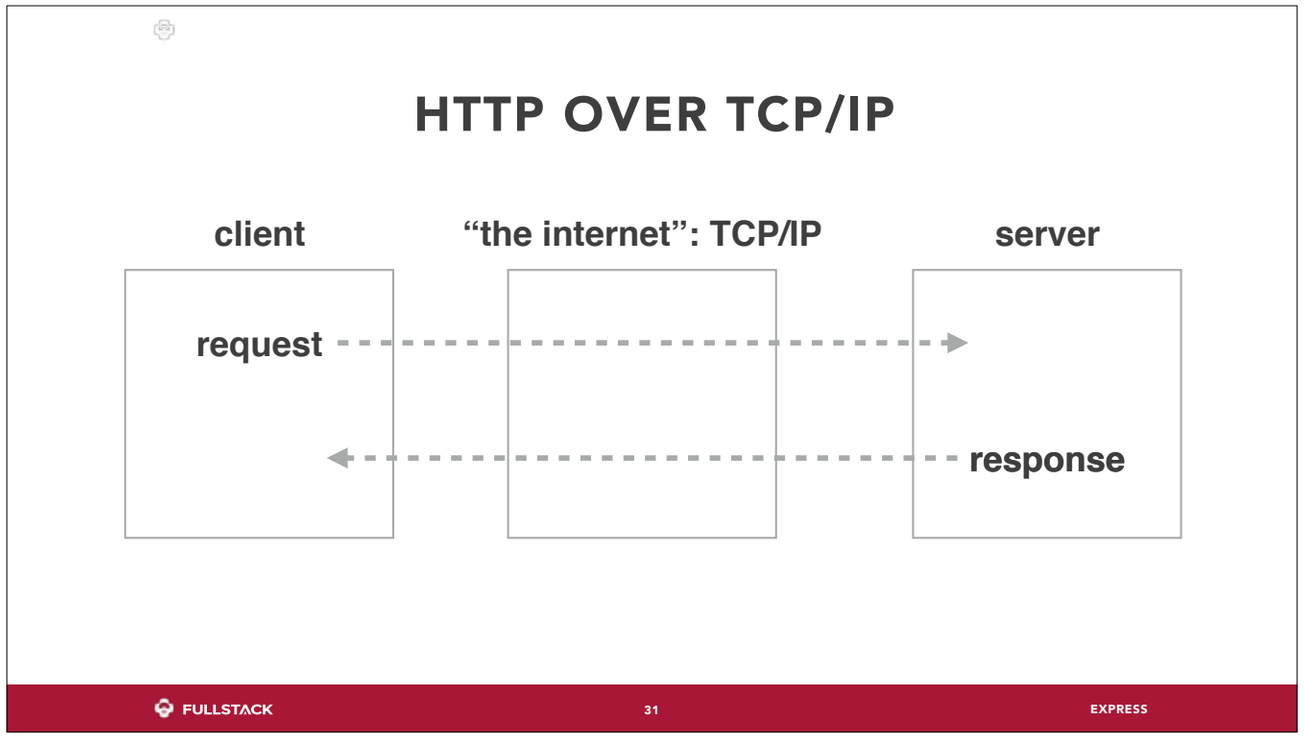
NOT A TRANSMISSION PROTOCOL!

HTTP OVER VOX



does not care that its over the internet.
aside from line formatting and such… this would be valid

HTTP OVER TEXT



we are learning web development
transmission Control protocol over internet protocol - several layers of other protocols that we’re not going to talk about right now - but I’ll send out a thing for you to read.

TCP/IP is a little bit like the mail system. Addressing, sorting, routing packets, etc. Not a perfect metaphor — the mail system doesn't split your letter into a thousand packets and then glue them back together on the other end, for example.

HTTP

Every request gets exactly one (total) response
(sometimes a response is broken up into chunks)

HTTP REQUEST

just a message with a certain format

method URI

headers

```
POST /docs/1/related HTTP/1.1
Host: www.test101.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)

bookId=12345&author=Nimit
```

body

(from http://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html)

uri === name
method === verb

you come up with labels, people ask, you decide how to respond. to http and to your server this is just a sequence of characters. You have to program the logic. you come up with a plan.
But… you shouldn't come up with the plan
there is a standard pattern people use: REST Representational state transfer. Basically, it alternates between noun, id, noun, id. of nouns collection, get me id 1, then nouns of this type that are a child of that resource.

Everything we've done up till now has been GET.

COMMON VERBS

GET	“read”
POST	“create”
PUT	“update”
DELETE	“delete”

url bar - Get request
80 is default port

get docs/
get docs/1
etc
http does not guarantee this. it’s up to you to program your server to go collect the docs and send them.

HTTP RESPONSE

status

headers

```
HTTP/1.1 200 OK
Date: Sun, 18 Oct 2009 08:56:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Sat, 20 Nov 2004 07:16:26 GMT
ETag: "10000000565a5-2c-3e94b66c2e680"
Accept-Ranges: bytes
Content-Length: 44
Connection: close
Content-Type: text/html
X-Pad: avoid browser bug

<html><body><h1>It works!</h1></body></html>
```

(from http://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html)

payload/body

incoming sequences of characters - linear pipe

COMMON STATUSES

200	“OK”
201	“created”
304	“cached”
400	“bad request”
401	“unauthorized”
404	“not found”
500	“server error”

Any attempt to brew coffee with a teapot should result in the error code "418 I'm a teapot". The resulting entity body MAY be short and stout.

300 - redirection
400 - client error
500 - server error

PROTOCOL

- Rules
- Specification, NOT implementation
- Often used for communication

node's `http` library is an implementation of HTTP

EXPRESS

A node library for request handling

Before we even talk about express though, let's code out a node server, and handle the requests manually

<CODE>

</CODE>

EXPRESS

- Treats requests as objects, created by event
- Matches on verb AND route
- Allows chaining of many handlers
- Enables modular layering with “routers”

POP QUIZ

CLIENT

Something that makes (HTTP) requests

SERVER

Something that responds to (HTTP) requests

REQUEST

A formatted message sent over the network by a client.
Contains VERB, URI (route), headers, and body.

RESPONSE

A server's reply to a request (formatted message).
Contains headers, payload, and status.

REQUEST-RESPONSE CYCLE

*The client **always** initiates by sending a request, and the server completes it by sending **exactly one** response*

EXPRESS MIDDLEWARE

A function that handles an incoming request, either by A) producing a side effect (e.g. logging), B) modifying the existing request or potential response (e.g. body parsing), and/or C) completing the response (e.g. an HTTP endpoint). Has the form: **`function(req, res, next){…}`**

REQUEST QUERY STRING

A way to pass data from client to server.

verb route

headers

```
POST /docs/index?x=123&foo=that HTTP/1.1
Host: www.test101.com
Accept: image/gif, image/jpeg, */*
Accept-L
Accept-E
User-Age                                                    NT 5.1)

bookId=12345&author=Nimit
```

In express…

```
request.query = {x:123, foo: 'that'}
```

body

(from http://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html)

REQUEST BODY

A way to pass data from client to server.

verb route

headers

```
POST /docs/index?x=123&foo=that HTTP/1.1
Host: www.test101.com
Accept: image/gif, image/jpeg, */*
Ac
Ac
U

bookId=12345&author=Nimit
```

In express…

```
request.body = {bookId:12345, author: 'Nimit'}
```

body

(from http://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html)

REQUEST PARAMS

A variable portion of the URI.

ROUTER

A "layer" of route handlers (middlewares).

GOTCHAS

- `app.get` v `app.get`
- routes are not file paths
- order matters
- `req.params` v `req.query` v `req.body`
- `app.use` v `app.all`