CSCI 6907.11

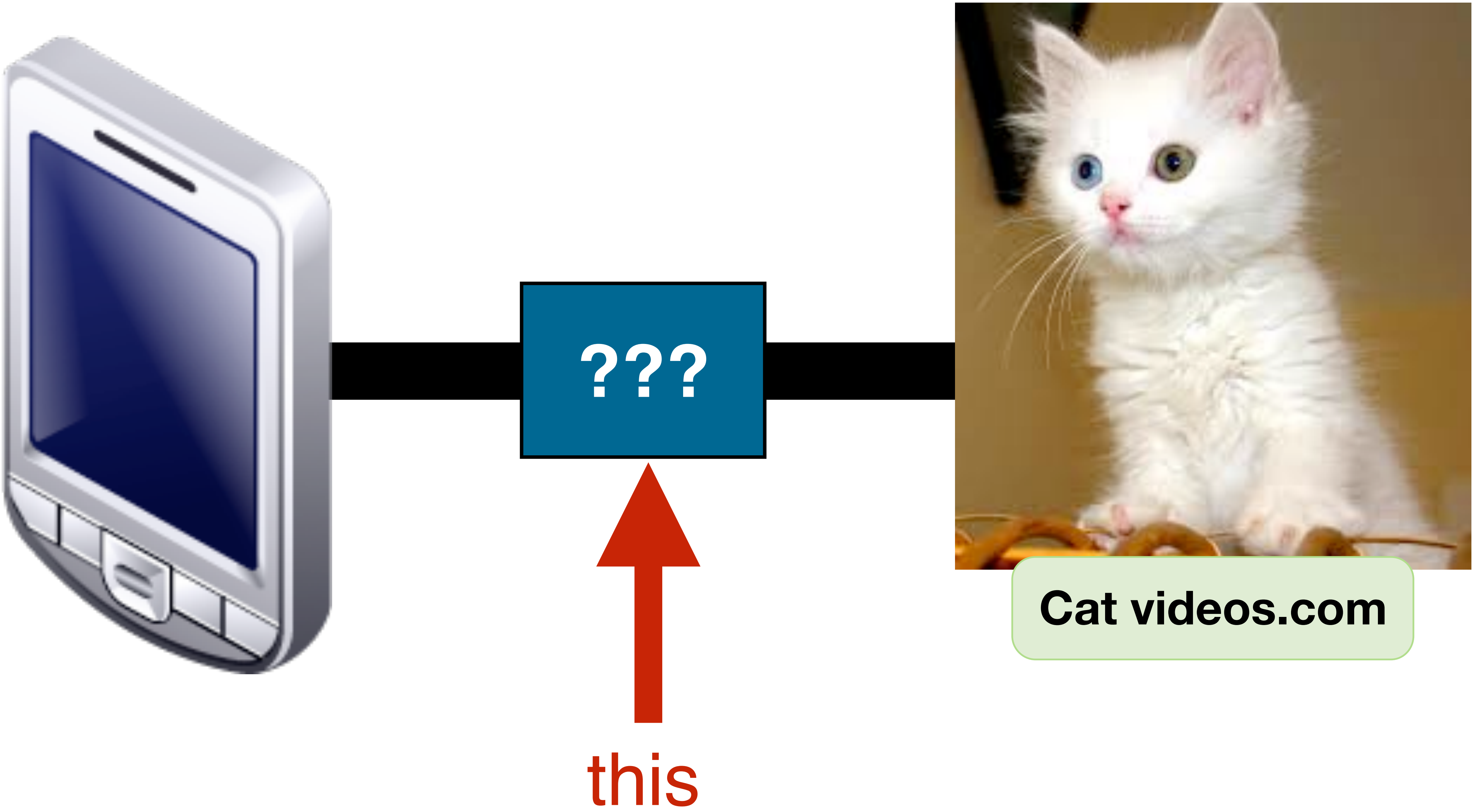
Adv. Net. Sys. Prog.

Lecture 1

Tim Wood
CS@GWU

2015

What is this course about?



???

Cat videos.com

this

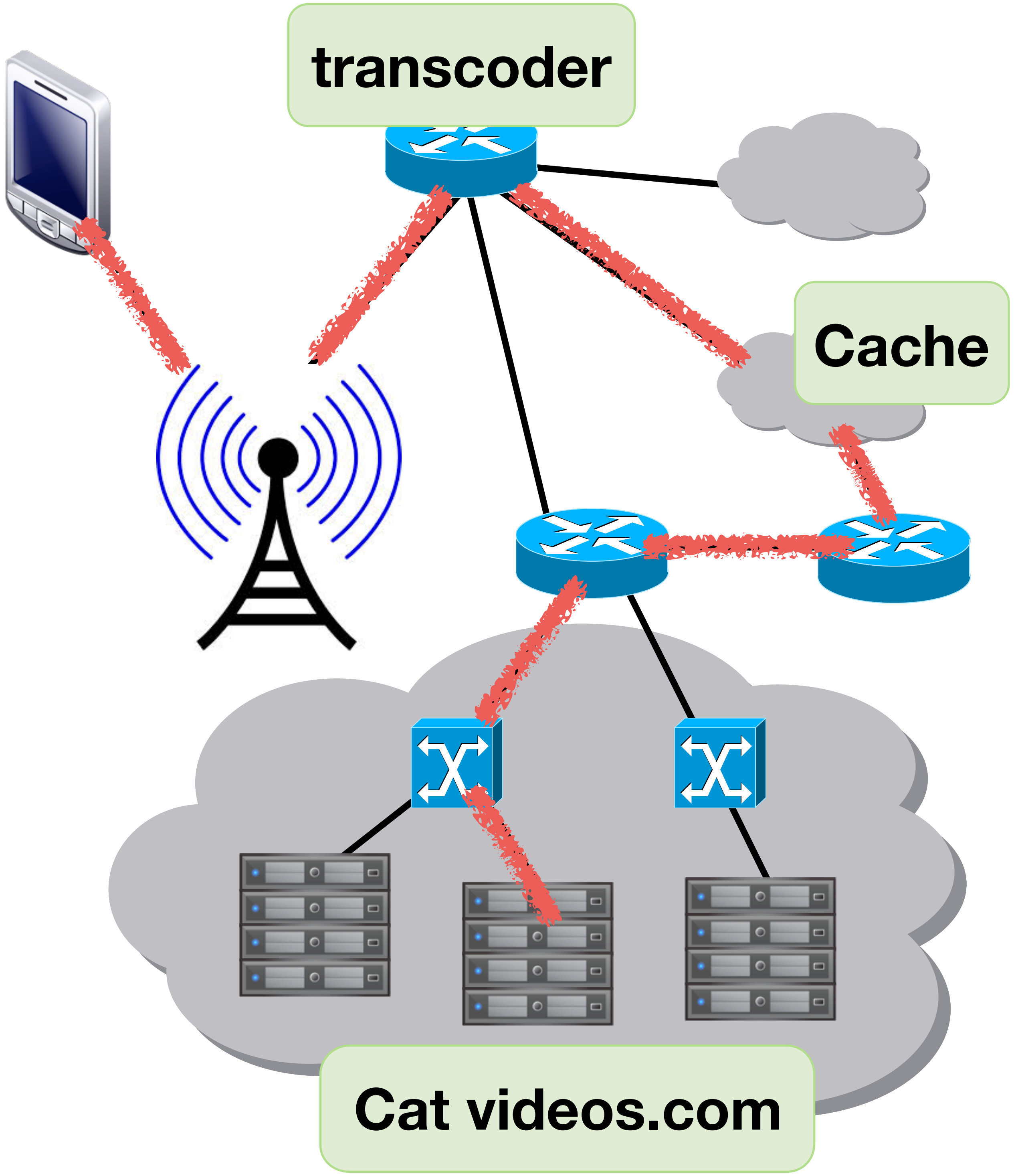
Software Based Networks

SDN: Centralized network **control plane**

- Software rules make decisions for how messages are routed

NFV: Software **data plane**

- Software replaces hardware switches/routers
- Can perform complex processing on individual network flows



Course Overview

Professor: Tim Wood

Class Time: Mondays 1-3:30PM SEH 1450

More of a “lab” than a “lecture”

Come prepared to code!

Prerequisites:

- Advanced OS course for grads
- Gabe’s OS course for undergrads

Ground Rules

No laptops during lecture segments

- I promise they won't be too long (except maybe today)

Be respectful and responsible

- Some students have to come late, leave early—do it quietly!
- It is your responsibility to find out what you miss

Team coding is fine for team projects

- But not for individual assignments (rare and clearly marked)

Be active in class

- If you aren't asking at least three questions per class you are wasting your own time

Today

What are networks?

What are the key network abstractions?

What are protocols?

March 9th

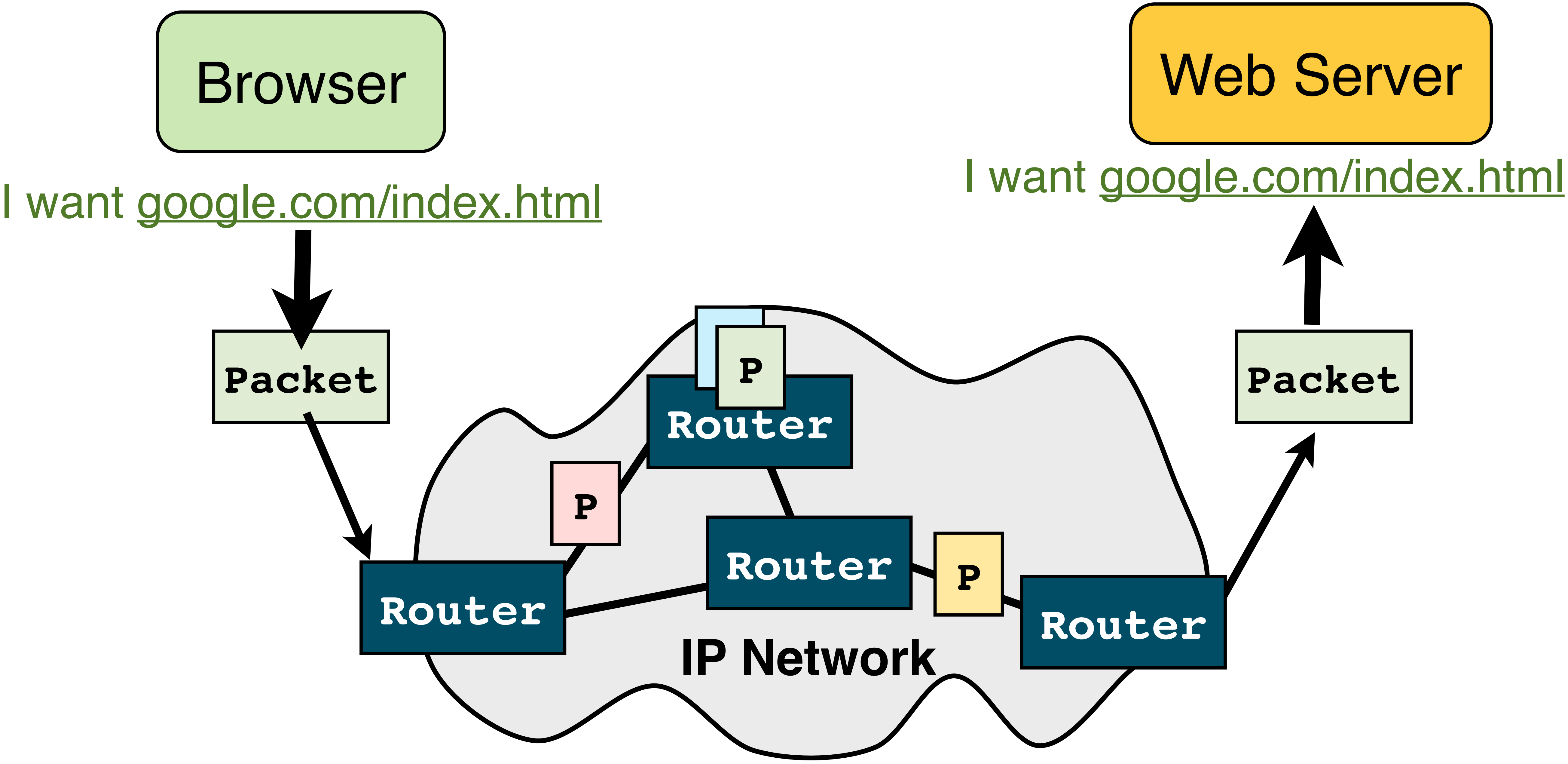
GW is running a competition to build network service apps on top of GENI

$10,000 in prize money given out to top 3 projects!

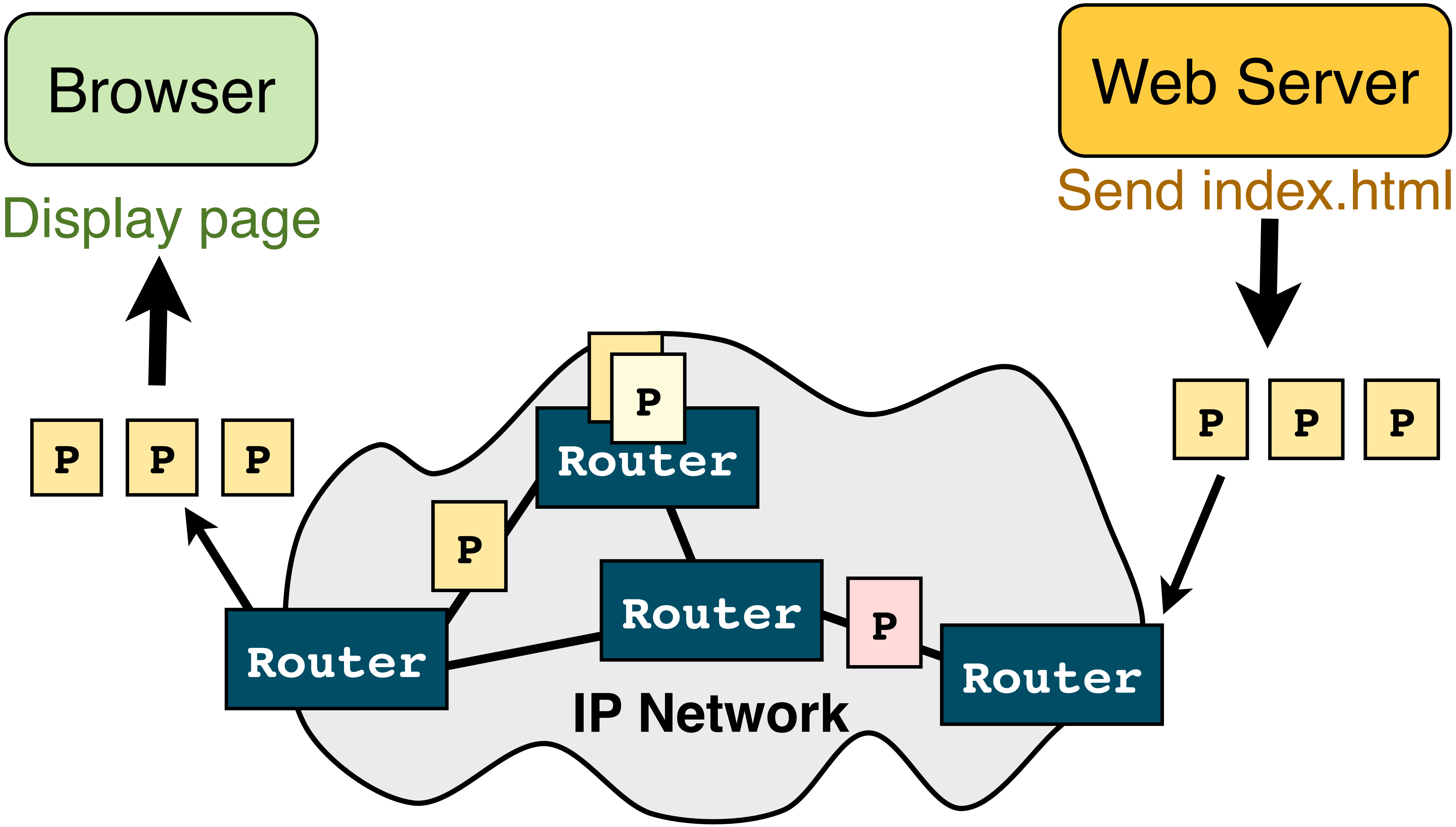
Monday Jan 26th we will have GENI tutorial

Win!

How the Web Works



How the Web Works



Traveling the Interwebs

Writes to a **socket** are split into **packets**

- Fixed size chunk of data (about 1KB)
- Some messages fit in one packet, others require many

Packets use **routers** to traverse the network

- Packet contains header information including the IP address and port it is destined for
- Router directs the packet to the next "hop"

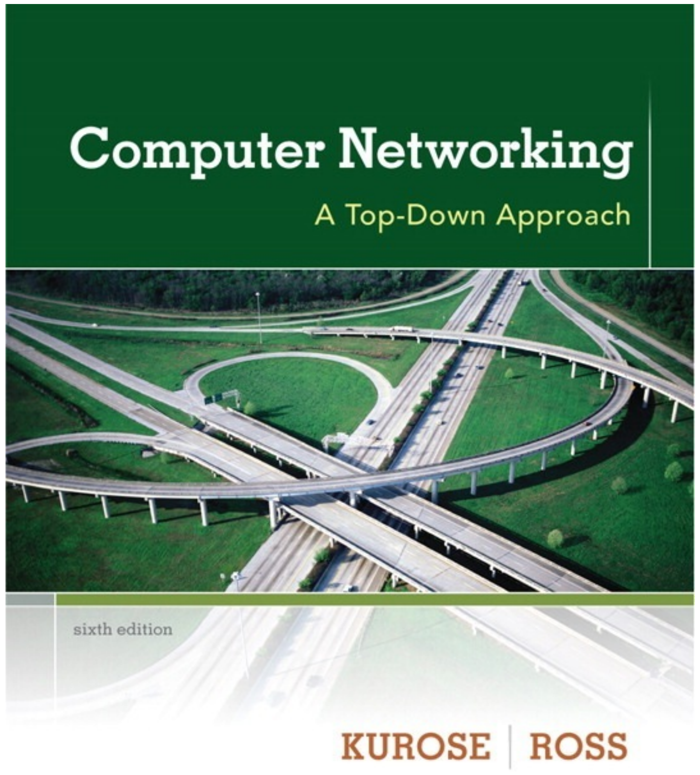
All of this is magically taken care of for you by the operating system / network drivers

- Your code doesn't need to deal with low level networking
- (Unless you are in this course)

How?

A note…

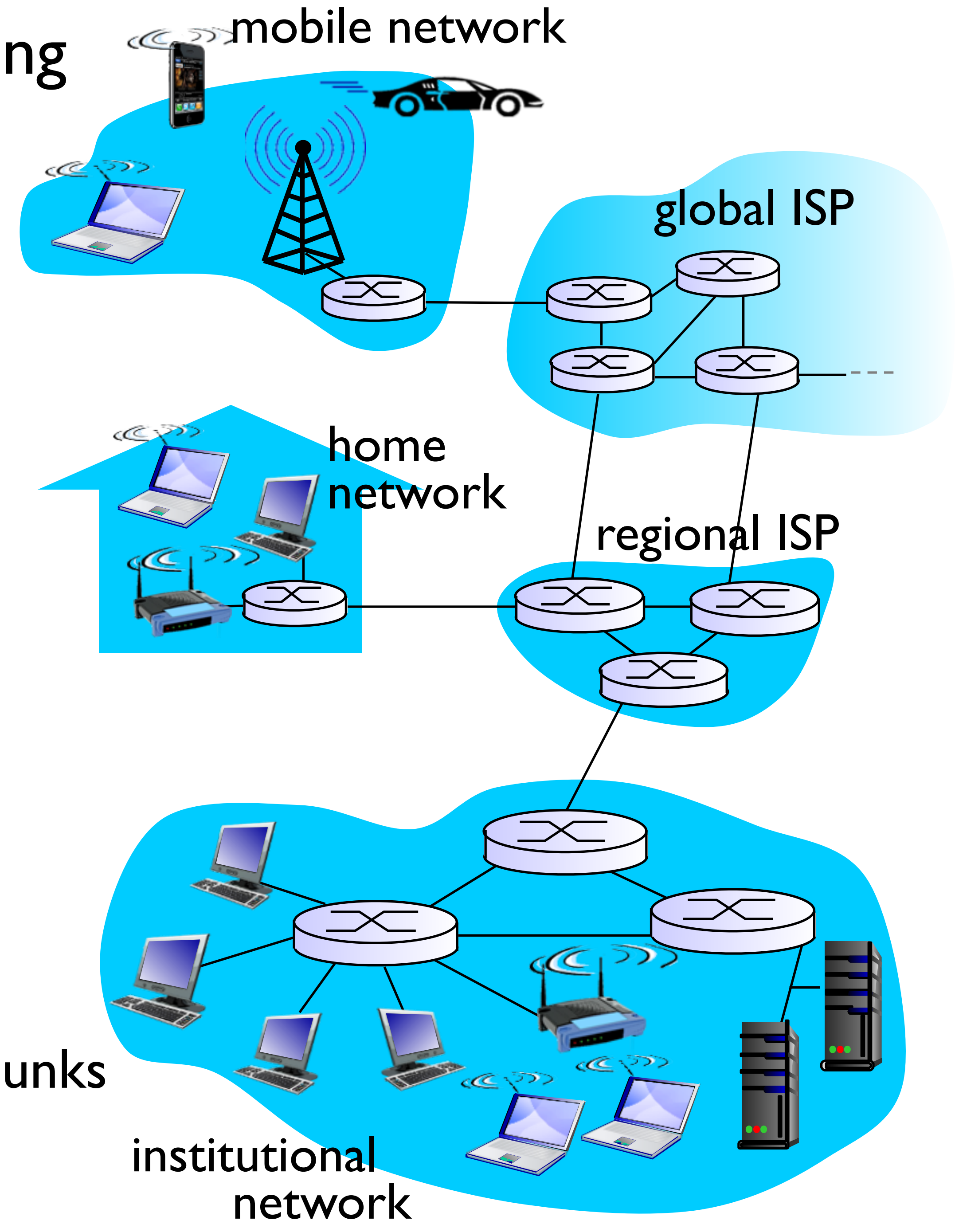
Slides that look like this come from:



Computer Networking: A Top Down Approach
6th edition
Jim Kurose, Keith Ross
Addison-Wesley
March 2012

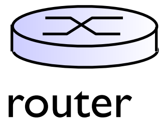
What's the Internet: "nuts and bolts" view

- millions of connected computing devices:
  - *hosts* = *end systems*
  - running *network apps*



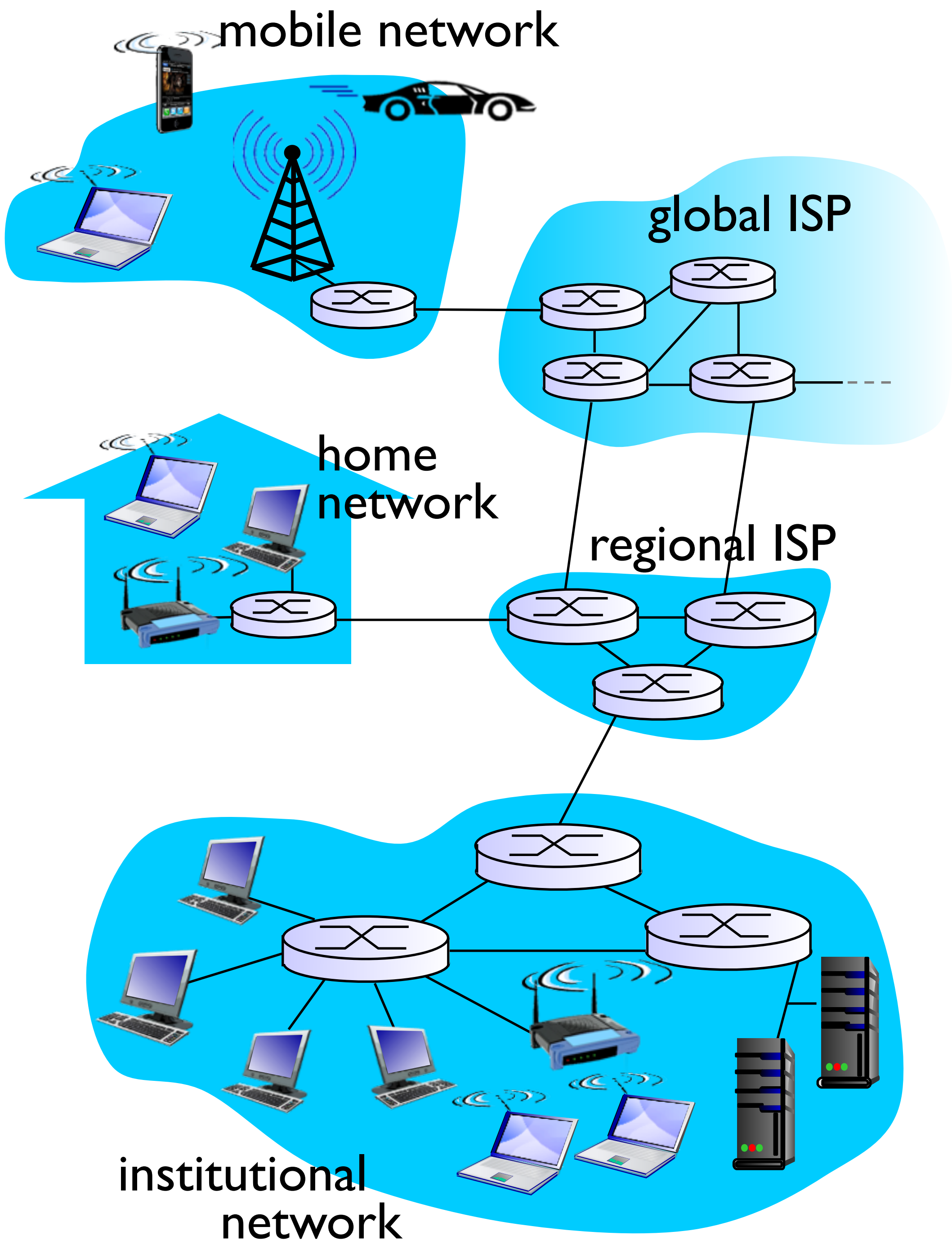
- *communication links*
  - fiber, copper, radio, satellite
  - transmission rate: *bandwidth*

- *Packet switches:* forward packets (chunks of data)
  - *routers* and *switches*



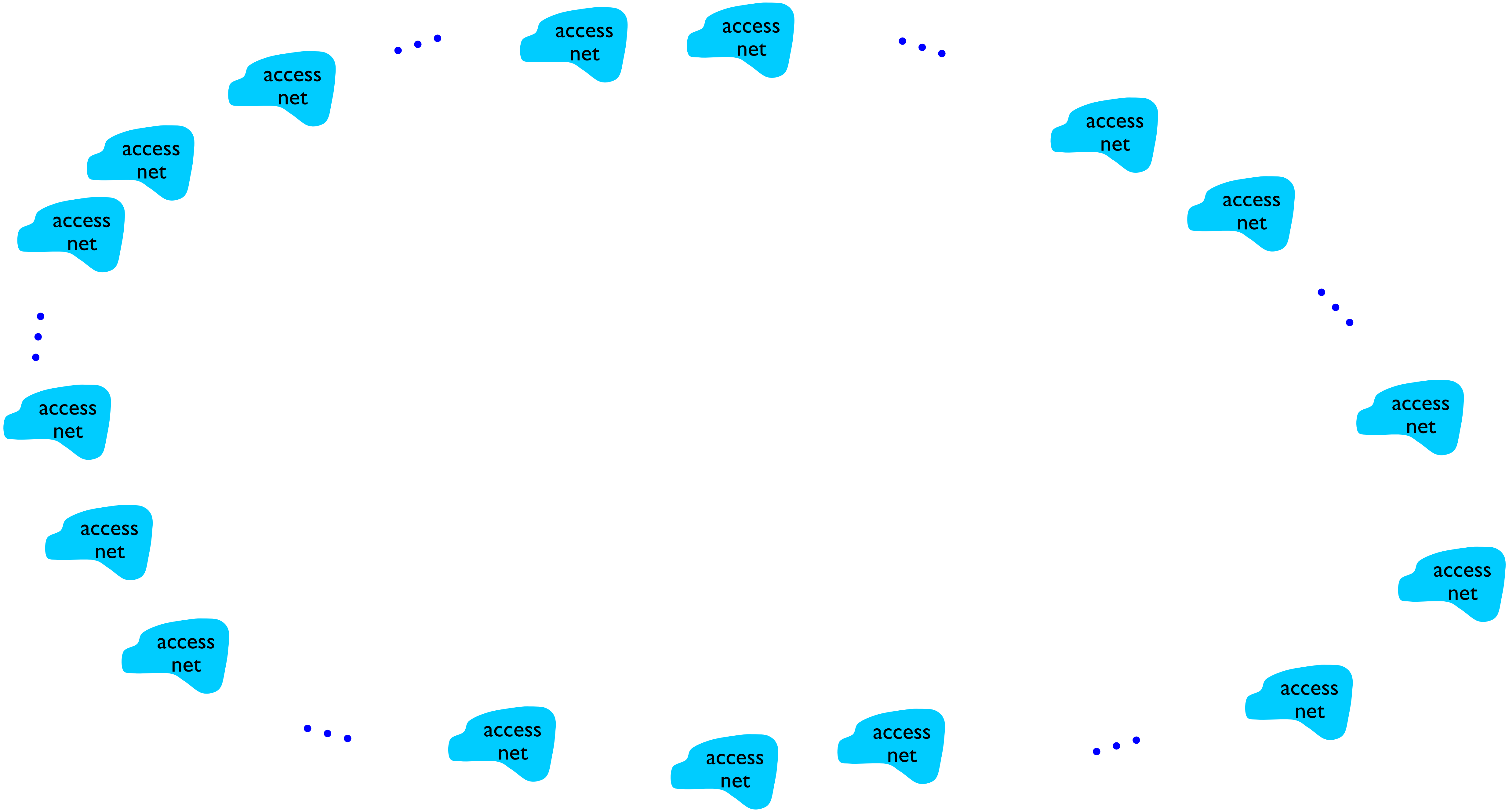
What's the Internet: a service view

- *Infrastructure that provides services to applications:*
  - Web, VoIP, email, games, e-commerce, social nets, …
- *provides programming interface to apps*
  - hooks that allow sending and receiving app programs to "connect" to Internet
  - provides service options, analogous to postal service



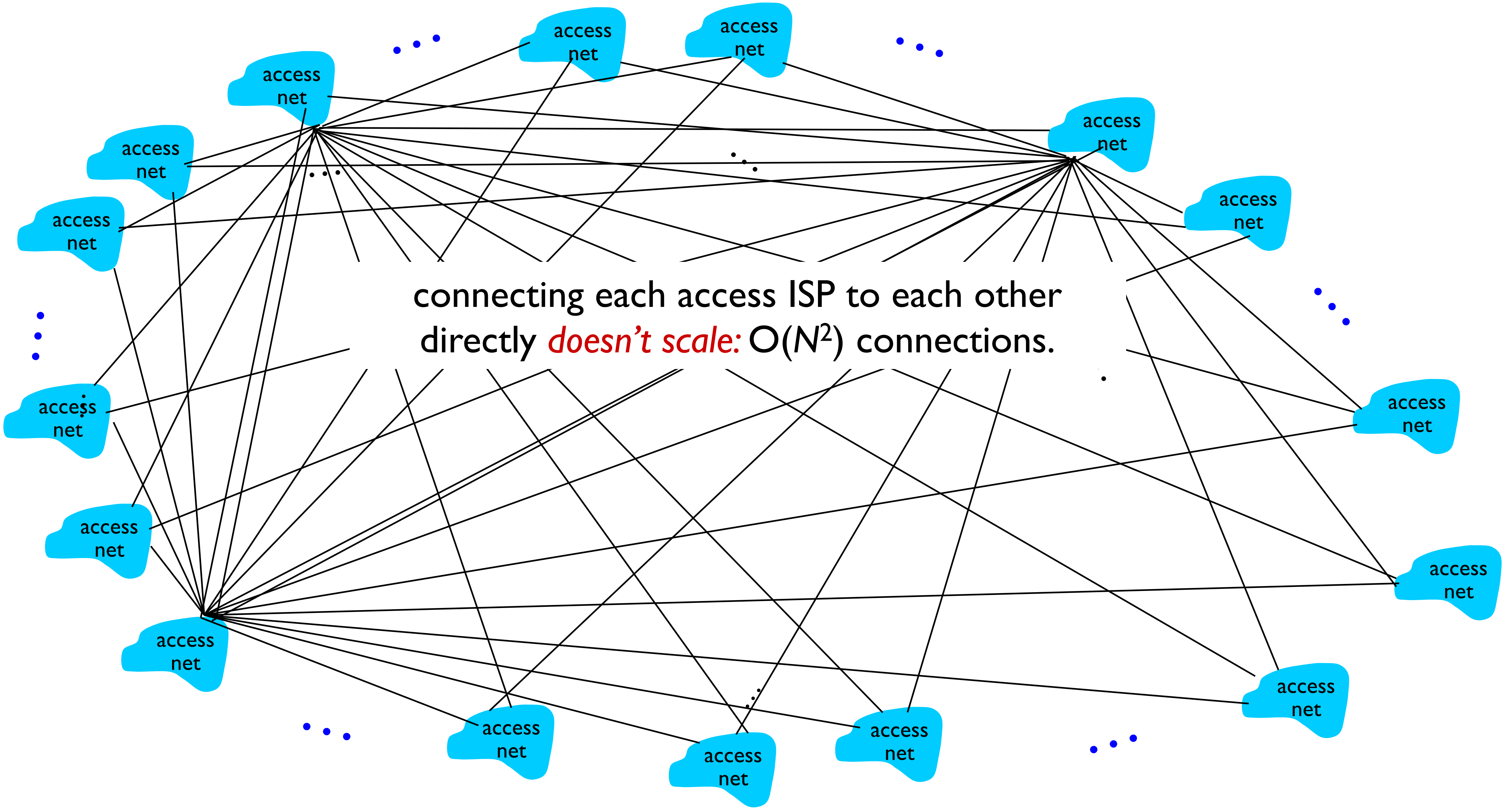
Internet structure: network of networks

Question: given *millions* of access ISPs, how to connect them together?



Internet structure: network of networks

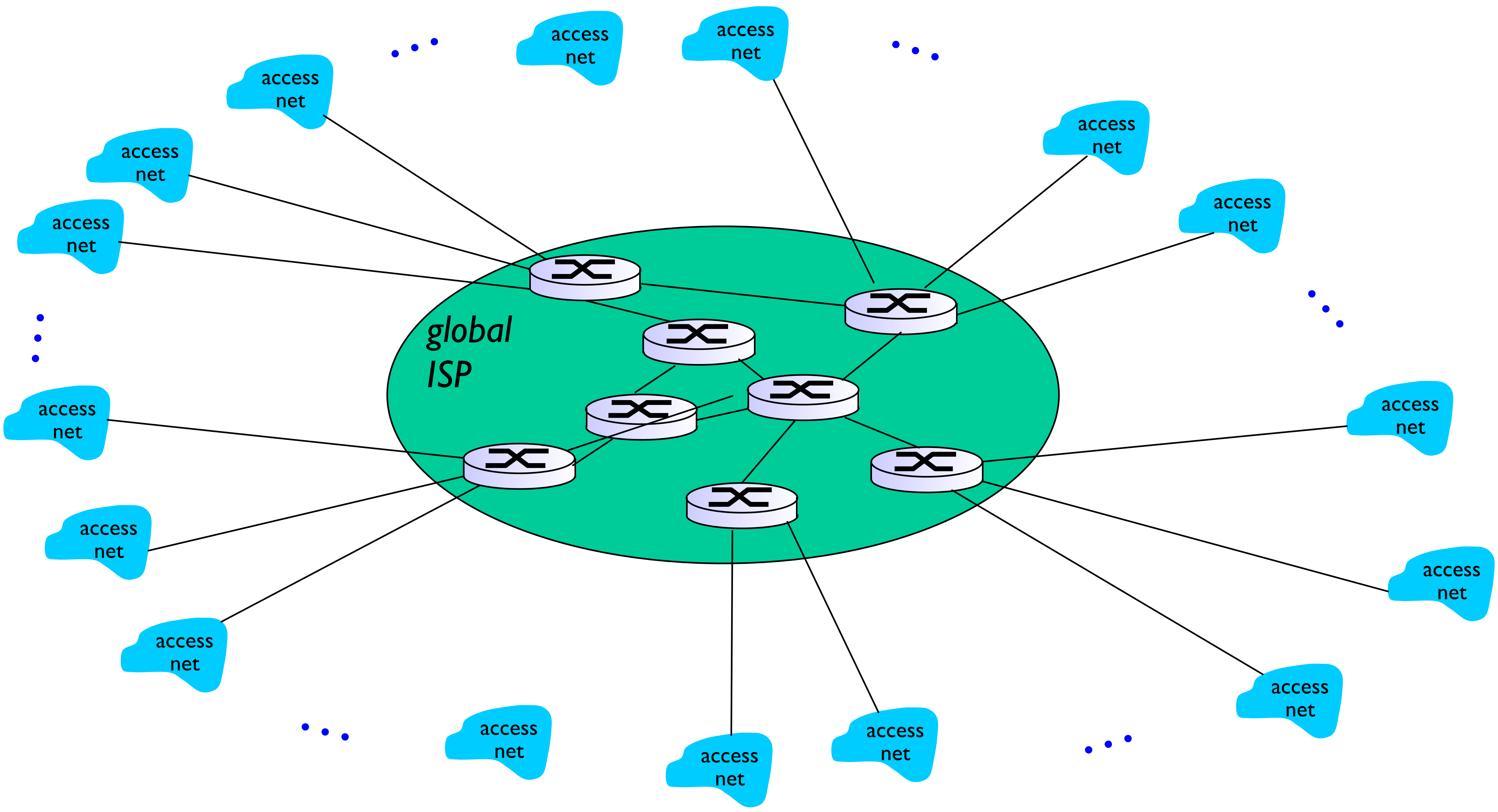
Option: connect each access ISP to every other access ISP?



connecting each access ISP to each other directly *doesn't scale:* O(N^2) connections.

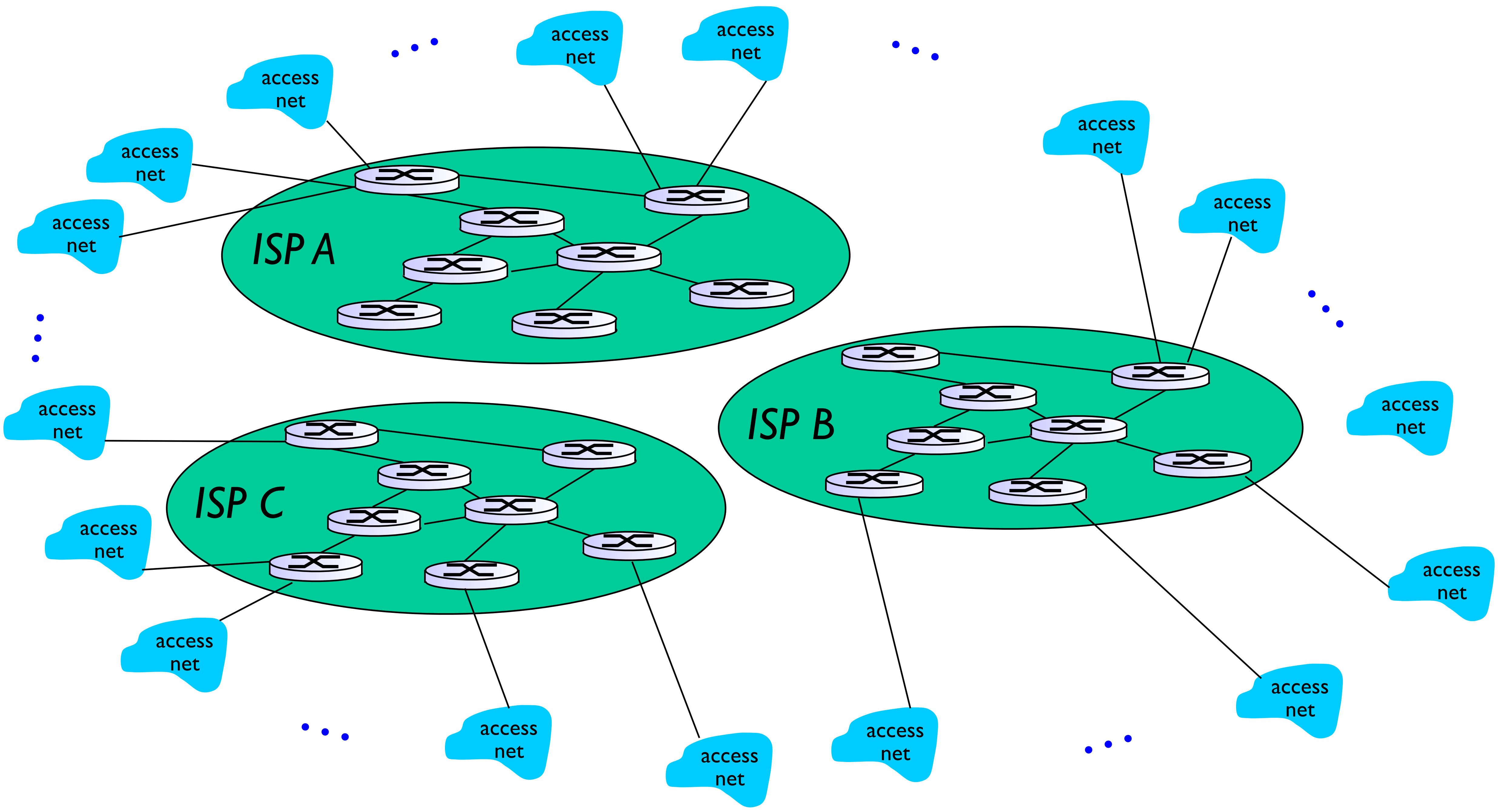
Internet structure: network of networks

Option: connect each access ISP to a global transit ISP? Customer and provider ISPs have economic agreement.



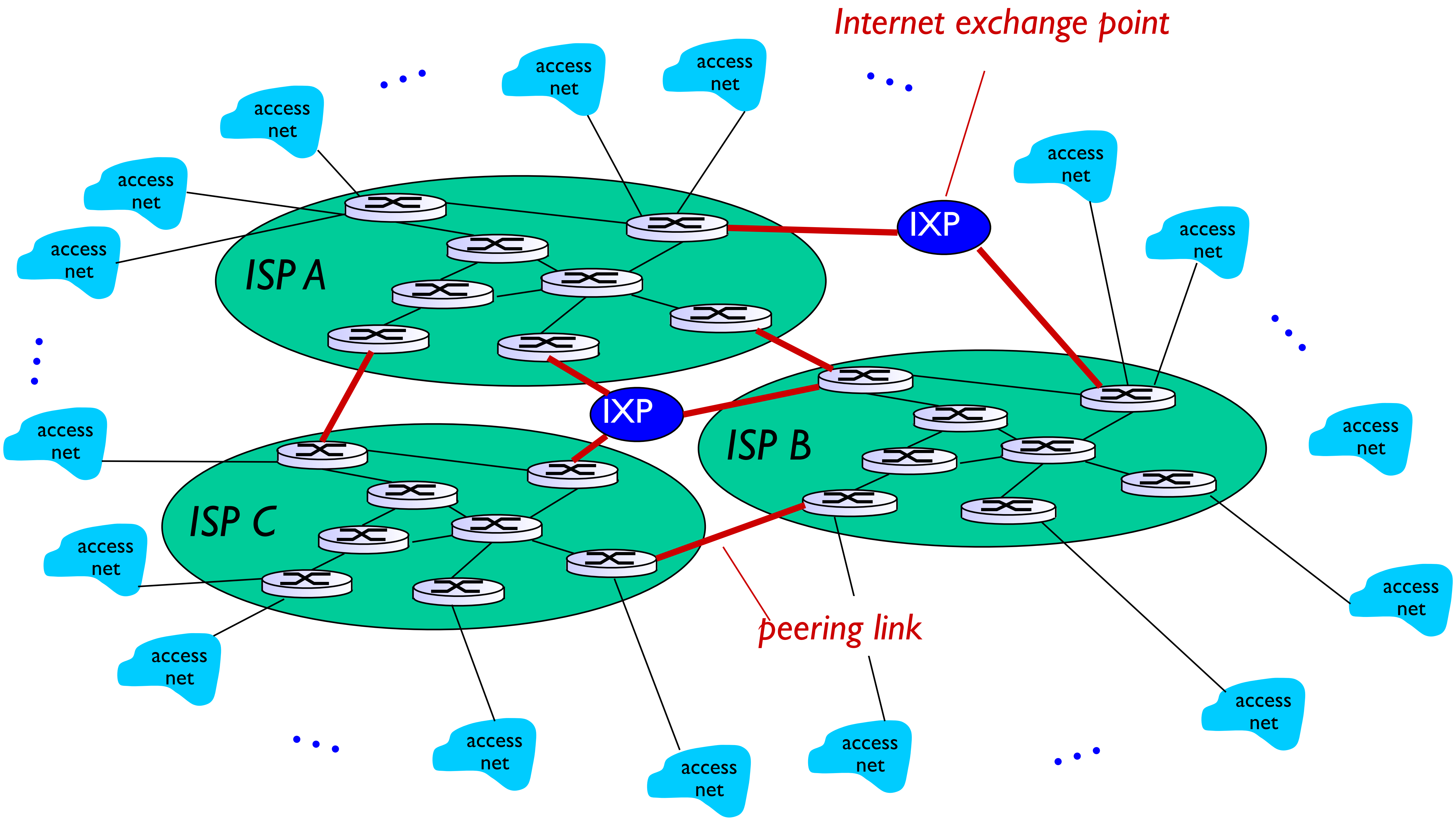
Internet structure: network of networks

But if one global ISP is viable business, there will be competitors ….



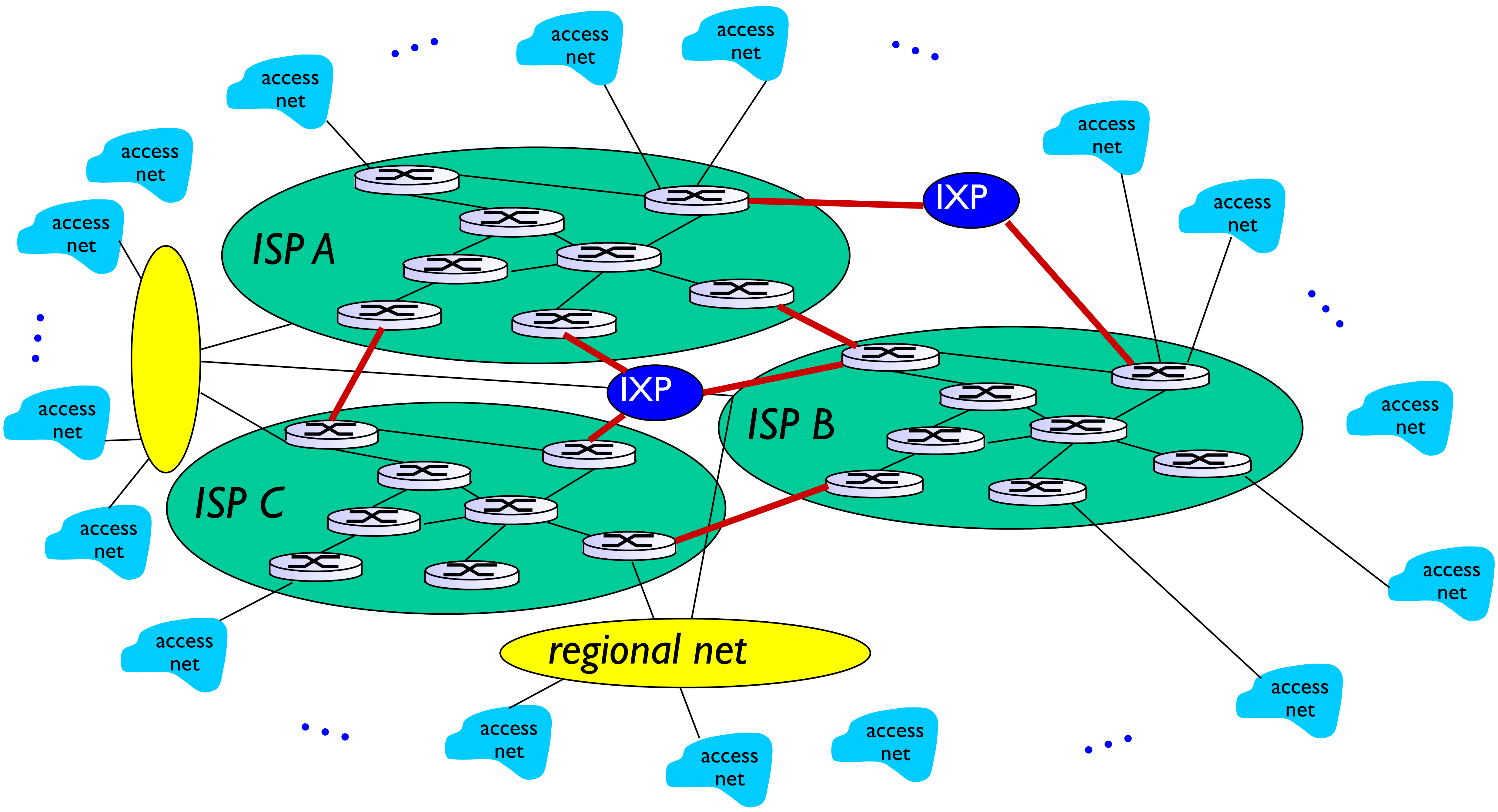
Internet structure: network of networks

But if one global ISP is viable business, there will be competitors …. which must be interconnected



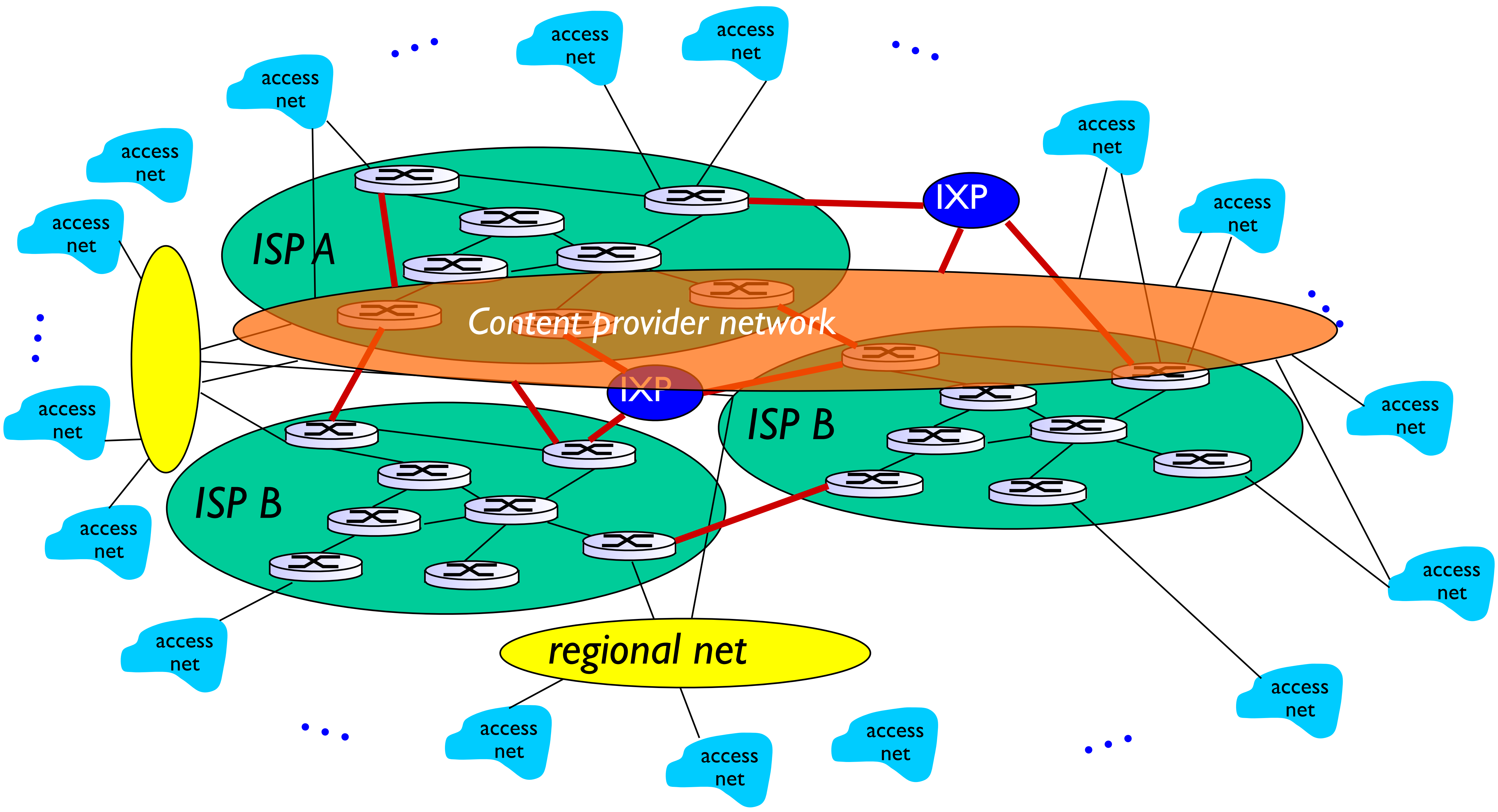
Internet structure: network of networks

… and regional networks may arise to connect access nets to ISPS

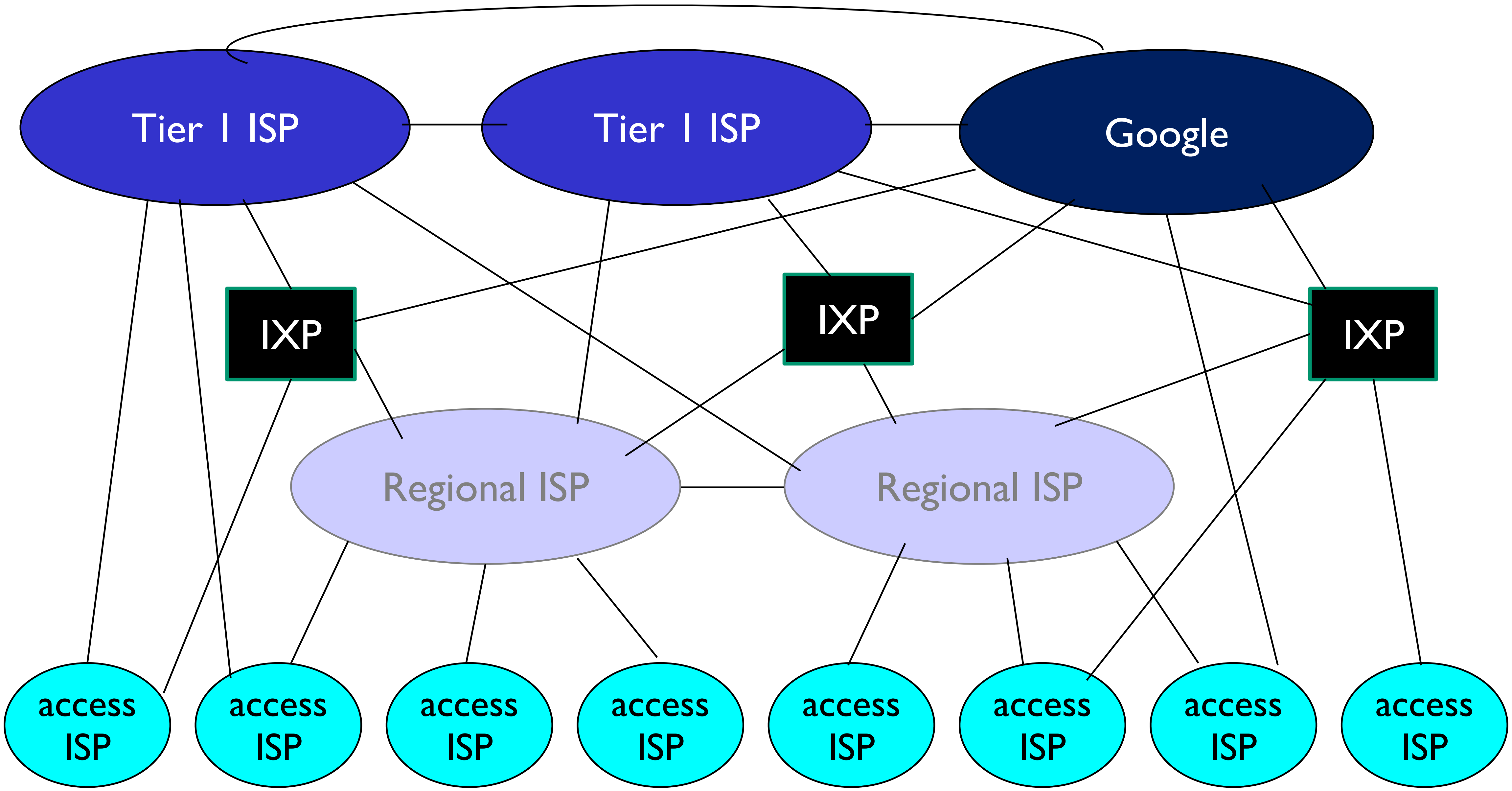


Internet structure: network of networks

… and content provider networks (e.g., Google, Microsoft, Akamai) may run their own network, to bring services, content close to end users



Internet structure: network of networks



- at center: small # of well-connected large networks
  - "tier-1" commercial ISPs (e.g., Level 3, Sprint, AT&T, NTT), national & international coverage
  - content provider network (e.g, Google): private network that connects it data centers to Internet, often bypassing tier-1, regional ISPs

Networks of Networks

This is looking really complicated.

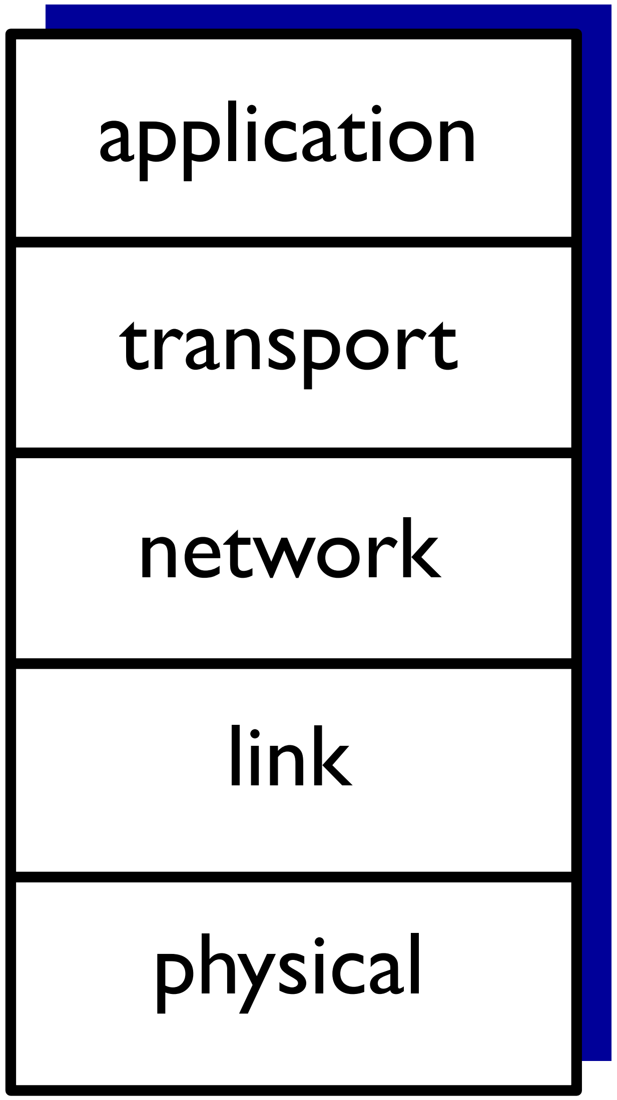
We use an abstraction layer!

Or better yet, 5-7 layers!

How do we deal with complexity in computer science?

Internet protocol stack

- *application:* supporting network applications
  - FTP, SMTP, HTTP
- *transport:* process-process data transfer
  - TCP, UDP
- *network:* routing of datagrams from source to destination
  - IP, routing protocols
- *link:* data transfer between neighboring network elements
  - Ethernet, 802.111 (WiFi), PPP
- *physical:* bits "on the wire"



(Full OSI model has 7 layers, 2 don't matter)

Application Layer

Let’s try to reverse engineer HTTP!

- Hypertext Transfer Protocol (that means web sites if you weren’t alive in the 90s)

- **man** will open the manual for anything
- **tcpdump** is a packet monitoring tool
- **ping** can give you the IP for a domain name
- **wget** will download a single html file (no JS, images, etc)

Try to cleanly intercept an HTTP request and response for a simple website like

http://faculty.cs.gwu.edu/~timwood/simple.html

Or you could read: http://www.w3.org/Protocols/HTTP/1.0/spec.html

HTTP Basics

HTTP is the protocol used for web sites

- Text based, request-response scheme

When a web browser opens a page it sends:

```
GET /somedir/somepage.html HTTP/1.0
<empty>
```

Blank line indicates end of request

Sent as a plain text string

- Browser can add optional information about who is making the request before the blank line

HTTP Reply

The server responds to a GET request with:

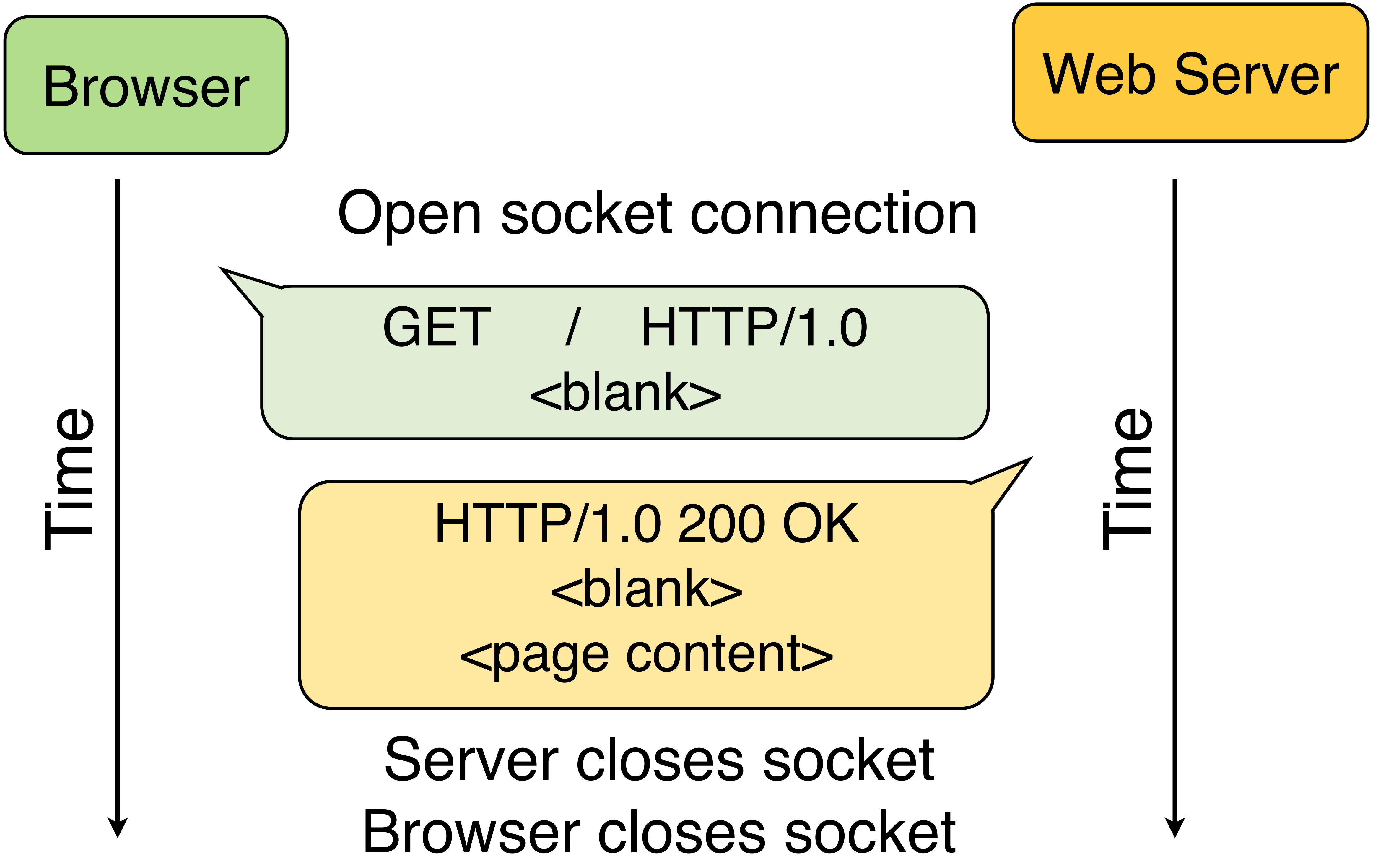
```
HTTP/1.0 200 OK
Date: Fri, 31 Dec 1999 23:59:59 GMT
Content-Type: text/html
Content-Length: 1354

<html><body>
... content of page ...
</body></html>
```

Header information followed by an empty line, then the requested content

Usually works even if
you leave out the header info

Web Protocol



Browser

Web Server

Open socket connection

GET / HTTP/1.0
<blank>

HTTP/1.0 200 OK
<blank>
<page content>

Server closes socket
Browser closes socket

Simple Web Server

A web server supports several types of requests

GET /path/file.html HTTP/1.0

- Return the specified file

actually, most use **`HTTP/1.1`** which is similar

HEAD /path/file.html HTTP/1.0

- Return header data about the file: modified date, file size, etc

POST /path/file.html HTTP/1.0

- The client also sends form data before the blank line
- The server uses the data for some kind of processing, then returns a result page

The Simplest Browser

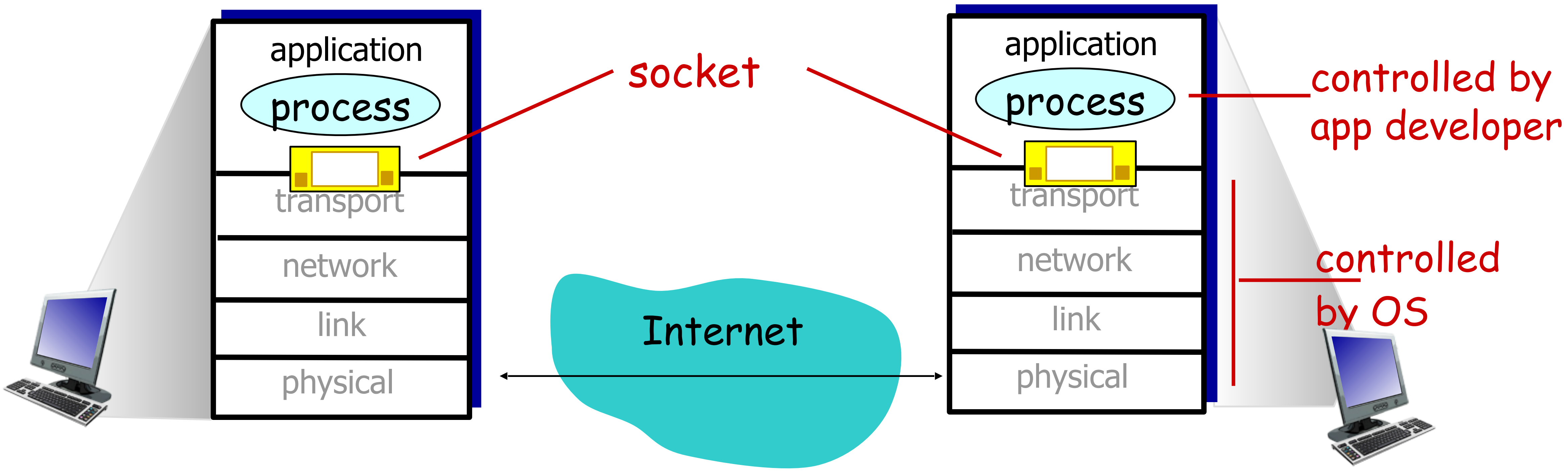
Who needs Firefox/Chrome/Safari/Netscape/Opera/Mosiac/Internet Explorer?!

You’ve got `telnet`!

```
telnet www.cs.gwu.edu 80
GET / HTTP/1.0
<empty>
```

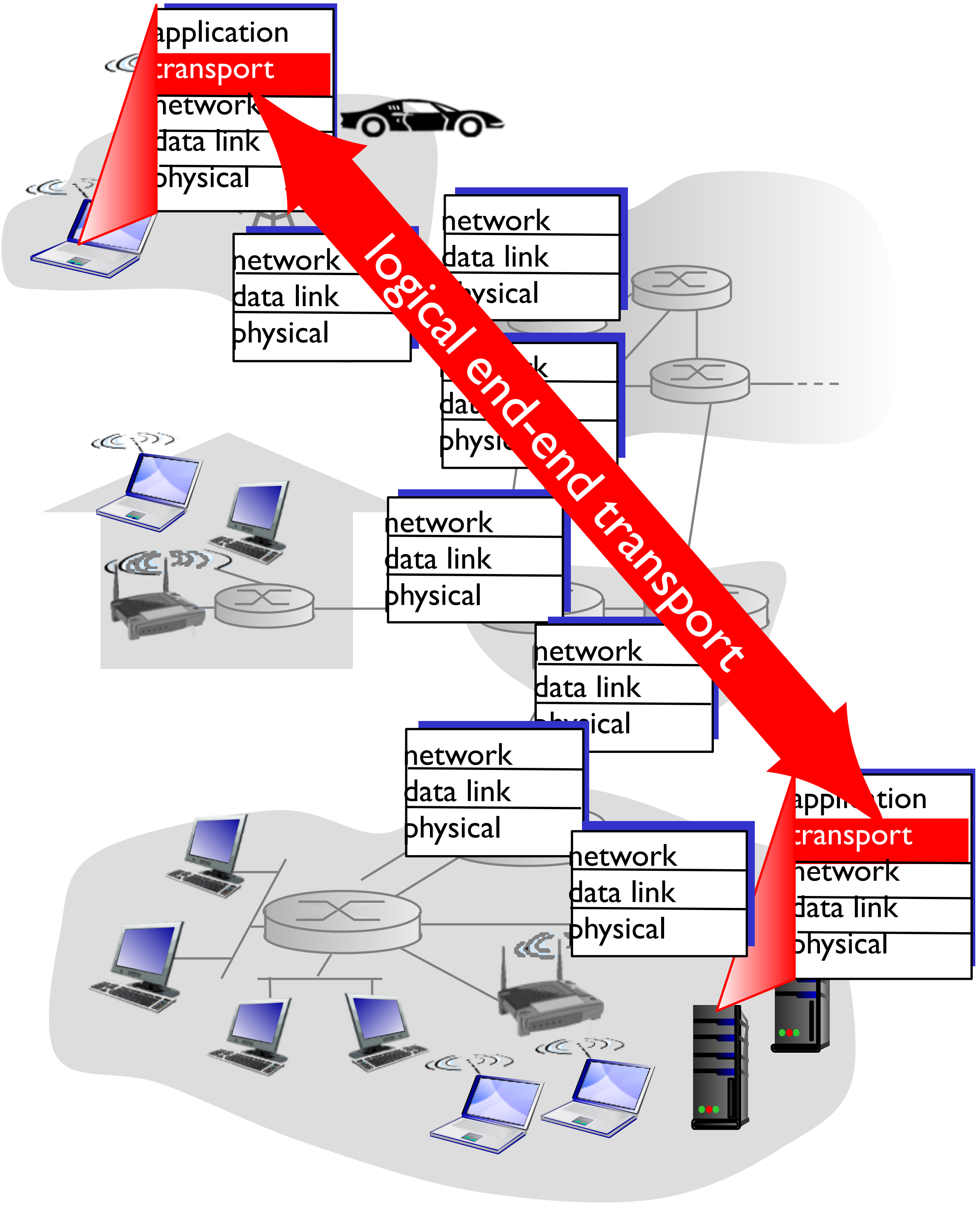
Sockets

- process sends/receives messages to/from its socket
- socket analogous to door
  - sending process shoves message out door
  - sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving process



Internet transport-layer protocols

- reliable, in-order delivery (TCP)
  - congestion control
  - flow control
  - connection setup
- unreliable, unordered delivery: UDP
  - no-frills extension of “best-effort” IP
- services not available:
  - delay guarantees
  - bandwidth guarantees

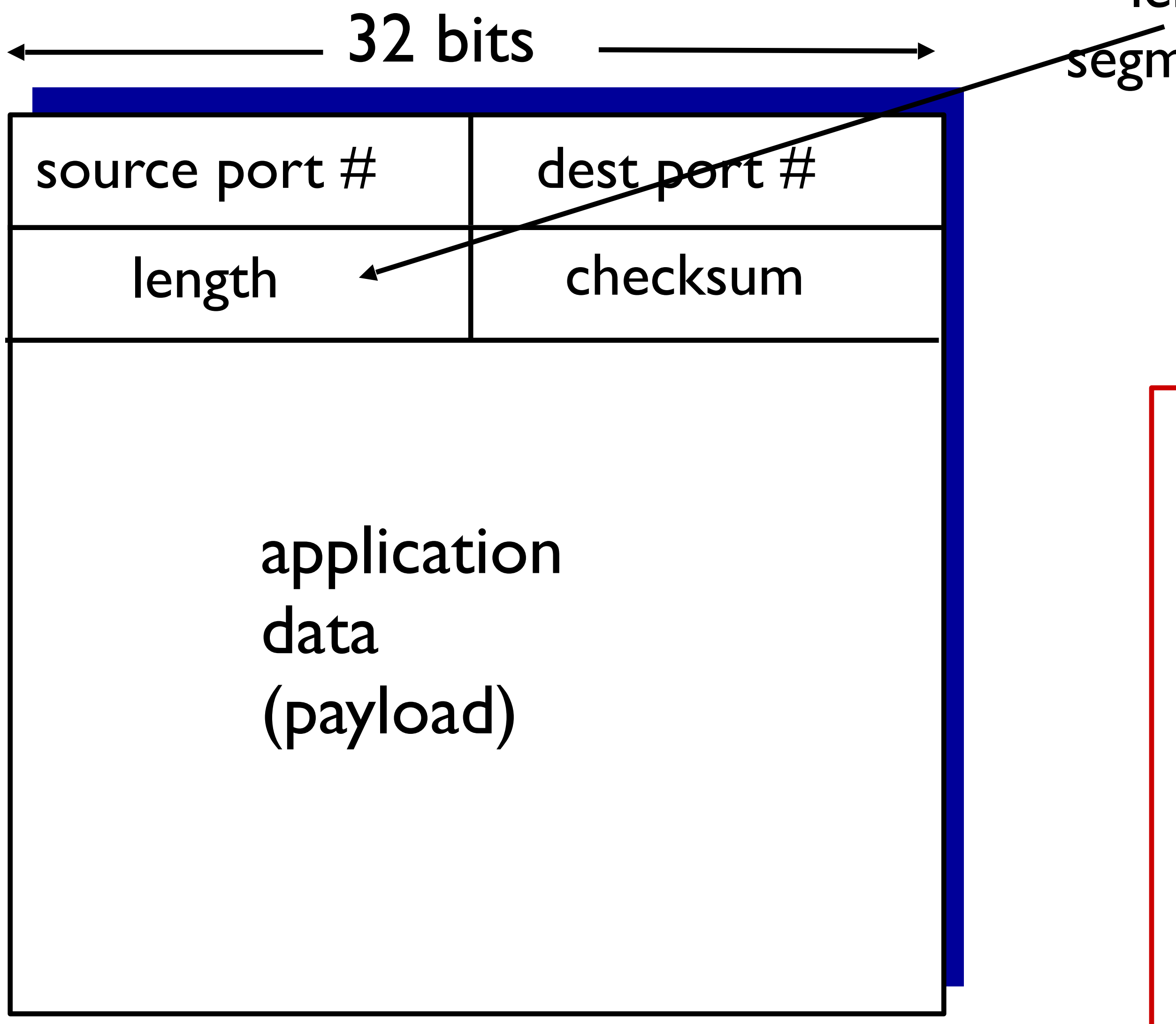


UDP: User Datagram Protocol [RFC 768]

- "no frills," "bare bones" Internet transport protocol
- "best effort" service, UDP segments may be:
  - lost
  - delivered out-of-order to app
- *connectionless:*
  - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others

- UDP use:
  - streaming multimedia apps (loss tolerant, rate sensitive)
  - DNS
  - SNMP
- reliable transfer over UDP:
  - add reliability at application layer
  - application-specific error recovery!

UDP: segment header



length, in bytes of UDP segment, including header

UDP segment format

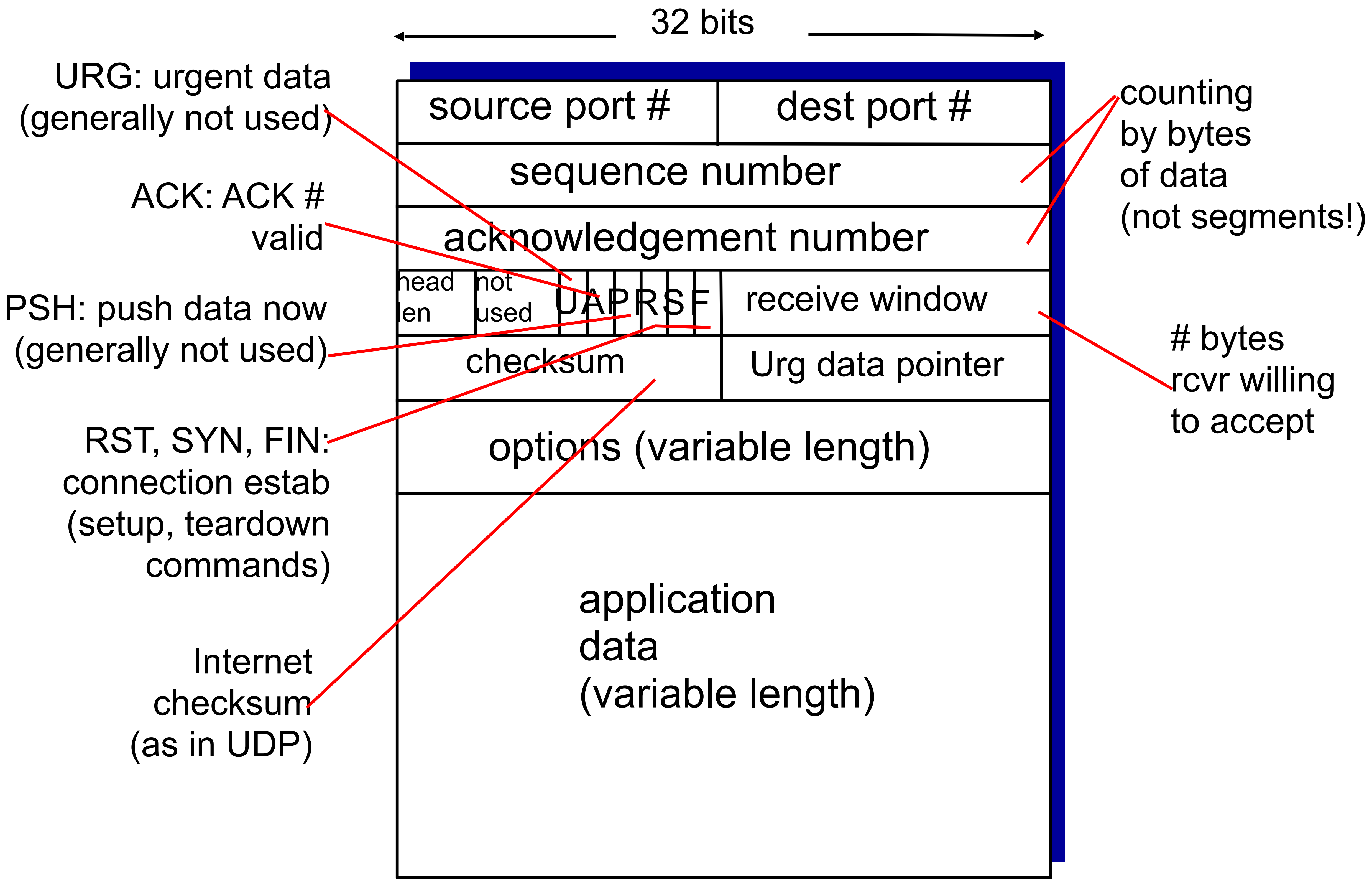
why is there a UDP?

- no connection establishment (which can add delay)
- simple: no connection state at sender, receiver
- small header size
- no congestion control: UDP can blast away as fast as desired

TCP: Overview RFCs: 793,1122,1323, 2018, 2581

- point-to-point:
  - one sender, one receiver
- reliable, in-order *byte steam:*
  - no "message boundaries"
- pipelined:
  - TCP congestion and flow control set window size
- full duplex data:
  - bi-directional data flow in same connection
  - MSS: maximum segment size
- connection-oriented:
  - handshaking (exchange of control msgs) inits sender, receiver state before data exchange
- flow controlled:
  - sender will not overwhelm receiver

TCP segment structure



32 bits

source port #

dest port #

sequence number

acknowledgement number

head len

not used

U A P R S F

receive window

checksum

Urg data pointer

options (variable length)

application
data
(variable length)

URG: urgent data (generally not used)

ACK: ACK # valid

PSH: push data now (generally not used)

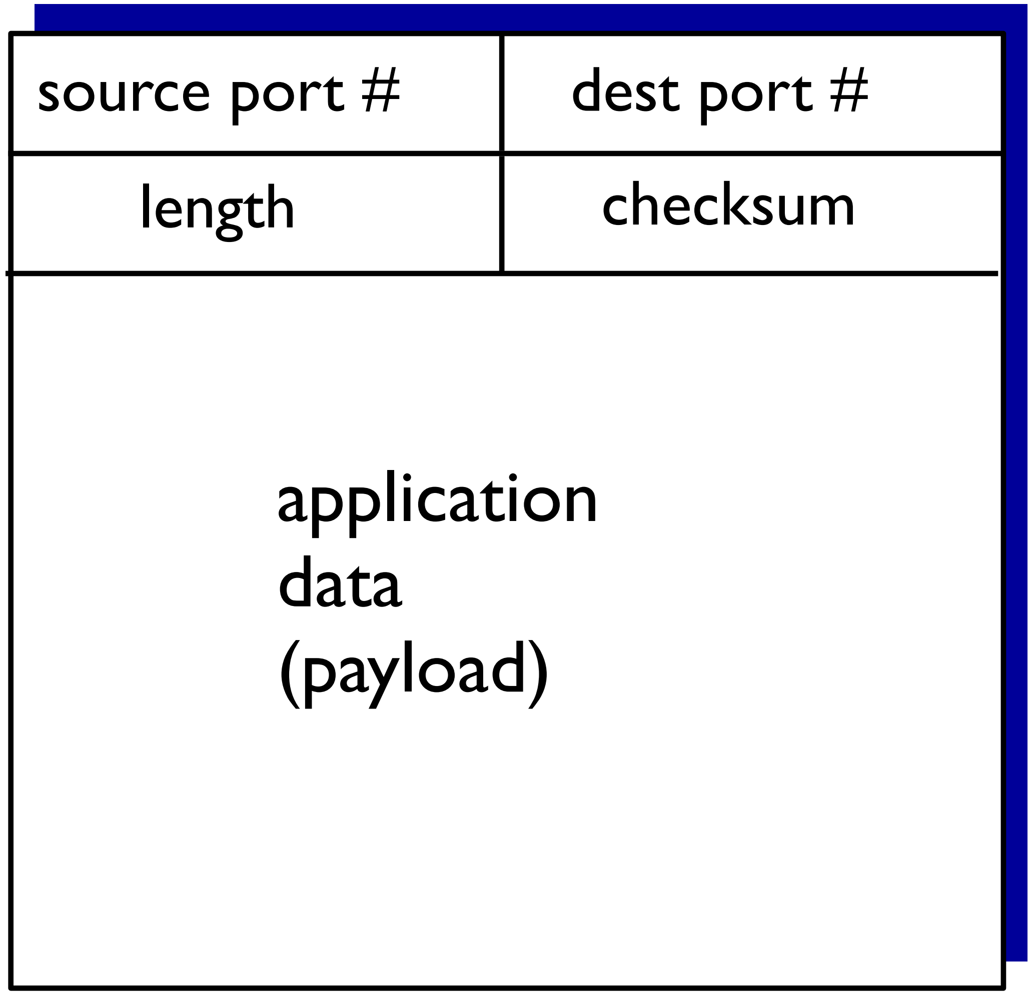
RST, SYN, FIN: connection estab (setup, teardown commands)

Internet checksum (as in UDP)

counting by bytes of data (not segments!)

# bytes rcvr willing to accept

TCP vs UDP



source port #

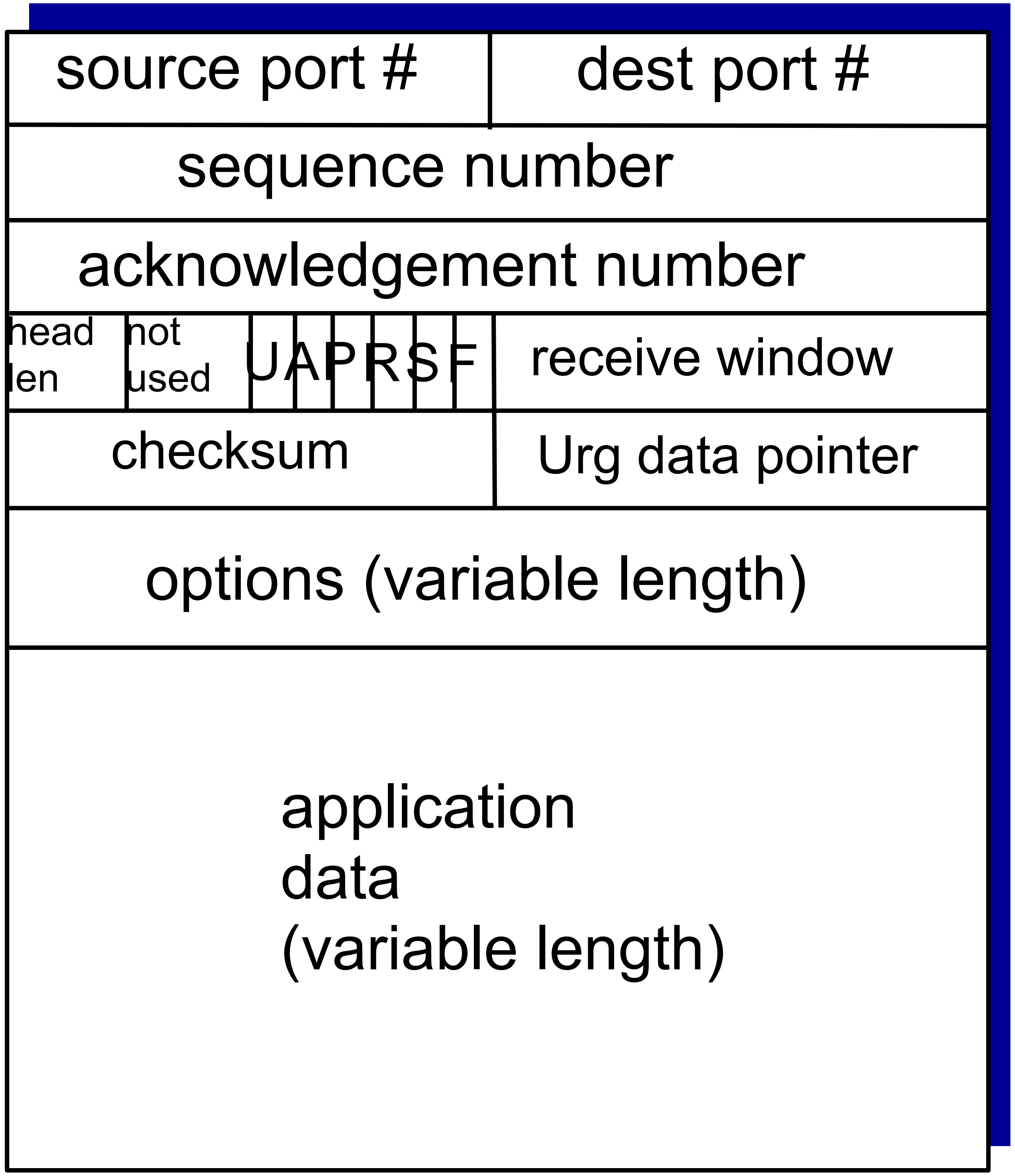
dest port #

length

checksum

application data (payload)

UDP format



source port #

dest port #

sequence number

acknowledgement number

head len

not used

U A P R S F

receive window

checksum

Urg data pointer

options (variable length)

application data (variable length)

TCP format

Network Programming

Server

- Creates a **socket**
- **Binds** socket
- **Listens** for new connections
- Loop forever:
  - **Accepts** client as new temp socket
  - **Receives** client requests
  - **Sends** response
  - …
  - **Closes** temp socket

Client

- Creates a **socket**
- **Connects** to server
- **Sends** request
- **Receives** response
- …
- **Closes** socket

Socket is created as UDP or TCP — hides all those details from application!
(UDP could skip connect steps)

Checkout the code!

Let's look at (and write) some real code

Go to: https://github.com/gwAdvNet2015/

- go to the **adv-net-samples** repository

Fork the repository to your account (top right button)

Clone **your fork** onto your laptop or koding.com

- **git clone https://github.com/USER/adv-net-samples.git**

Setup **upstream** repository

- **git remote add upstream** https://github.com/gwAdvNet2015/adv-net-samples.git