How to Address Processes

- to receive messages, process must have identifier
- host device has unique 32-bit IP address
- Q: does IP address of host on which process runs suffice for identifying the process?
- A: no, many processes can be running on same host

- identifier includes both IP address and port numbers associated with process on host.
- example port numbers:
  - HTTP server: 80
  - mail server: 25
- to send HTTP message to ee.hawaii.edu web server:
  - IP address: 128.171.61.135
  - port number: 80

Application Layer Protocols Define:

- types of messages exchanged,
  - e.g., request, response
- message syntax:
  - what fields in messages & how fields are delineated
- message semantics
  - meaning of information in fields
- rules for when and how processes send & respond to messages

- open protocols:
  - defined in RFCs, everyone has access to protocol definition
  - allows for interoperability
    - e.g., HTTP, SMTP
- proprietary protocols:
  - e.g., Skype, Zoom

What do Applications Need from the Transport Layer Below?

- data integrity
  - some apps (e.g., file transfer, web transactions) require 100% reliable data transfer
  - other apps (e.g., audio) can tolerate some loss
- timing
  - some apps (e.g., Internet telephony, interactive games) require low delay to be “effective”
- throughput
  - some apps (e.g., multimedia) require minimum amount of throughput to be “effective”
  - other apps (“elastic apps”) make use of whatever throughput they get
- Security
  - encryption, data integrity, …

Transport service requirements: common apps

application	data loss	throughput	time sensitive?
file transfer/download			
e-mail			
Web documents			
real-time audio/video			
streaming audio/video			
interactive games			
text messaging			

Internet transport protocols services

TCP service:

- reliable transport between sending and receiving process
- flow control: sender won't overwhelm receiver
- congestion control: throttle sender when network overloaded
- connection-oriented: setup required between client and server processes
- does not provide: timing, minimum throughput guarantee, security

UDP service:

- unreliable data transfer between sending and receiving process
- does not provide: reliability, flow control, congestion control, timing, throughput guarantee, security, or connection setup.

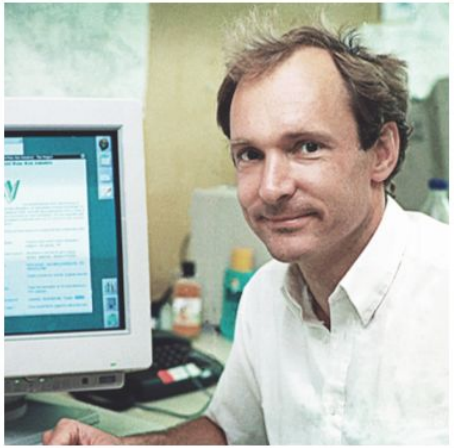
- Question: Why bother to have UDP?
- Answer: TCP is much heavier weight, and in many cases UDP is good enough (in addition to being simpler / faster)

Internet applications, and transport protocols

application	application layer protocol	transport protocol
file transfer/download	FTP [RFC 959]	TCP
e-mail	SMTP [RFC 5321]	TCP
Web documents	HTTP [RFC 7230, 9110]	TCP
Internet telephony	SIP [RFC 3261], RTP [RFC 3550], or proprietary	TCP or UDP
streaming audio/video	HTTP [RFC 7230], DASH	TCP
interactive games	WOW, FPS (proprietary)	UDP or TCP

The Web

The Web as we know it was founded in ~1990, by Tim Berners-Lee, physicist at CERN



Tim Berners-Lee Photo: CERN

His goal:

provide distributed access to data

The World Wide Web (WWW):

a distributed database of “pages”
linked together via the
Hypertext Transport Protocol (HTTP)

Why was the web so successful?

- Had networks in mind from the beginning

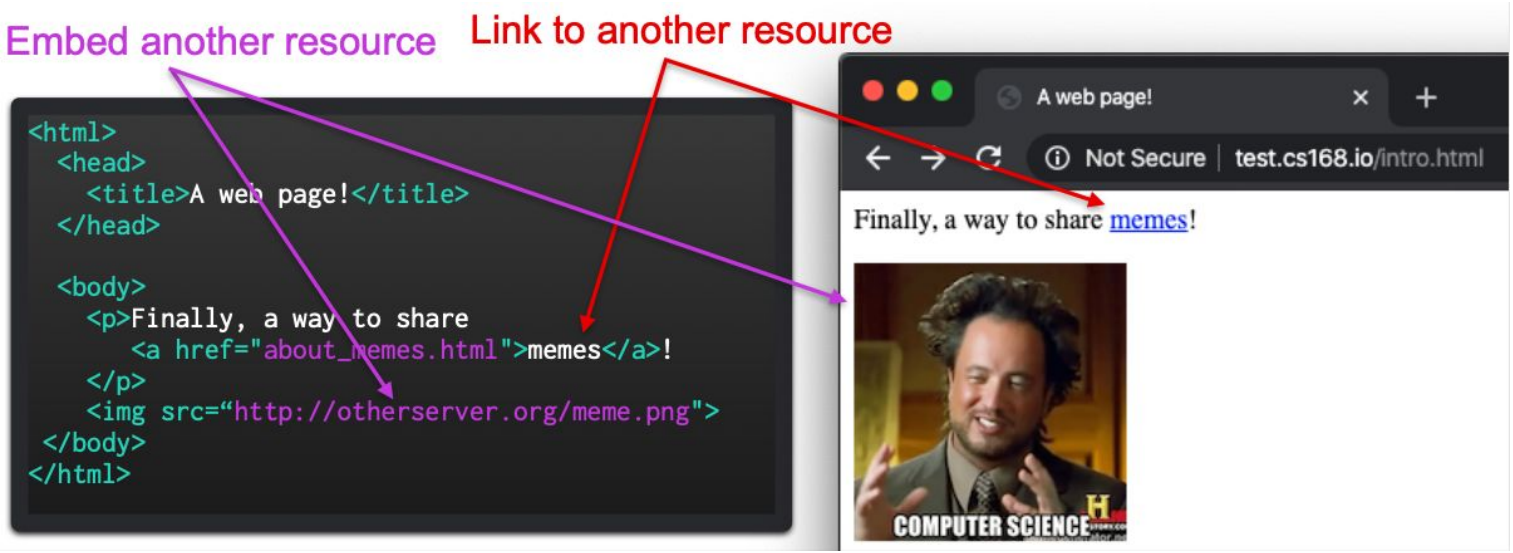
- What made it successful in the beginning is what makes it successful now
  - It gives a lot of leeway for how websites work (didn't over-specify)
  - Not tied to any one underlying system
  - No central authority — you can just add your own server/content
  - The ability to quickly navigate information from different sources

The web: basic requirements

- Something to represent content with links: **HTML**
- Client program to access/navigate/display content (e.g. HTML): **Web browser**
- A way to reference content: **URLs**
  - It's how you link/embed content to/in other content across a network
  - First general "handle" for arbitrary Internet content
  - Not just naming a host/processes (address/port)
- Something to host content: **Web servers**
- A protocol to get content from server to client: **HTTP**
  - Turns web URLs into TCP connections

Web basics

- HTML: HyperText Markup Language - Represent content with links
- Browser: Access/navigate/display content
- Provide integrated interface to scattered information



Web basics: URL syntax

scheme: //host[:port]/path/resource

scheme	Typically a protocol: http, ftp, https, smtp, rtsp, etc.
host	DNS hostname or IP address
port	Defaults to protocol's standard port e.g. http: 80 https: 443
path	Traditionally reflecting file system
resource	Identifies the desired resource (traditionally a file)

The web: basic requirements

- Something to represent content with links: **HTML**
- Client program to access/navigate/display content (e.g. HTML): **Web browser**
- A way to reference content: **URLs**
  - It's how you link/embed content to/in other content across a network
  - First general "handle" for arbitrary Internet content
  - Not just naming a host/processes (address/port)
- Something to host content: **Web servers**
- A protocol to get content from server to client: **HTTP**
  - Turns web URLs into TCP connections

HyperText Transfer Protocol (HTTP)

- Focusing our discussion on common/current versions of HTTP:
  - HTTP 1.0 (1996) and HTTP 1.1 (1997)
  - These are (significant) outgrowth of original “HTTP 0.9”
- HTTP 2 published in 2015
  - Largely based on work by Google
  - As of 2020, 44% of websites use it
  - Significant departure; largely performance optimizations
- HTTP 3 forthcoming standard
  - Largely based on work by Google
  - As of 2020, 5% of websites use it (more or less Google and Facebook?)
  - Significant departure; largely performance optimizations

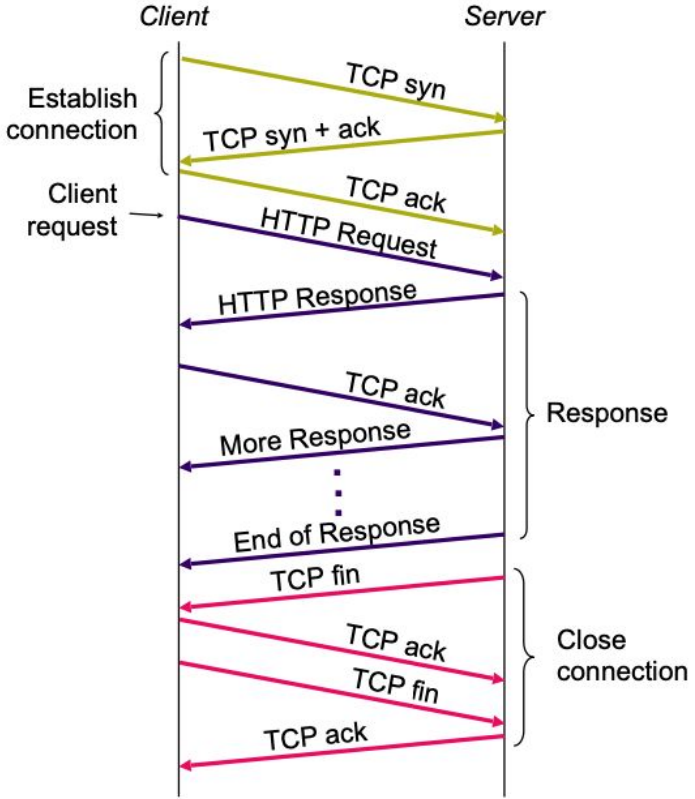
HyperText Transfer Protocol (HTTP)

- (Simple HTTP 1.0 “GET” request)

- Client creates TCP connection (port 80)
- Client sends request

- Server sends response packets
- Client ACKs them

- Server closes connection

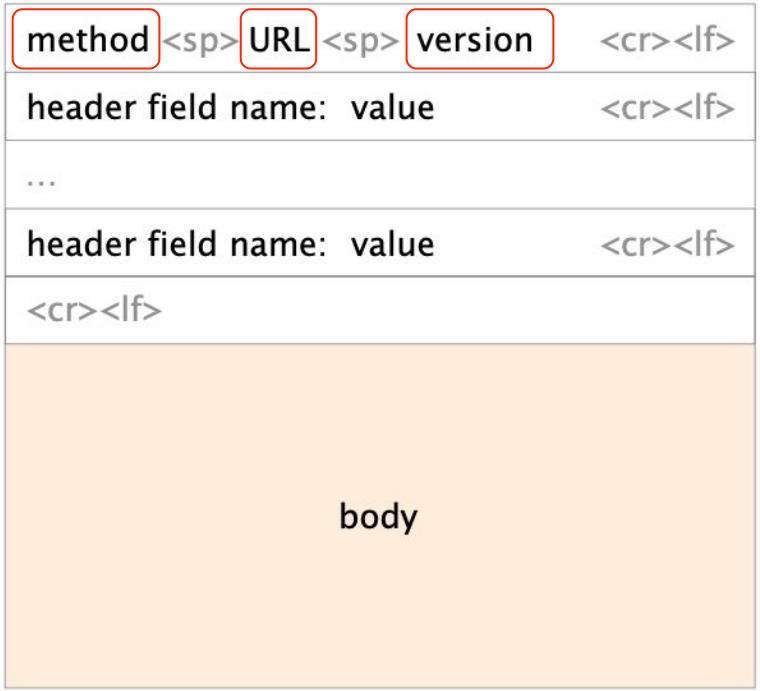


HTTP client requests

HTTP request

HTTP client requests

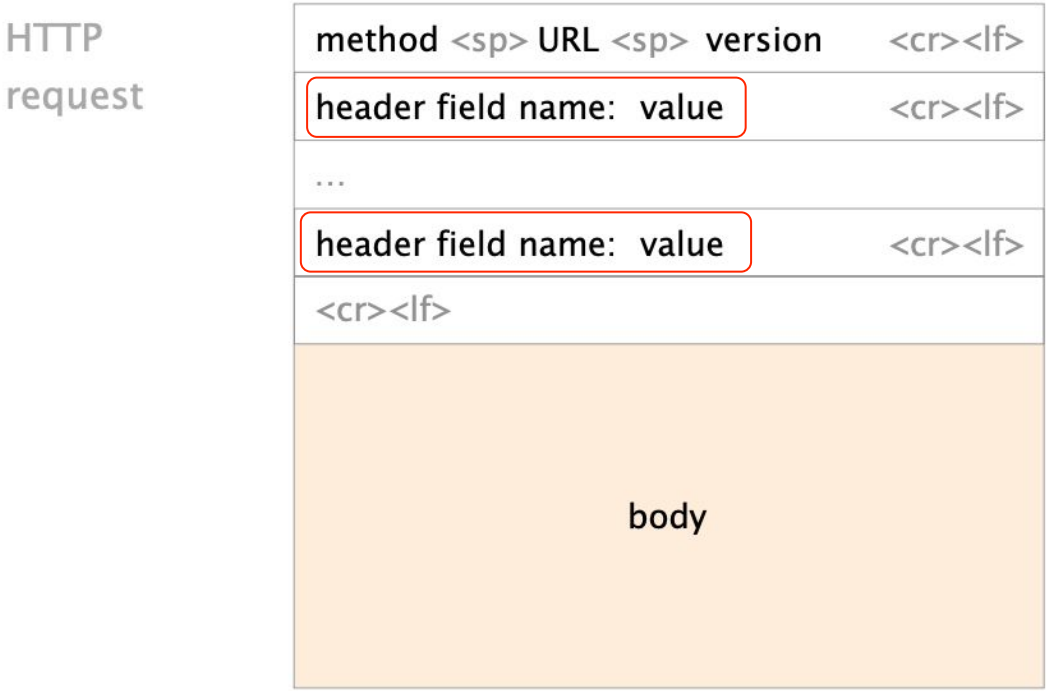
HTTP request



HTTP client requests

method	GET	return resource
	HEAD	return headers only
	POST	send data to server (forms)
URL		relative to server (*e.g.*, /index.html)
version		1.0, 1.1, 2.0

HTTP client requests



Request headers are variable length but still human readable

Uses

Authorization info

Acceptable document types/encoding

From (user email)

Host (identify the server to which the request is sent)

If-Modified-Since

Referrer (cause of the request)

User Agent (client software)

Request headers are variable length but still human readable

Uses	Authorization info
	Acceptable document types/encoding
	From (user email)
	Host (identify the server to which the request is sent)

Why would you need this? You're already connected?

Request headers are variable length but still human readable

Uses

Authorization info

Acceptable document types/encoding

From (user email)

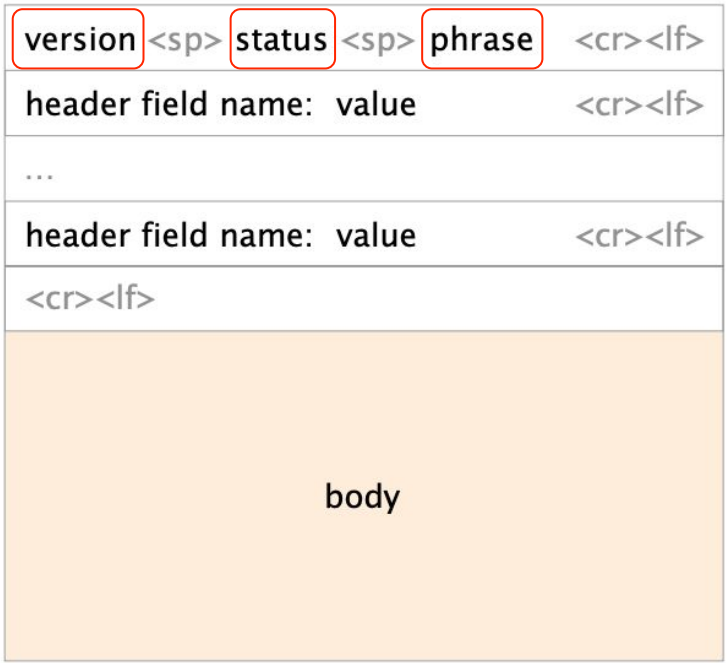
Host (identify the server to which the request is sent)

Why would you need this? You're already connected?

In shared infrastructures (AWS, GCP, Fastly, etc.) multiple names can be mapped to a single IP address - known as *virtual hosting*. More on this when we discuss CDNs

HTTP server responses

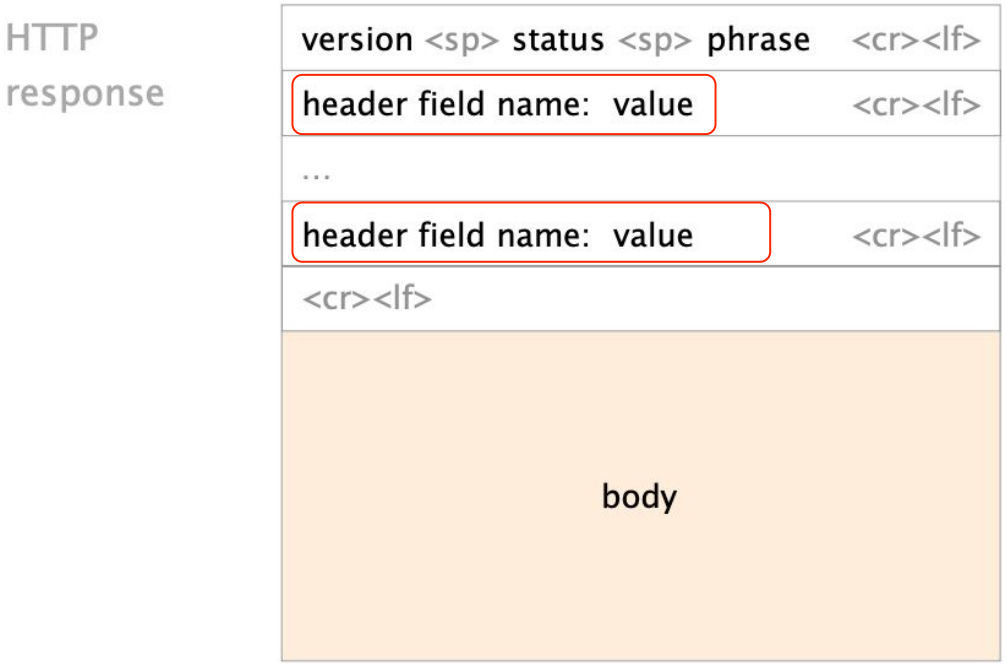
HTTP response



HTTP server responses

	3 digit response code			reason phrase
Status	1XX	informational		
	2XX	success	200	OK
	3XX	redirection	301	Moved Permanently
			303	Moved Temporarily
			304	Not Modified
	4XX	client error	404	Not Found
	5XX	server error	505	Not Supported

HTTP server responses



Like request headers, response headers are of variable lengths and human-readable

Uses

Location (for redirection)

Allow (list of methods supported)

Content encoding (*e.g.*, gzip)

Content-Length

Content-Type

Expires (caching)

Last-Modified (caching)