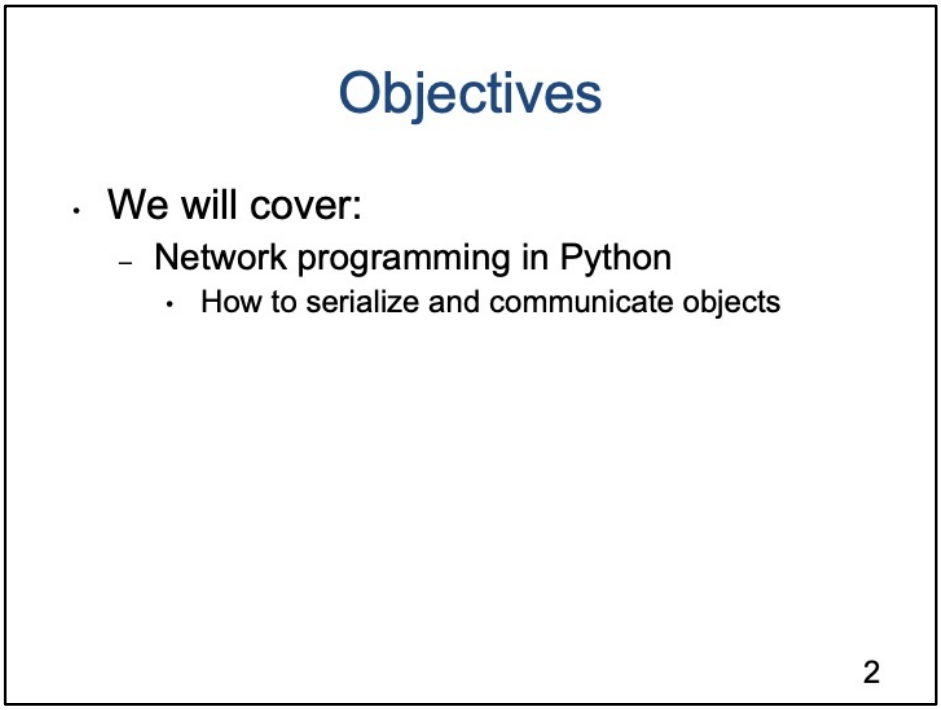
Network Programming (Part 2)

Robert M. Dondero, Ph.D.
Princeton University



Objectives

Continued from last lecture…

[see slide]

Agenda

- **Communicating objects**
- Communicating shared objects

Communicating Objects

- Running example…

- **Course application**
  - The job
    - Server writes data for two courses to socket
    - Client reads data for two courses from socket
    - Client writes data to stdout
      - To illustrate that the comm worked

Communicating Objects

- **Question:**
  - How to communicate objects between client and server?
- **Answer 1:**
  - Convert objects to char sequences; send char sequences; convert char sequences back to objects

Communicating Objects

- See **courses1** app

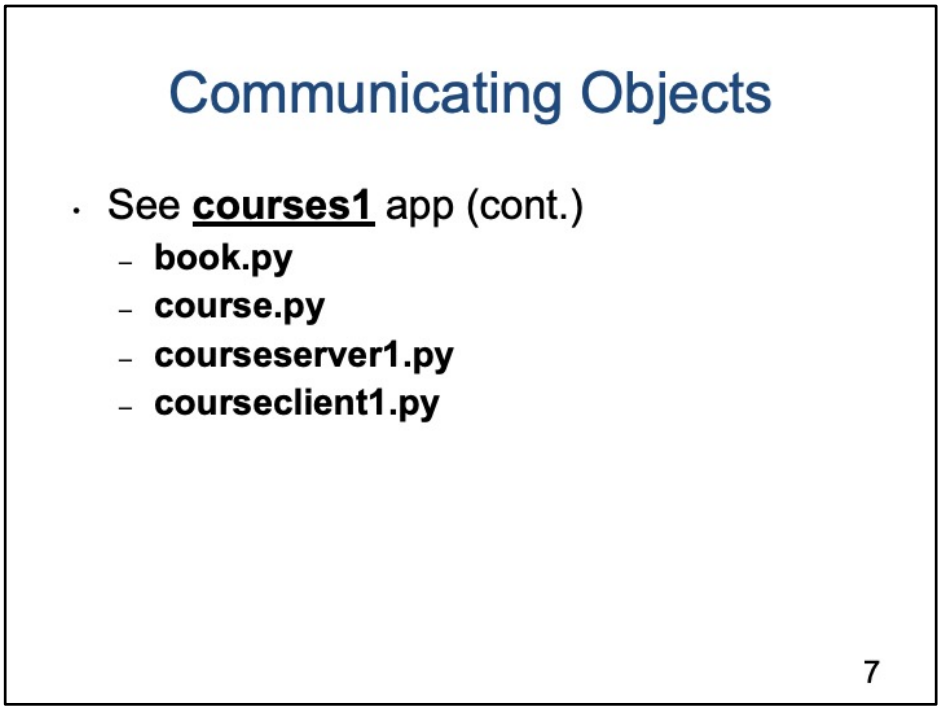
Server: On host 192.168.1.8

```
$ python courseserver1.py 55555
Opened server socket
Bound server socket to port
Listening
```

Client

```
$ python courseclient1.py 192.168.1.8 55555
COS 217
C Programming, 88.55

COS 333
The Practice of Programming, 35.14

$
```

```
$ python courseserver1.py 55555
Opened server socket
Bound server socket to port
Listening
Accepted connection
Opened socket
Server IP addr and port: ('192.168.1.8',
55555)
Client IP addr and port: ('192.168.1.8',
50958)
Wrote courses to client
```



Communicating Objects

See **courses1** app (cont.)

Code notes: book.py
Define a Book class
Code notes: course.py
Define a Course class
Each Course object has a Book object
Code notes: courseserver1.py
Convert `Course` object to UTF-8; write to client; repeat 2 times
Code notes: courseclient1.py
Read UTF-8 from server; convert to `Course` object; repeat 2 count times

Communicating Objects

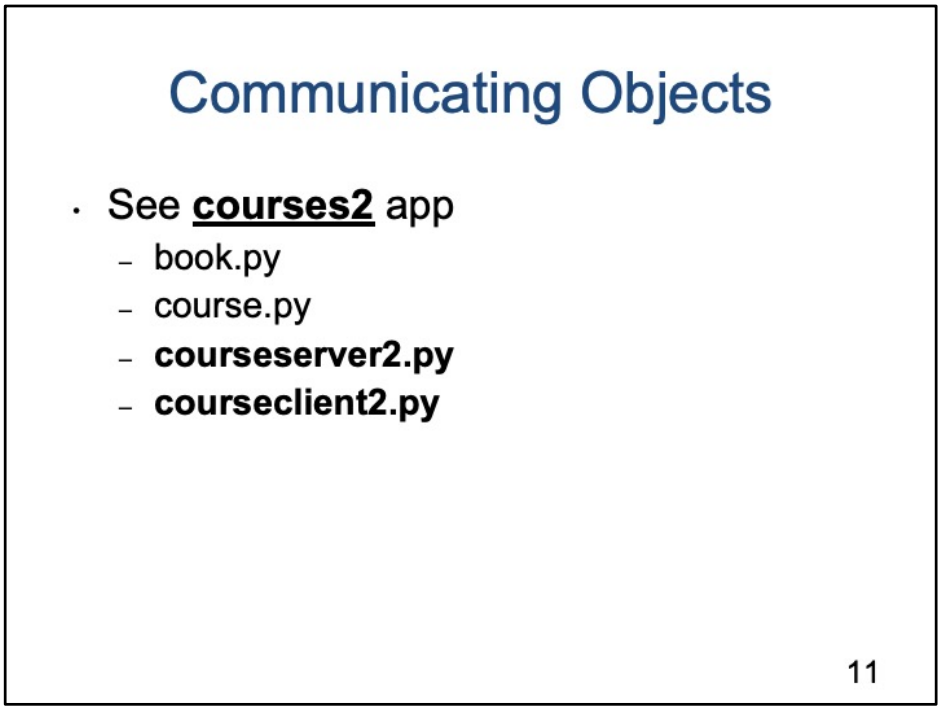
- Fine
- But is there an easier way?

Communicating Objects

- **Question:**
  - How to communicate objects between client and server?
- **Answer 1:**
  - Convert objects to char sequences; send char sequences; convert char sequences back to objects
- **Answer 2:**
  - Communicate “pickled” objects

Communicating Objects

- `pickle.dump(obj, flo)`
  - Serializes `obj` to byte stream
  - Writes byte stream to `flo`
- `obj = pickle.load(flo)`
  - Reads byte stream from `flo`
  - Deserializes byte stream to form `obj`



Communicating Objects

[see slide]

Code notes: courseserver2.py

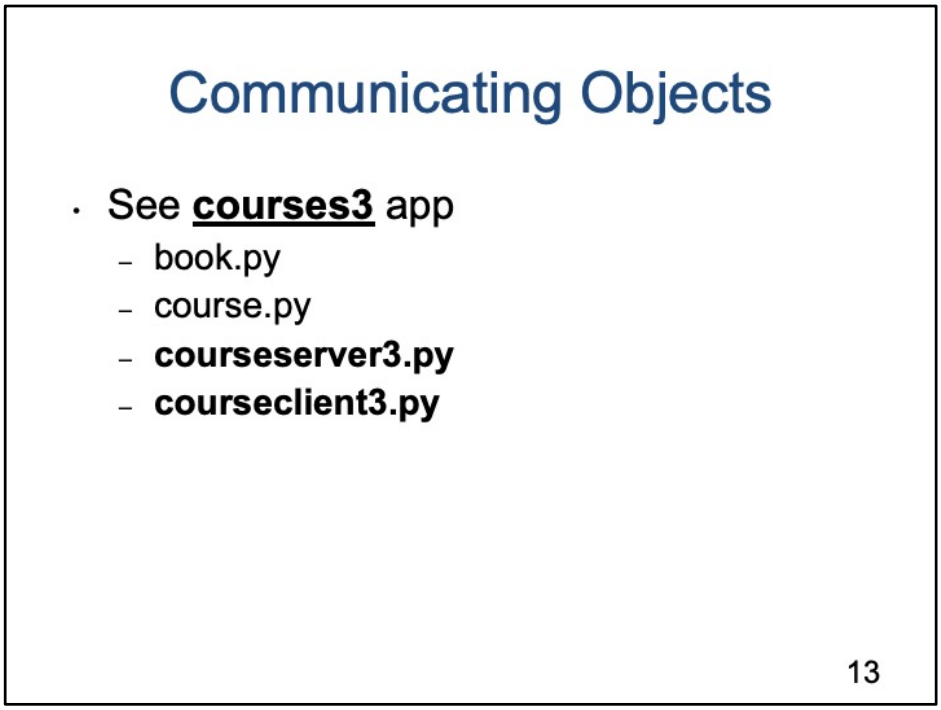
Pickle `Course` object's `str` objects & `float` object to create byte sequences, write byte sequences, repeat 2 times

Code notes: courseclient2.py

Read byte sequences, unpickle to create `str` objects & `float` object; create `Course` object; repeat 2 times

Communicating Objects

- But wait!
- `Book` objects and `Course` objects are objects and so can be “pickled”...



Communicating Objects

[see slide]

Code notes: courseserver3.py

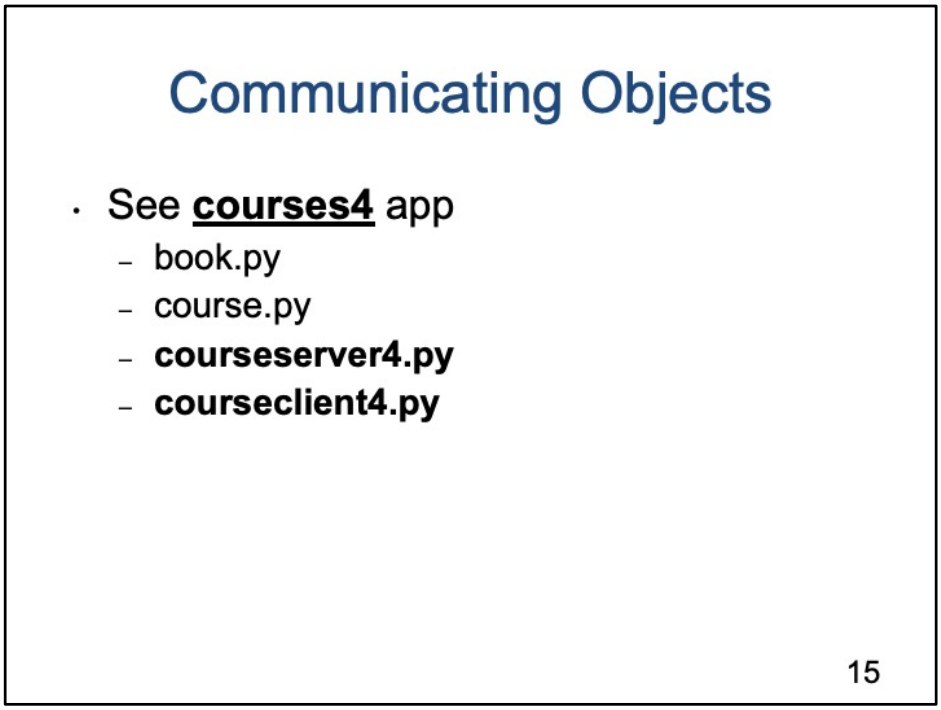
Pickle `Course` object to create byte sequence, write byte sequence, repeat 2 times

Code notes: courseclient3.py

Read byte sequence, unpickle to create `Course` object; repeat 2 times

Communicating Objects

- But wait!
- `list` objects are objects, and so can be "pickled"



Communicating Objects

[see slide]

Code notes: courseserver4.py

Pickle `list` object to create byte sequence, write byte sequence!

Code notes: courseclient4.py

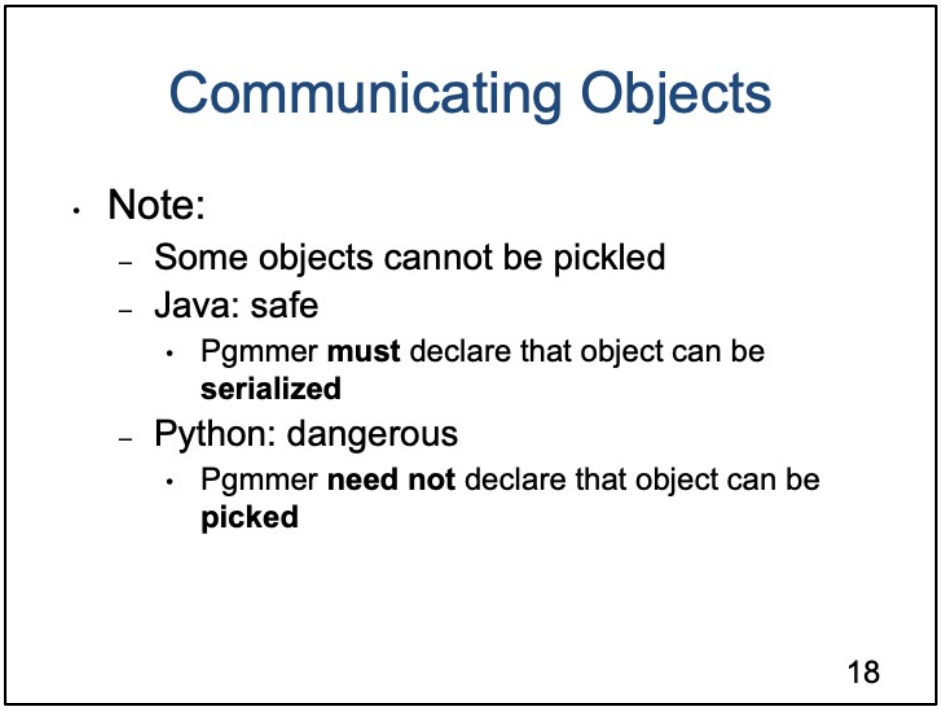
Read byte sequence, unpickle to create `list` object!

Communicating Objects

- Note:
  - courseserver1.py works with:
    - courseclient1.py
    - Telnet!
    - A course client written in Java, C, …
  - UTF-8 is language-independent

Communicating Objects

- Note:
  - courseserver2.py works only with a course client written in Python
  - courseserver3.py works only with a course client written in Python
  - courseserver4.py works only with a course client written in Python
  - Object pickling is specific to Python



Communicating Objects

Note:

Some objects cannot be pickled
Example: An object that has a field which references a file

Java: safe

Programmer must declare that object can be serialized

Python: dangerous

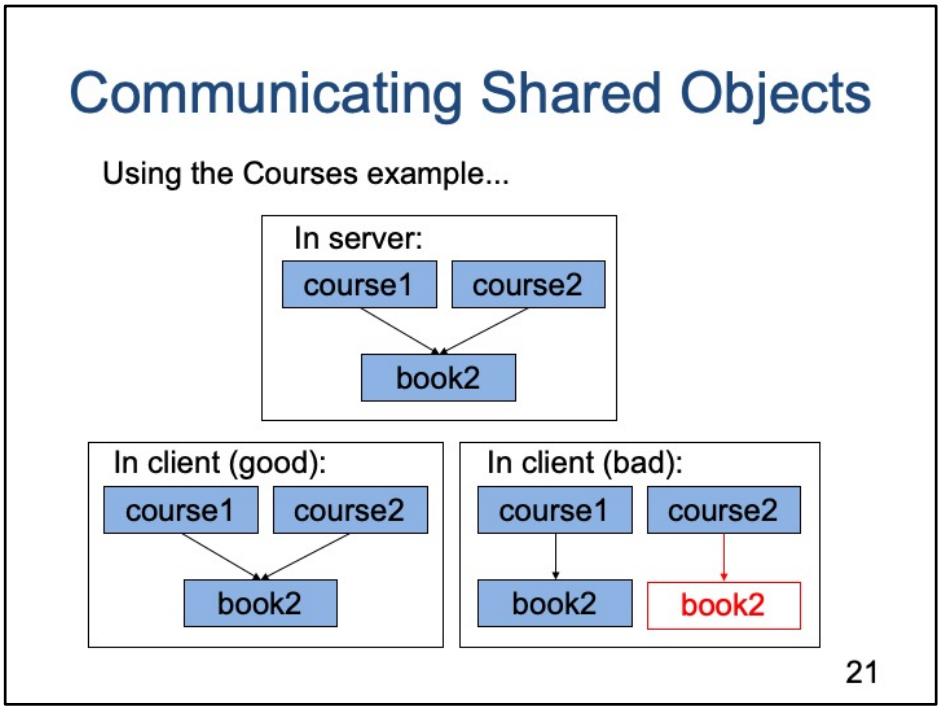
Programmer need not declare that object can be picked

Agenda

- Communicating objects
- **Communicating shared objects**

Communicating Shared Objects

- Problem:
  - What if multiple objects are composed of the same nested objects...



Communicating Shared Objects

[see slide]

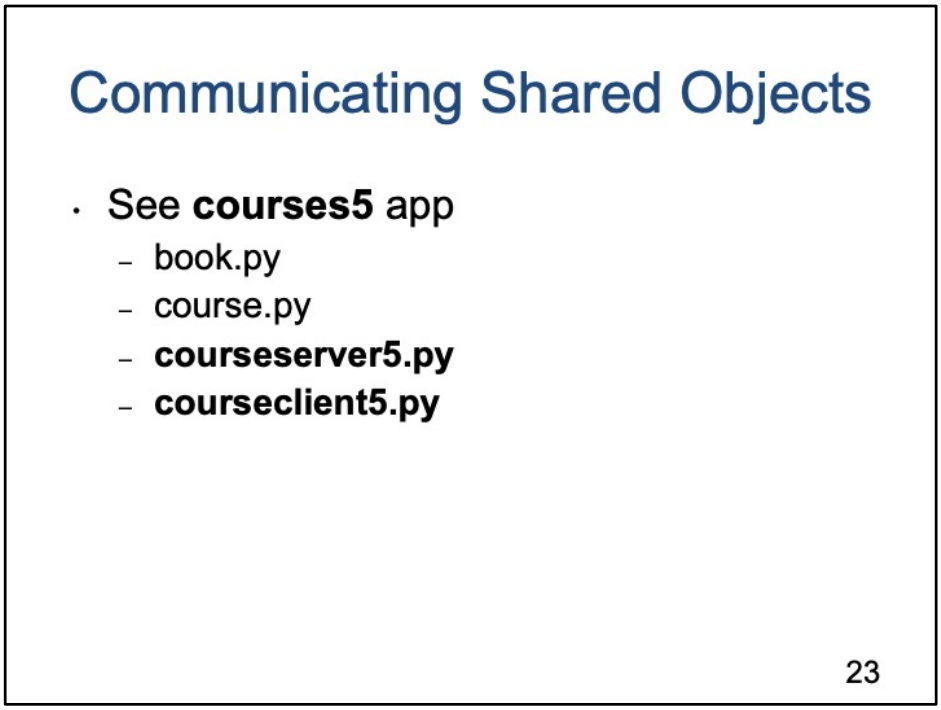
In the server, suppose the course1 object and the course2 object reference the same book2 object

After data comm…

In the client, we want the course1 object and the course2 object reference the same book2 object
If we're not careful, the course1 object and the course2 object might reference different book objects

Communicating Shared Objects

- Solution:
  - Handled automatically by pickle module…
  - **Iff the data structure is dumped/loaded as a whole**



Communicating Shared Objects

[see slide]

Code notes: courseserver5.py

Write two `Course` objects which share the same `Book` object
Write them via distinct calls of `pickle.dump()`

Code notes: courseclient5.py

Read two `Course` objects which should share the same `Book` object
Read them via distinct calls of `pickle.load()`

Communicating Shared Objects

- See **courses5** app

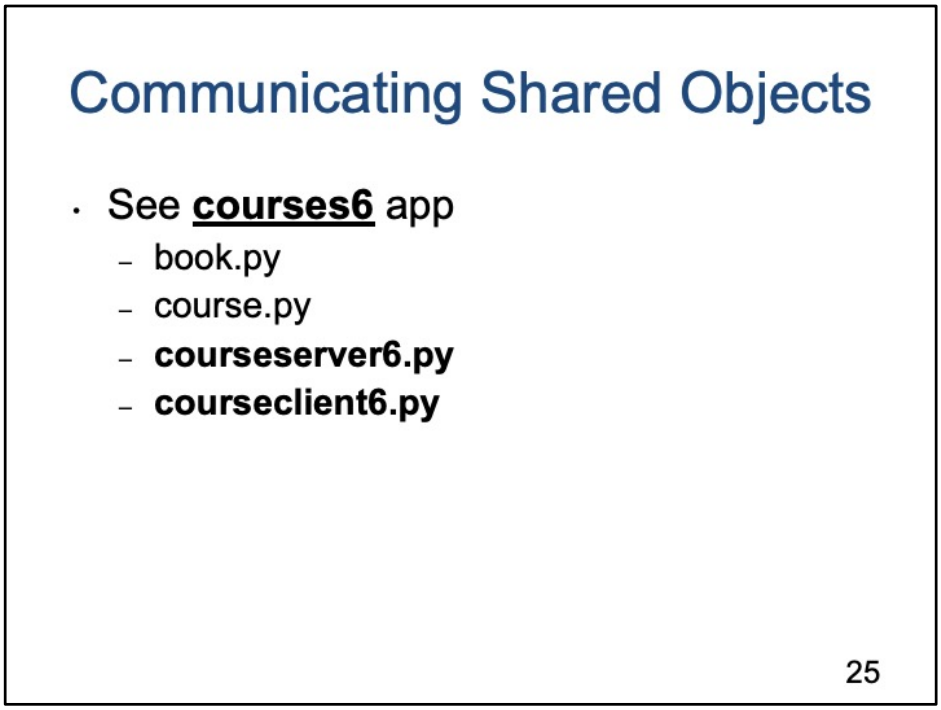
Server: On host 192.168.1.8

```
$ python courseserver5.py 55555
Opened server socket
Bound server socket to port
Listening
```

```
$ python courseclient5.py 192.168.1.8 55555
COS 217
The Practice of Programming, 35.14

COS 333
The Practice of Programming, 35.14

COS 217
The Practice of Programming, 99.99

COS 333
The Practice of Programming, 35.14

$
```

```
$ python courseserver5.py 55555
Opened server socket
Bound server socket to port
Listening
Accepted connection
Opened socket
Server IP addr and port: ('192.168.1.8',
55555)
Client IP addr and port: ('192.168.1.8',
50959)
Wrote courses to client
```

Change of `Book` price in client confirms that shared object **was not** written/read properly



Communicating Shared Objects

[see slide]

Code notes: courseserver6.py

Write one list object containing two Course objects which share the same Book object

Note: one call of pickle.dump()

Code notes: courseclient6.py

Read one list objectc containing two Course objects which share the same Book object

Note: one call of pickle.load()

Communicating Shared Objects

- See **courses6** app

Server: On host 192.168.1.8

```
$ python courseserver6.py 55555
Opened server socket
Bound server socket to port
Listening
```

```
$ python courseclient6.py 192.168.1.8 55555
COS 217
The Practice of Programming, 35.14

COS 333
The Practice of Programming, 35.14

COS 217
The Practice of Programming, 99.99

COS 333
The Practice of Programming, 99.99

$
```

```
$ python courseserver6.py 55555
Opened server socket
Bound server socket to port
Listening
Accepted connection
Opened socket
Server IP addr and port: ('192.168.1.8',
55555)
Client IP addr and port: ('192.168.1.8',
50960)
Wrote courses to client
```

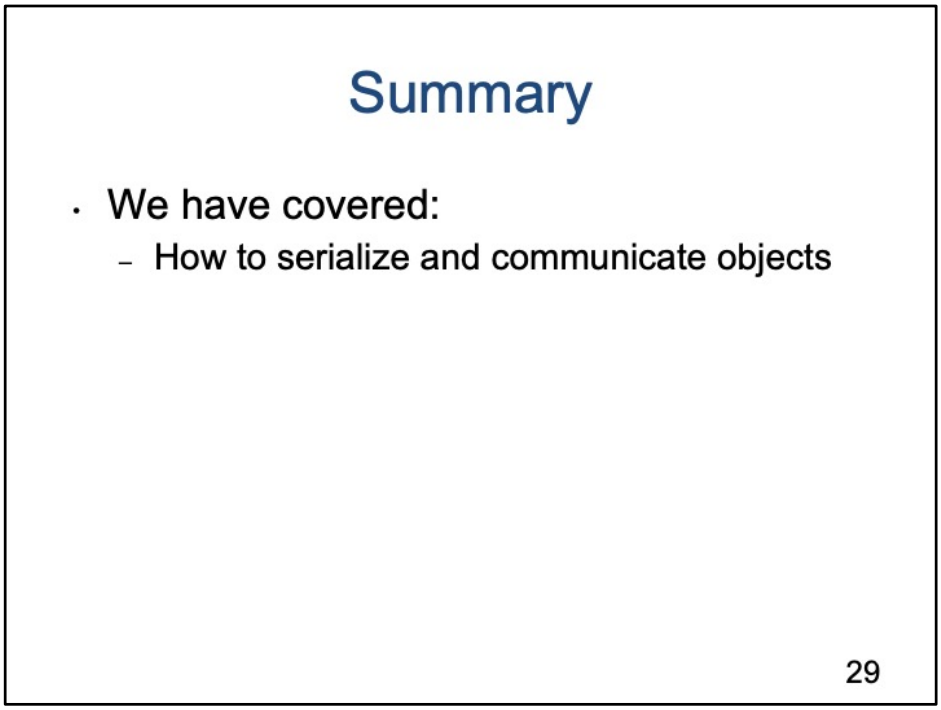
Change of `Book` price in client confirms that shared object **was** written/read properly

Communicating Shared Objects

- How does it work?
  - `pickle.dump()` maps each object address to a unique serial num
    - Normally: writes serial num and object
    - Sees same object address again => writes serial number only
  - `pickle.load()` maps each serial number to a unique object address
    - Normally: creates object for each serial num that it reads
    - Sees same serial num again => creates ref to existing object

Communicating Shared Objects

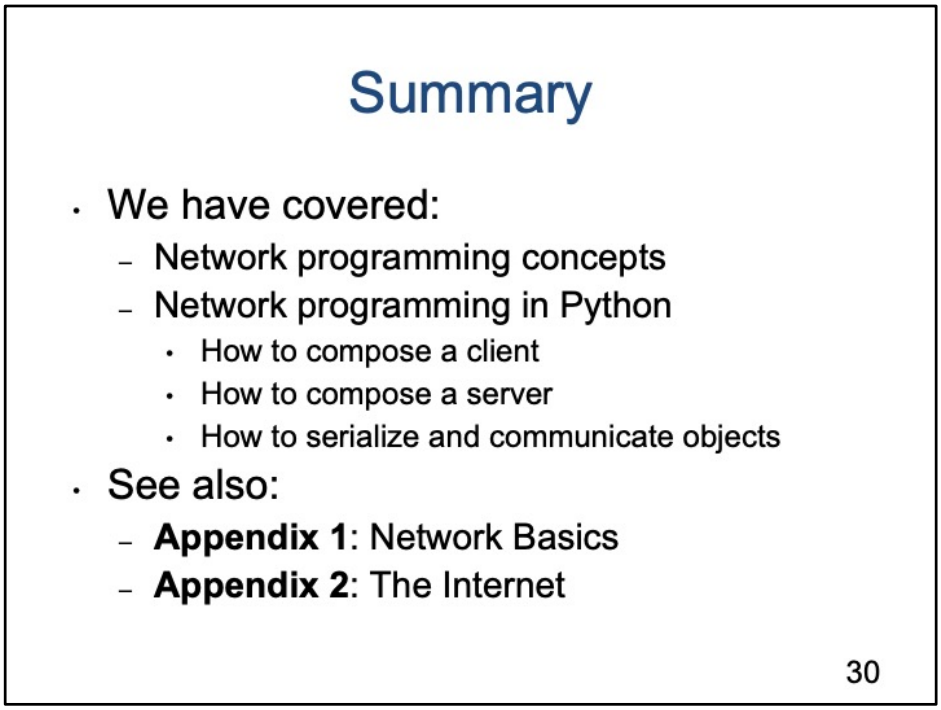
- For more information:
  - *Core Java: Volume II - Advanced Features* by Cay Horstmann, Chapter 2



Summary

Of this lecture…

[see slide]



Summary

Of the Network Programming sequence of lectures…

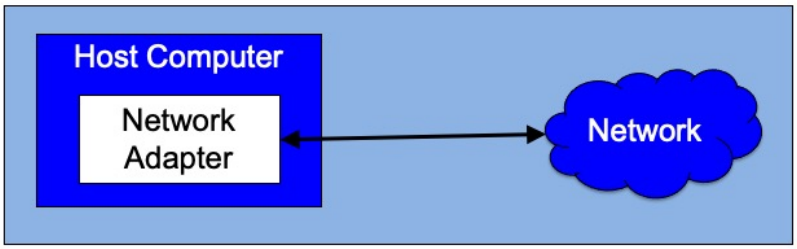
[see slide]

Appendix 1:
Network Basics

Network Basics

- Networking
  - Huge topic
  - Can only scratch the surface
- Goal
  - Convey workable mental model
  - … From programmer's point of view
- Approach
  - Bottom-up...

Network Basics



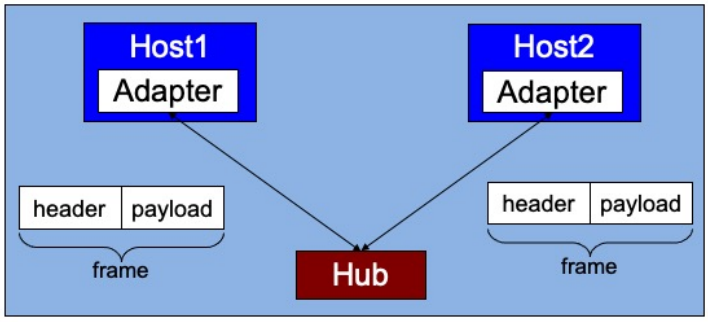
Network adapter:
- HW that connects host computer to network
- Each has unique "medium access control (MAC) address"

MAC address:
- xx:xx:xx:yy:yy:yy, where each x and y is a hexadecimal digit
- xx:xx:xx is unique to vendor
- yy:yy:yy is unique to device

Network Basics

- Try on courselab:
  - `ifconfig`
    - Note “ether”

Network Basics

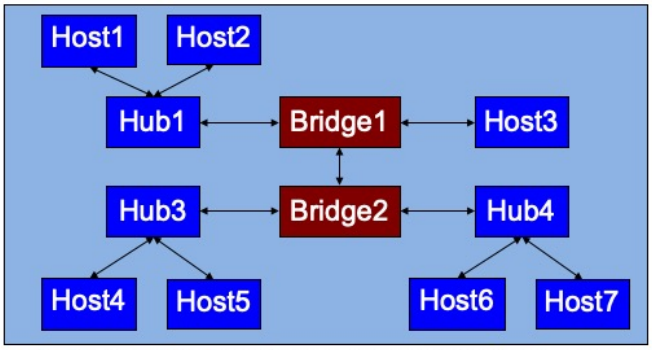


Frame = header + payload
- **Header** identifies source and destination addresses
- **Payload** contains user data

Hub (alias **repeater**) = HW (no SW) that transmits frames among hosts
- Receives frame from adapter; doesn't examine header; copies to all others
- Every adapter **sees** frame; only destination adapter **reads** it

Network Segment = hosts + hub
- Scope: one room; one floor of a building

Network Basics

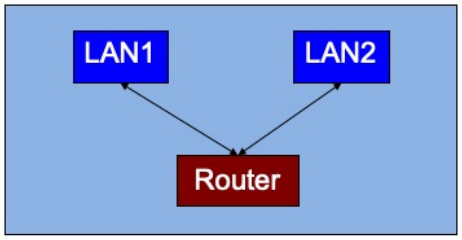


LAN

Bridge (alias **switch**) = HW+SW that connects hosts, hubs, other bridges
- Does examine headers
- Analyzes message sources; **learns** where hosts are
  - host1 → host2: hub1 → bridge1 → (discard)
  - host1 → host7: hub1 → bridge1 → bridge2 → hub4

Local Area Network (LAN) = hosts + hubs + bridges
- Scope: one building; one campus

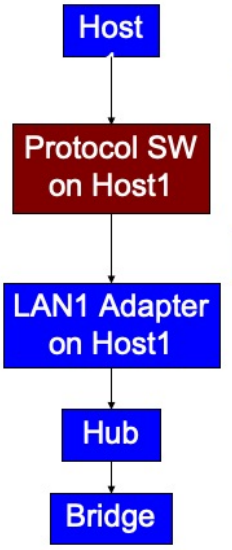
Network Basics



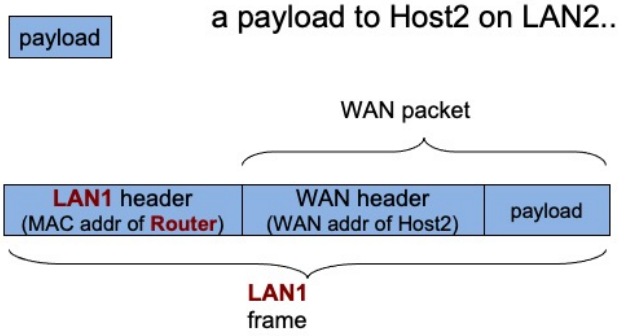
WA
N

Router (~alias **gateway**) = HW that connects multiple (incompatible) LANs
Wide Area Network (WAN) = LANs + routers
Scope: planet Earth! And beyond!
Each host has both a LAN (MAC) address and a **WAN address**
LAN address: not related to network topology
WAN address: related to network topology

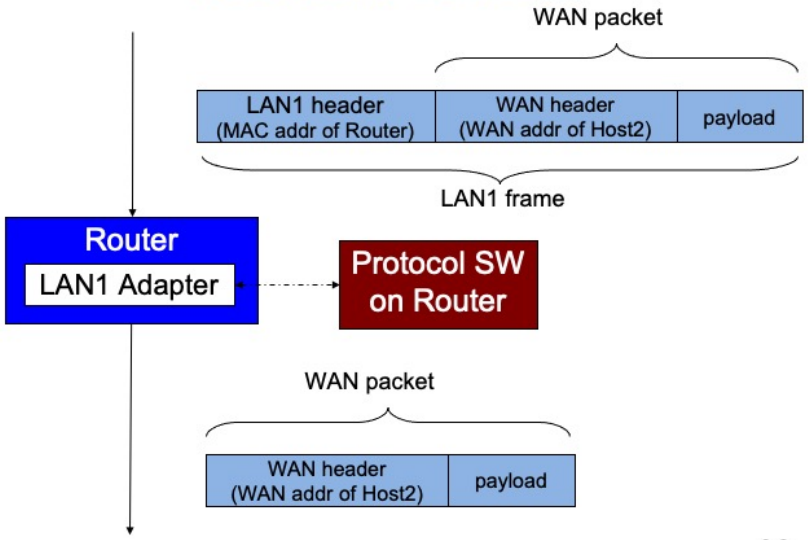
Network Basics



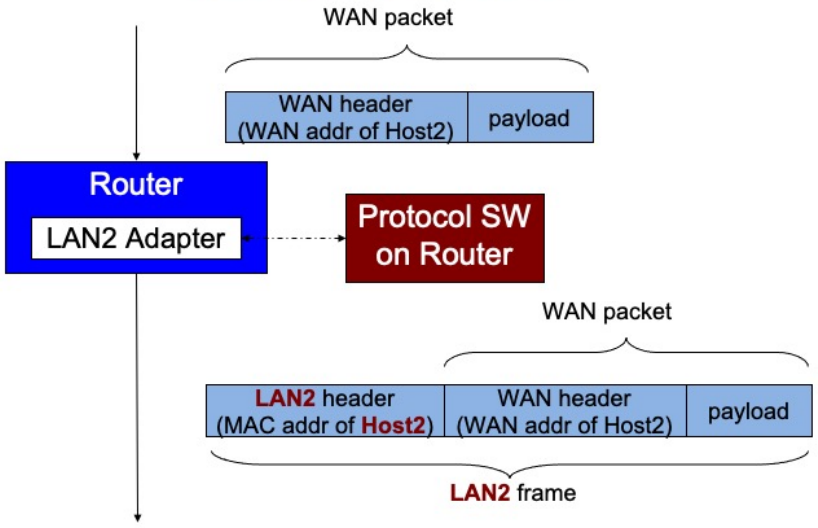
When Host1 on LAN1 sends a payload to Host2 on LAN2...



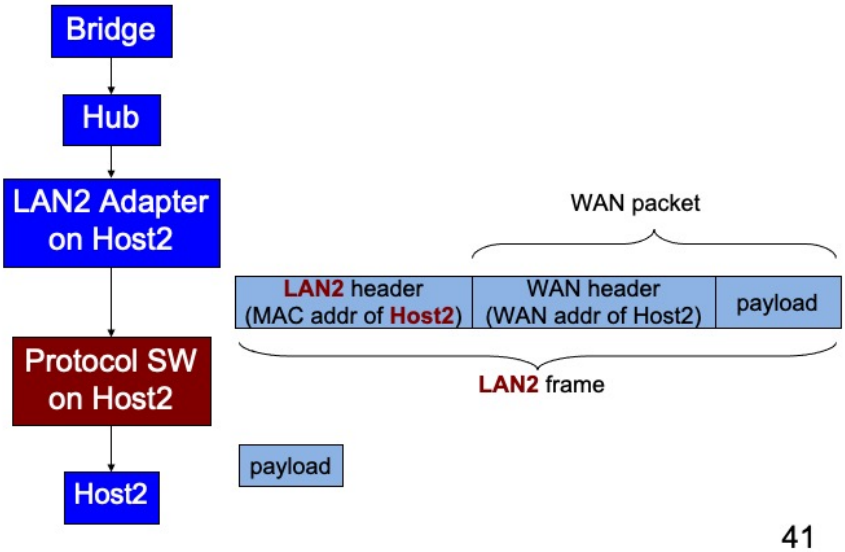
Network Basics



Network Basics

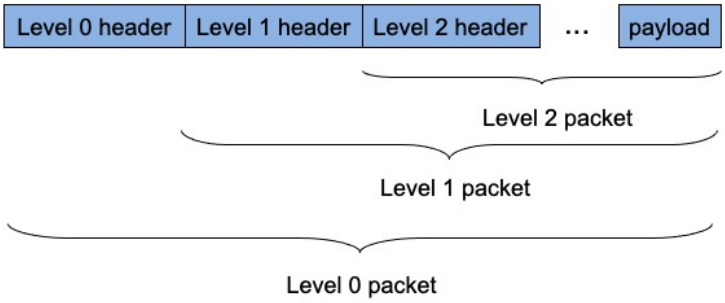


Network Basics



Network Basics

Generic Packet Structure



Protocols can be (and typically are) **layered/stacked**
And so packets can be (and typically are) **layered/stacked**

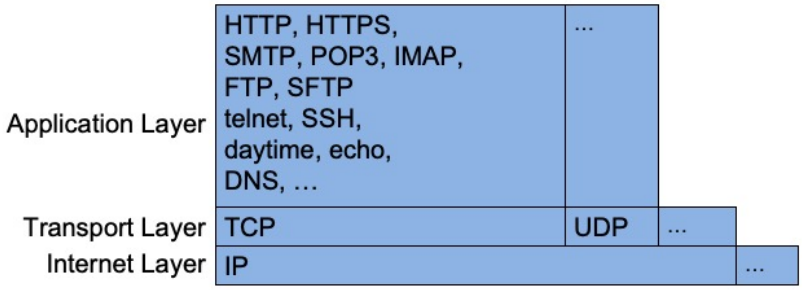
Appendix 2:
The Internet

The Internet

- The Internet
  - A WAN that uses a particular protocol stack

The Internet

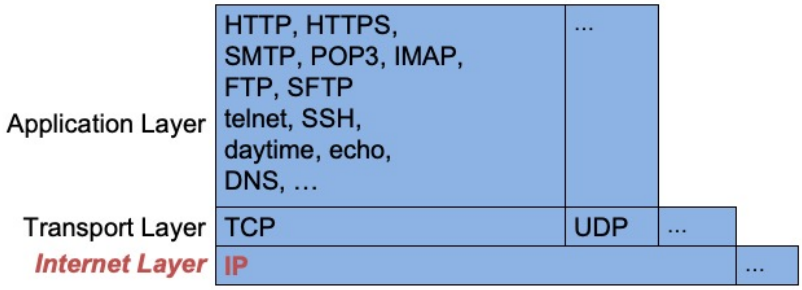
The Internet Protocol Stack



Typical Internet Packet Structure

The Internet

The Internet Protocol Stack



Dominant protocol: **IP (Internet Protocol)**…

The Internet

- IP characteristics
  - Connectionless
    - No persistent connection
  - Packetized
    - Sender splits long message into packets
    - Receiver re-assembles the packets
  - Unreliable
    - Packets can be lost (errors, congestion)
    - Receiver not notified of lost packets

The Internet

- IP Header
  - Source WAN address
  - Destination WAN address
  - …

The Internet

- IP addresses
  - WAN address = ***IP address***
  - Internal form: 32 bits
  - Human-readable form: “dotted decimal”
    - xxx.xxx.xxx.xxx

The Internet

Some IP Addresses:

IP Address	Computer
128.112.155.159 128.112.155.160	Princeton courselab computers
129.6.15.28	A US gov computer
127.0.0.1	The local host

The Internet

- Try on courselab: `ifconfig`
  - Note “inet”
  - Note: 127.0.0.1 is IP address of “the local host”
    - Useful for testing networking apps

The Internet

- **Problem**: Difficult for humans to remember/use "dotted decimal" IP addresses
- **Solution**: Domain names
- ***Domain name***: A symbolic name for IP address(es)
  - Example: courselab.cs.princeton.edu
  - Example: courselab03.cs.princeton.edu
  - Example: time-a.nist.gov
  - Example: localhost

The Internet

- **Problem**: How to map domain name to IP address(es)?
- **Early Solution**: hosts.txt file at SRI International
  - Downloaded ~weekly
  - Didn't scale
- **Current Solution**: ***Domain Name System***

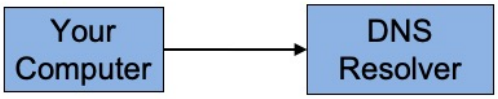
The Internet

- ***Domain name system (DNS)***
  - The “phone book” of the Internet
  - Maps domain names to their IP addresses
  - Distributed hierarchical database

The Internet

The Domain Name System

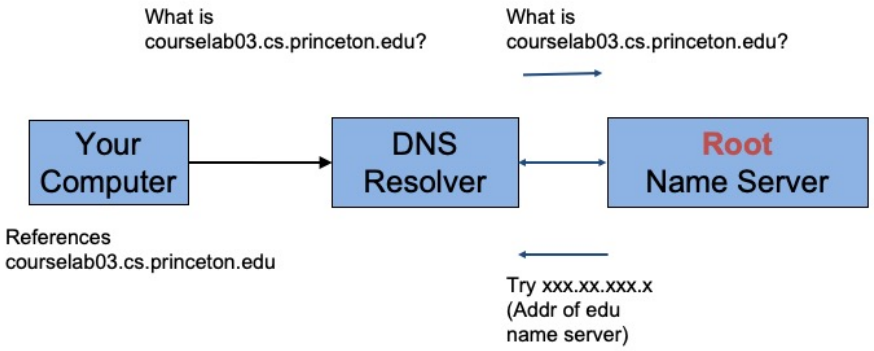
What is
courselab03.cs.princeton.edu?



References
courselab03.cs.princeton.edu

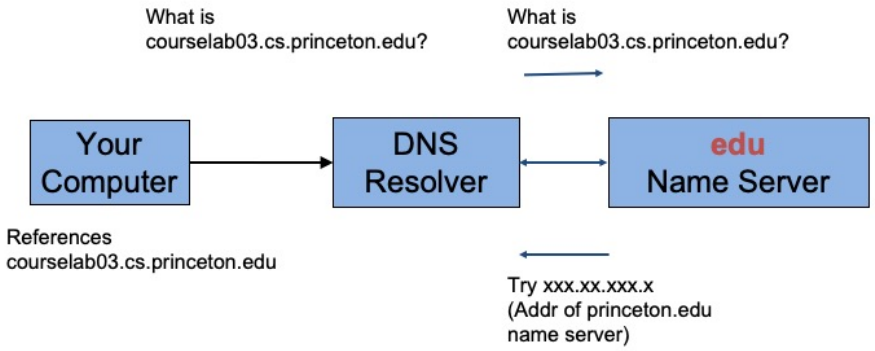
The Internet

The Domain Name System



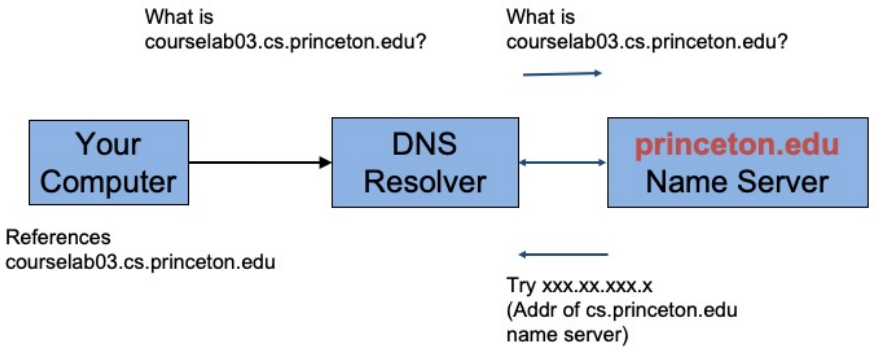
The Internet

The Domain Name System



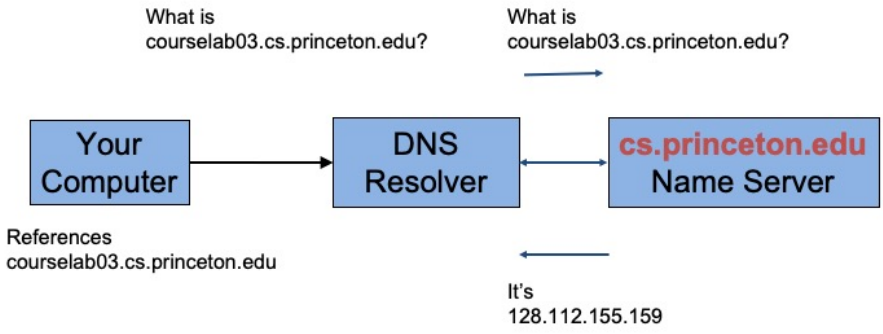
The Internet

The Domain Name System



The Internet

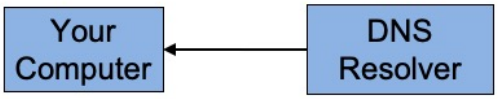
The Domain Name System



The Internet

The Domain Name System

It's
128.112.155.159



References
courselab03.cs.princeton.edu

The Internet

DNS root servers:

Many hundreds; over 130 physical locations

The Internet

- **Question**: How can DNS root servers handle the heavy workload?
- **Answer**: Caching at each level of the DNS hierarchy

- In reality, root servers handle mostly silly requests

The Internet

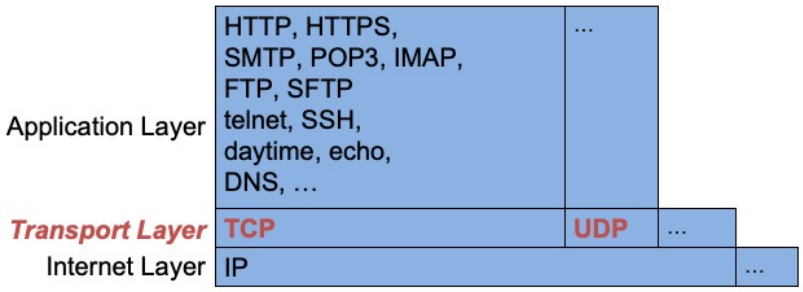
- Try (on Linux, Mac, MS Windows):
  - `nslookup courselab.cs.princeton.edu`
  - `nslookup courselab03.cs.princeton.edu`
  - `nslookup courselab04.cs.princeton.edu`
  - `nslookup time-a.nist.gov`
  - `nslookup nonexistingdomainname`

The Internet

- See ***ipaddress.py***
  - `python ipaddress.py courselab.cs.princeton.edu`
  - `python ipaddress.py courselab03.cs.princeton.edu`
  - `python ipaddress.py courselab04.cs.princeton.edu`
  - `python ipaddress.py time-a.nist.gov`
  - `python ipaddress.py localhost`
  - `python ipaddress.py nonexistingdomainname`

The Internet

The Internet Protocol Stack



Dominant protocols

- **UDP (User Datagram Protocol)**
- **TCP (Transmission Control Protocol)**

The Internet

- UDP characteristics
  - Connections are not persistent
  - Connections are unreliable
  - Adds ***port*** number to IP protocol
- Why UDP?
  - Fast
  - Not all applications need reliable transmission
    - E.g. streaming audio
    - E.g. DNS
- We will not use UDP in COS 333

The Internet

- TCP characteristics
  - Connections are persistent
    - "Virtual circuit"
  - Connections are reliable
    - Reassembles packets in proper order
    - Requests resend of missing/corrupted packets
  - "Ordered reliable byte stream"
    - Recall file descriptors
- Most Internet apps use TCP
- We will use TCP in COS 333

The Internet

- Header
  - Source ***port***
  - Destination ***port***
  - …

The Internet

- ***Port***
  - A software abstraction
  - 16-bit integer
  - Identifies unique process on specified host
- Client and server communicate via ports
  - ***Known*** port on server
  - ***Ephemeral*** port on client
- Ports allow comm between specific **processes** on specific hosts (as opposed to comm between **hosts**)

The Internet

- Port numbers
  - 0-1023: ***Well-Known*** Server Ports
    - Used by common servers (HTTP, FTP, etc.)
  - 1024-49151: ***Registered*** Server Ports
    - Registered by software vendors to reduce likelihood of conflicts
  - 49152-65535: ***Dynamic/Private*** Ports
    - Available for any purpose

The Internet

Some Well-Known Server Ports

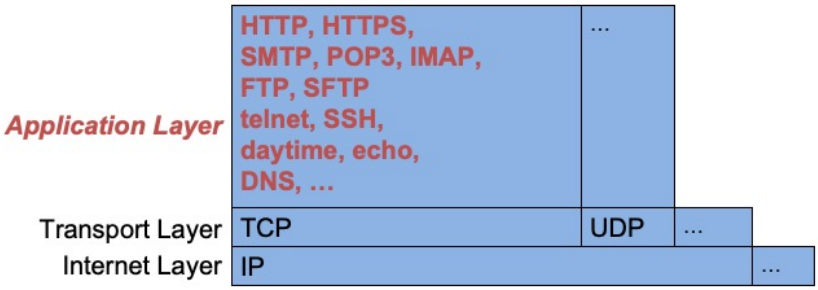
Port	Kind of Process
7	echo
13	daytime
20	FTP-control
21	FTP-data
22	SSH, SFTP
23	telnet
25	SMTP
53	DNS
80	HTTP
110	POP2
143	IMAP
443	HTTPS

Some Registered Server Ports

Port	Kind of Process
3306	MySQL
5432	PostgreSQL
6379	Redis
8080	Apache Tomcat
27017	MongoDB

The Internet

The Internet Protocol Stack



Many protocols, each with its own header/payload format…

The Internet

Some application layer protocols:

Protocol	Internet Application
daytime	Time of Day
echo	Echo
HTTP (Hypertext Transfer Protocol) HTTPS (Hypertext Transfer Protocol Secure)	World Wide Web
SMTP (Simple Mail Transfer Protocol) POP3 (Post Office Protocol 3) IMAP (Internet Message Address Protocol)	E-Mail
FTP (File Transfer Protocol) SFTP (Secure File Transfer Protocol)	File Transfer
Telnet SSH (Secure Shell)	Remote Shell
DNS (Domain Name System)	DNS

The Internet

The Internet Protocol Hourglass