Network Programming (Part 2)

Robert M. Dondero, Ph.D.
Princeton University

Objectives

- We will cover:
  - Communicating char sequences
  - Communicating JSON docs
  - Communicating pickled Python objects

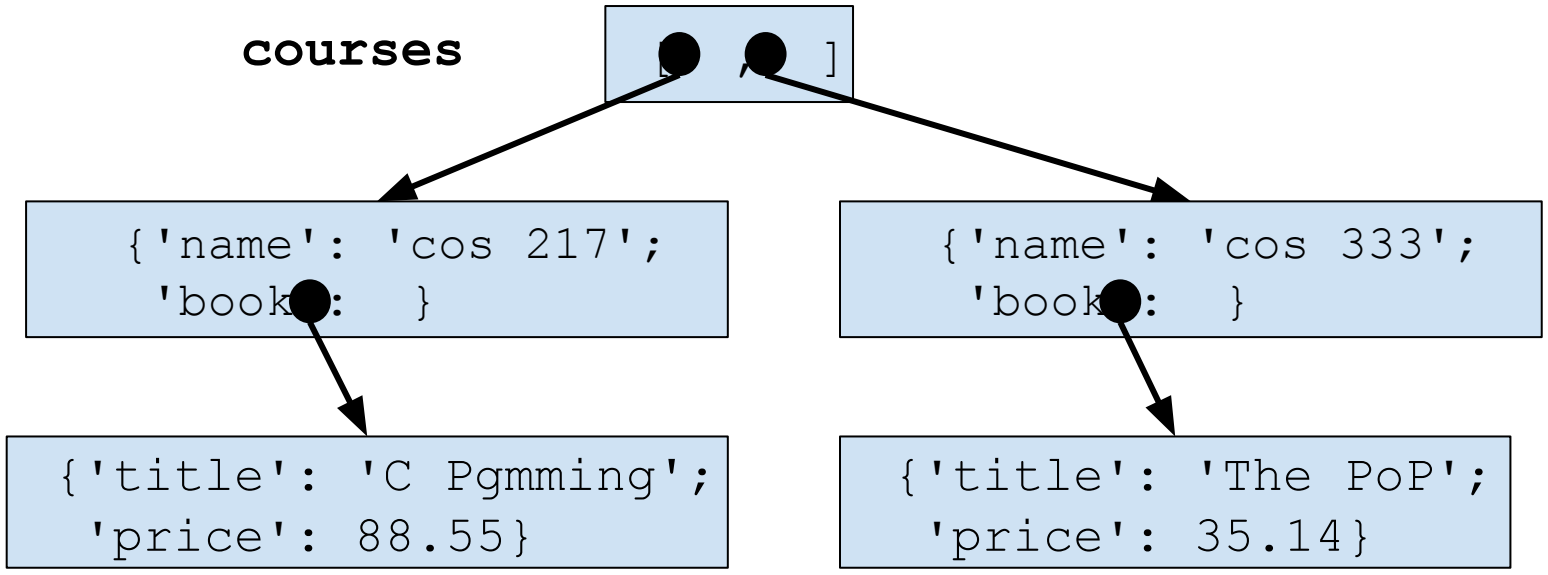
Running Example

Example: Courses and Books

```
book0 = {'title': 'C Programming', 'price': 88.55}
book1 = {'title': 'The Practice of Programming', 'price': 35.14}
course0 = {'name': 'COS 217', 'book': book0}
course1 = {'name': 'COS 333', 'book': book1}
courses = [course0, course1]
```

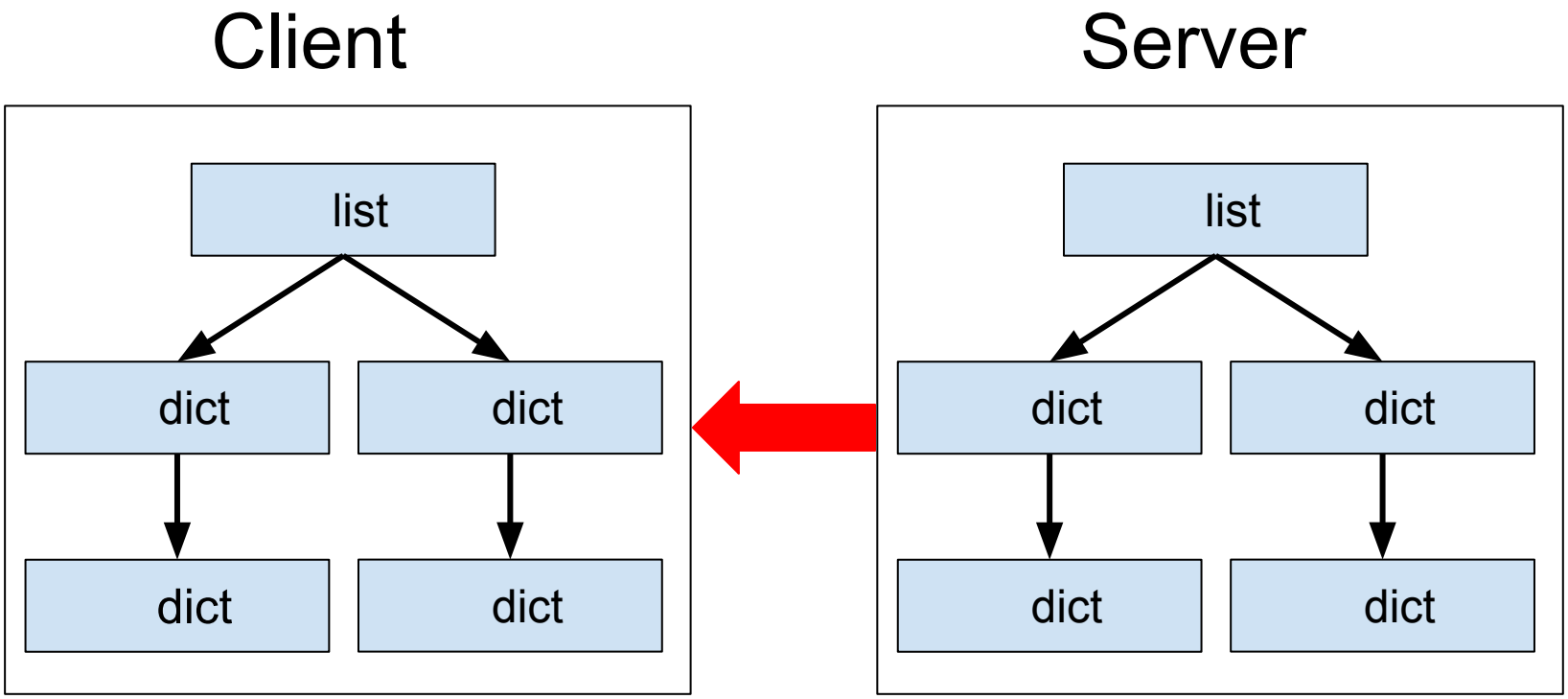
Running Example

Example (cont.):



Running Example

Example (cont.):



Agenda

- **Communicating char sequences**
- Communicating JSON docs
- Communicating pickled Python objects

Communicating Char Sequences

- **Question:**
  - How to communicate objects between client and server?
- **Answer 1:**
  - Communicate char sequences

Communicating Char Sequences

- See **commchars** app

Server: On host 192.168.1.8

```
$ python server.py 55555 (1)
Opened server socket     (1)
Accepted connection      (3)
Wrote to client          (3)
```

Client

```
$ python client.py 192.168.1.8 55555 (2)
COS 217                              (4)
C Programming, 88.55                 (4)
                                     (4)
COS 333                              (4)
The Practice of Programming, 35.14   (4)
                                     (4)
$
```

Communicating Char Sequences

- See **commchars** app (cont.)
  - **client.py**
  - **server.py**

Agenda

- Communicating char sequences
- **Communicating JSON docs**
- Communicating pickled Python objects

Communicating JSON

- **Question:**
  - How to communicate objects between client and server?
- **Answer 1:**
  - Communicate characters
- **Answer 2:**
  - Communicate JSON

Aside: JSON

- ***JavaScript Object Notation (JSON)***
  - *[Unavoidable forward reference to JavaScript…]*
  - A ***JSON document*** is a char sequence representation of a JavaScript data structure
  - JavaScript data structure can consist of:
    - Strings, Numbers, Booleans, or null
    - Objects or arrays having properties (fields) that are Strings, Numbers, Booleans, null, objects, or arrays

Aside: JSON

- **Observation**
  - A char sequence rep of a **JavaScript** data structure is similar to a char sequence rep of a **Python** data structure
  - So…
  - We can think of a JSON document as a char sequence representation of a **Python** data structure

Aside: JSON

- `json_doc = json.dumps(obj)`
  - Converts Python object to JSON doc
- `obj = json.loads(json_doc)`
  - Converts JSON doc to Python object

Aside: JSON

```
$ python
Python 3.12.4 (main, Jun  8 2024, 18:29:57) [GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>> import json
>>> somelist1 = ['Ruth', 'Gehrig']
>>> somelist
['Ruth', 'Gehrig']
>>> jsondoc = json.dumps(somelist)
>>> jsondoc
'["Ruth", "Gehrig"]'
>>> somelist2 = json.loads(jsondoc)
>>> somelist2
['Ruth', 'Gehrig']
>>>
```

Python object of class list

Python object of class str

Python object of class list

Aside: JSON

```
$ python
Python 3.12.4 (main, Jun  8 2024, 18:29:57) [GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>> import json
>>> somedict1 = {'Ruth': 3, 'Gehrig': 4}
>>> somedict1
{'Ruth': 3, 'Gehrig': 4}
>>> jsondoc = json.dumps(somedict1)
>>> jsondoc
'{"Ruth": 3, "Gehrig": 4}'
>>> somedict2 = json.loads(jsondoc)
>>> somedict2
{'Ruth': 3, 'Gehrig': 4}
>>>
```

Python object of class dict

Python object of class str

Python object of class dict

Aside: JSON

```
$ python
Python 3.12.3 (main, Sep 11 2024, 14:17:37) [GCC 13.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> book0 = {'title': 'C Programming', 'price': 88.55}
>>> book1 = {'title': 'The Practice of Programming', 'price': 35.14}
>>> course0 = {'name': 'COS 217', 'book': book0}
>>> course1 = {'name': 'COS 333', 'book': book1}
>>> courses = [course0, course1]
>>> courses
[{'name': 'COS 217', 'book': {'title': 'C Programming', 'price': 88.55}},
{'name': 'COS 333', 'book': {'title': 'The Practice of Programming', 'price':
35.14}}]
>>> import json
>>> jsondoc = json.dumps(courses)
>>> jsondoc
'[{"name": "COS 217", "book": {"title": "C Programming", "price": 88.55}},
{"name": "COS 333", "book": {"title": "The Practice of Programming", "price":
35.14}}]'
>>> courses2 = json.loads(jsondoc)
>>> courses2
[{'name': 'COS 217', 'book': {'title': 'C Programming', 'price': 88.55}},
{'name': 'COS 333', 'book': {'title': 'The Practice of Programming', 'price':
35.14}}]
>>>
```

Python object of class list

Python object of class str

Python object of class list

Communicating JSON

- See **commjson** app

Server: On host 192.168.1.8

```
$ python server.py 55555 (1)
Opened server socket     (1)
Accepted connection      (3)
Wrote to client          (3)
```

Client

```
$ python client.py 192.168.1.8 55555 (2)
COS 217                               (4)
C Programming, 88.55                  (4)
                                      (4)
COS 333                               (4)
The Practice of Programming, 35.14    (4)
                                      (4)
$
```

Communicating JSON

- See **commjson** app (cont.)
  - **client.py**
  - **server.py**

Communicating JSON

- See **commjson** app (cont.)
  - Alternative
    - Server:
      - ~~`flo.write(json_str + '\n')`~~
      - `flo.write(json_str)`
    - Client:
      - ~~`flow.readline()`~~
      - `flow.read()`
  - Problem with the alternative
    - Works for **one-way** comm, but not for **two-way** comm
    - (Won't work in Assignment 2)

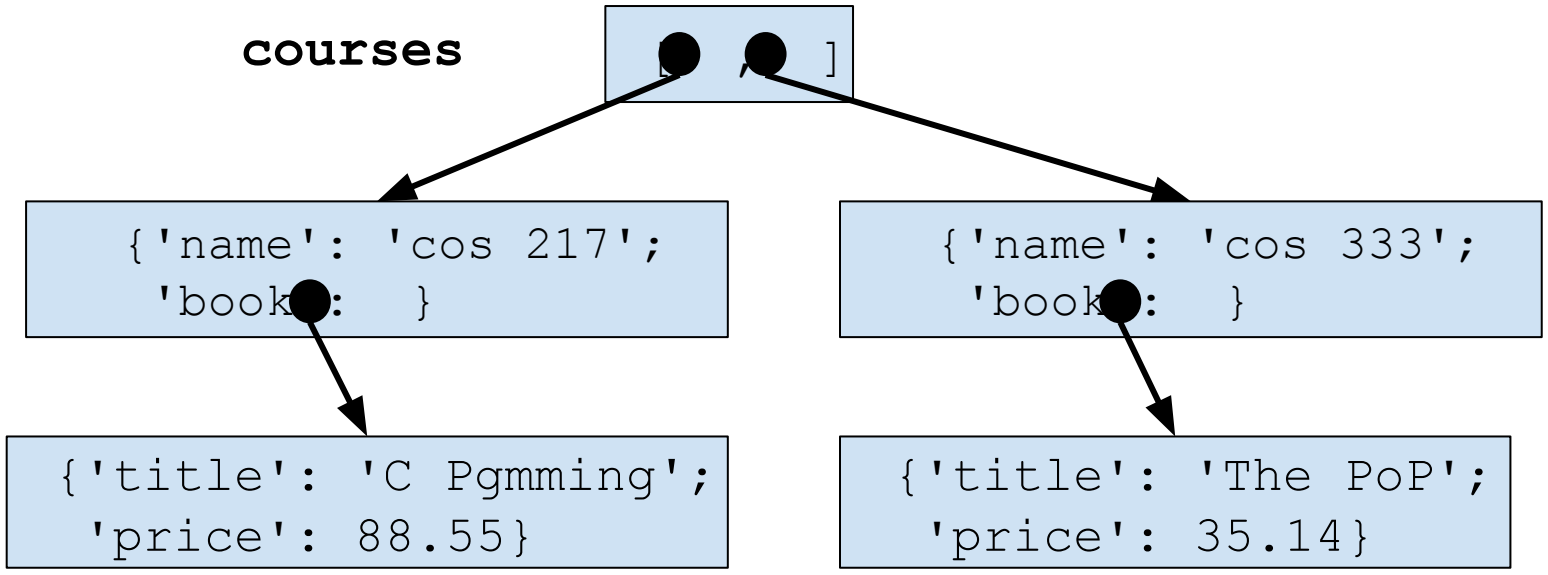
Communicating JSON

Previously:

```
book0 = {'title': 'C Programming', 'price': 88.55}
book1 = {'title': 'The Practice of Programming', 'price': 35.14}
course0 = {'name': 'COS 217', 'book': book0}
course1 = {'name': 'COS 333', 'book': book1}
courses = [course0, course1]
```

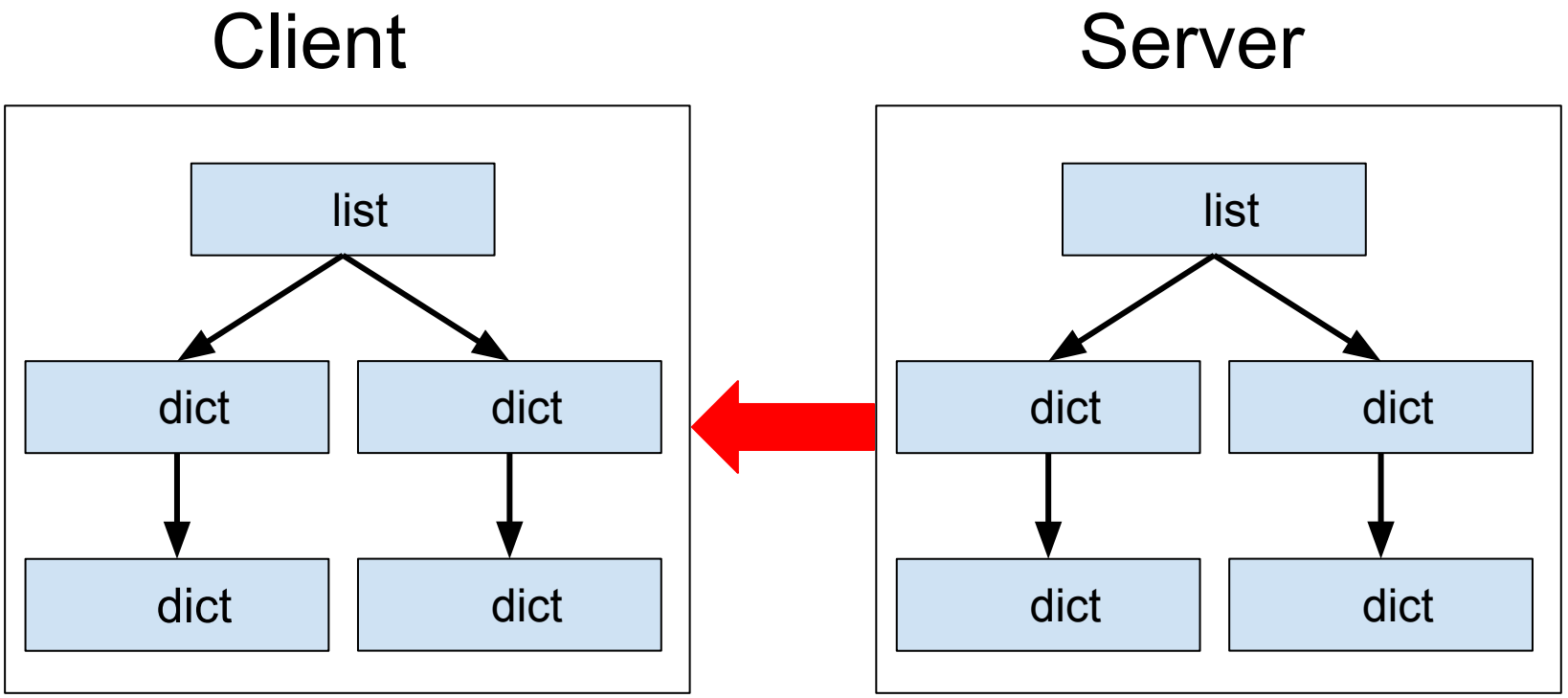
Communicating JSON

Previously:



Communicating JSON

Previously:



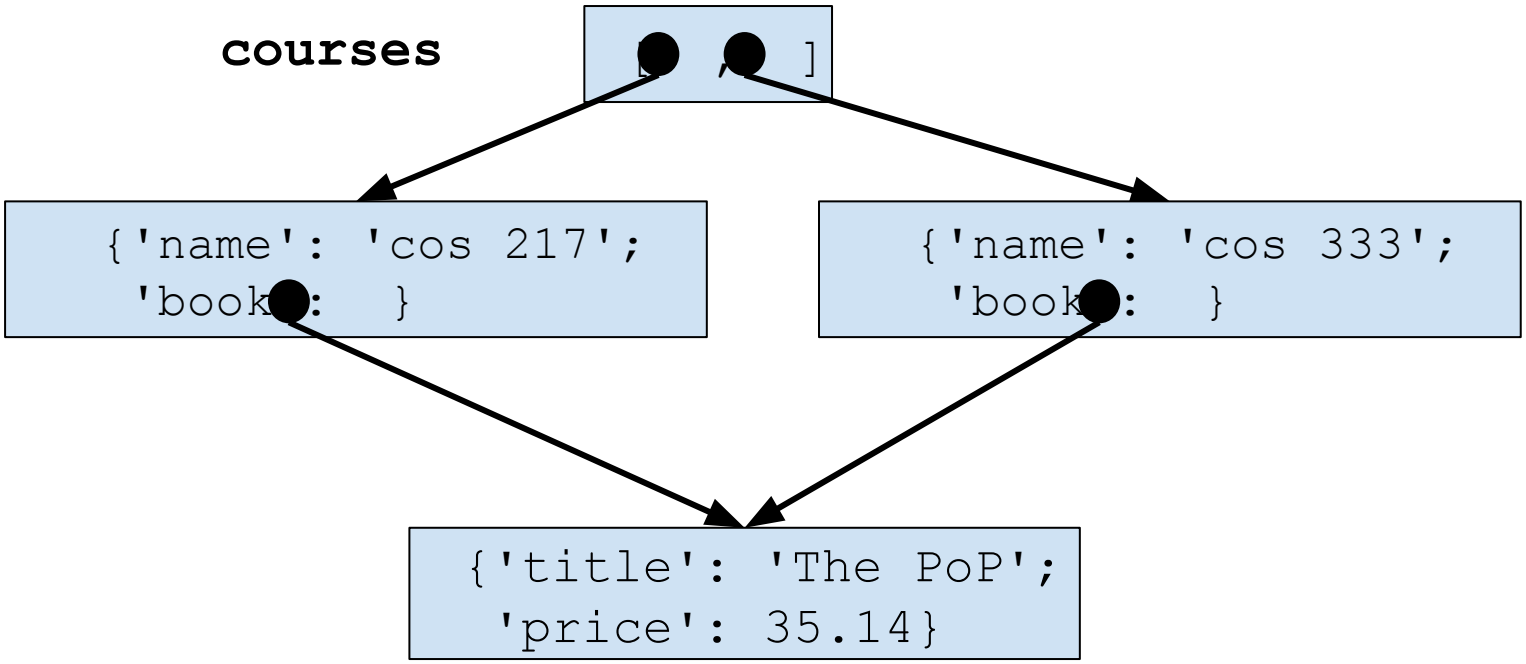
Communicating JSON

Now:

```
book0 = {'title': 'C Programming', 'price': 88.55}
book1 = {'title': 'The Practice of Programming', 'price': 35.14}
course0 = {'name': 'COS 217', 'book': book1}
course1 = {'name': 'COS 333', 'book': book1}
courses = [course0, course1]
```

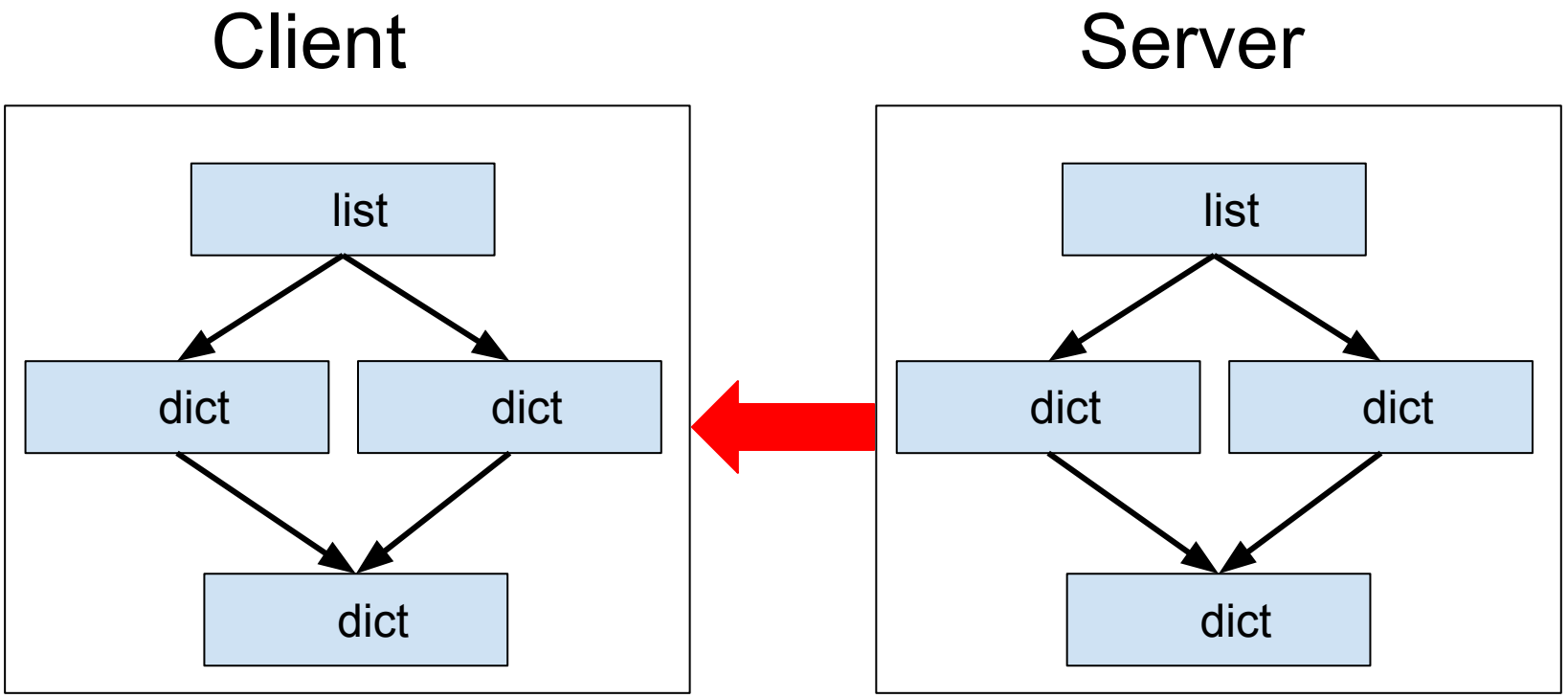
Communicating JSON

Now:



Communicating JSON

Example (cont.):



Communicating JSON

- See **commjsonbad** app
  - **client.py**
  - **server.py**

Agenda

- Communicating char sequences
- Communicating JSON docs
- **Communicating pickled Python objects**

Communicating Pickles

- **Question:**
  - How to communicate objects between client and server?
- **Answer 1:**
  - Communicate char sequences
- **Answer 2:**
  - Communicate JSON docs
- **Answer 3:**
  - Communicate pickled Python objects

Aside: Python Pickles

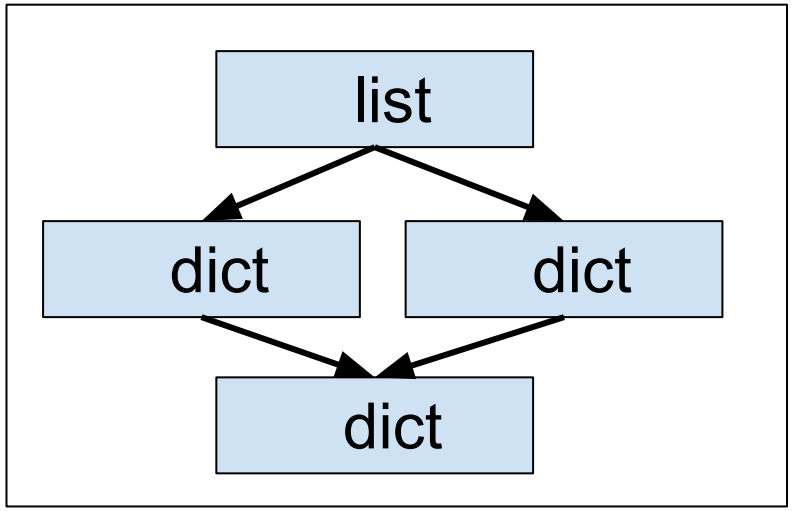
- `pickle.dump(obj, flo)`
  - Serializes `obj` to byte sequence
  - Writes byte sequence to `flo`
- `obj = pickle.load(flo)`
  - Reads byte sequence from `flo`
  - Deserializes byte sequence to form `obj`

Communicating Pickles

- See **commpickle** app (cont.)
  - **client.py**
  - **server.py**

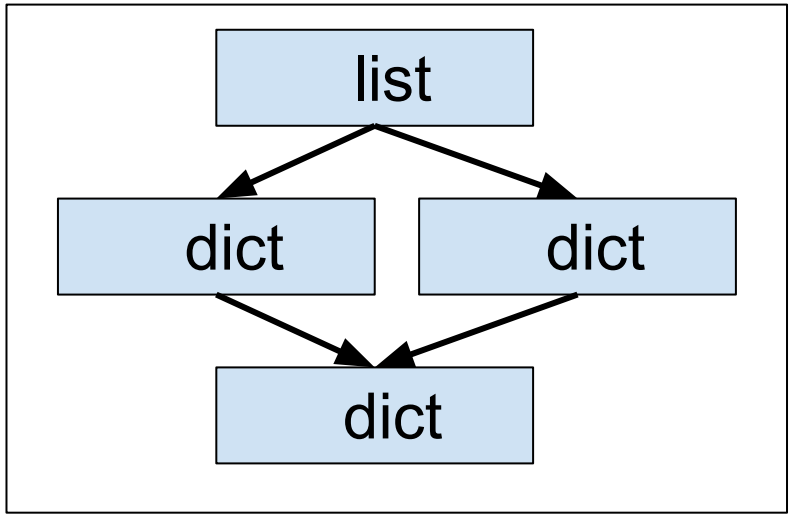
Communicating Pickles

In server:



See **commpickle** app

In client:



It works!

Summary

	Communicating JSON	Communicating pickled Python objects
Programming language independence	+	- *
Handling of object graphs	-	+

* Java has a similar mechanism: `Serializable` interface

Summary

- We have covered:
  - Network programming concepts
  - Network programming in Python
    - How to compose a client
    - How to compose a server
    - How to communicate char sequences
    - How to communicate JSON docs
    - How to communicate pickled Python objects
- See also…

Summary

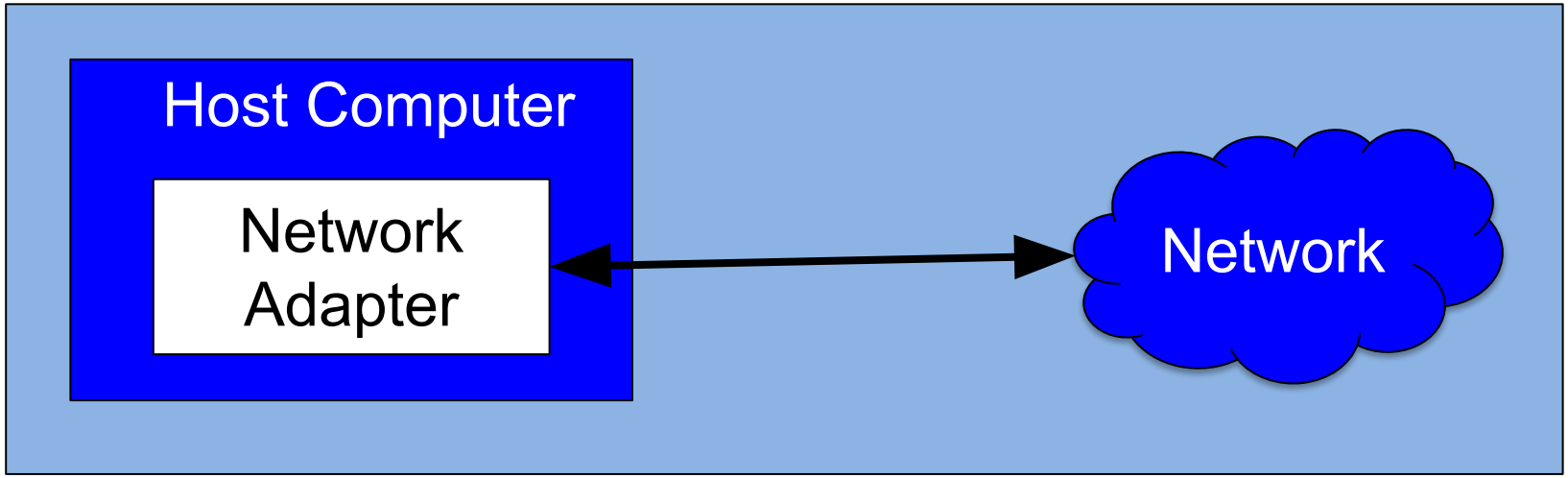
- **Appendix 1**: Network Basics
- **Appendix 2**: The Internet

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

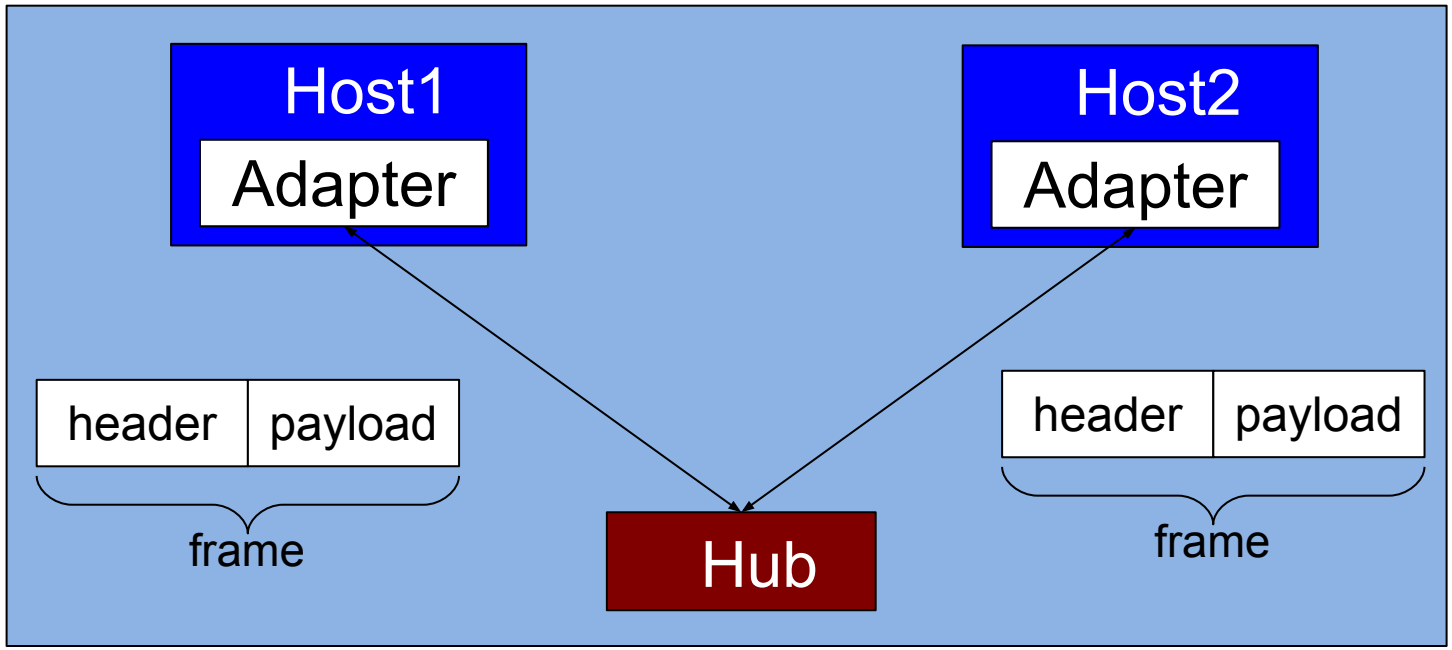
xx:xx:xx is unique to vendor

yy:yy:yy is unique to device

Network Basics

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

Network Basics



Network Segment

Frame = header + payload

Header identifies source and destination addresses

Payload contains user data

Hub (alias **repeater**) = HW (no SW) that transmits frames among hosts

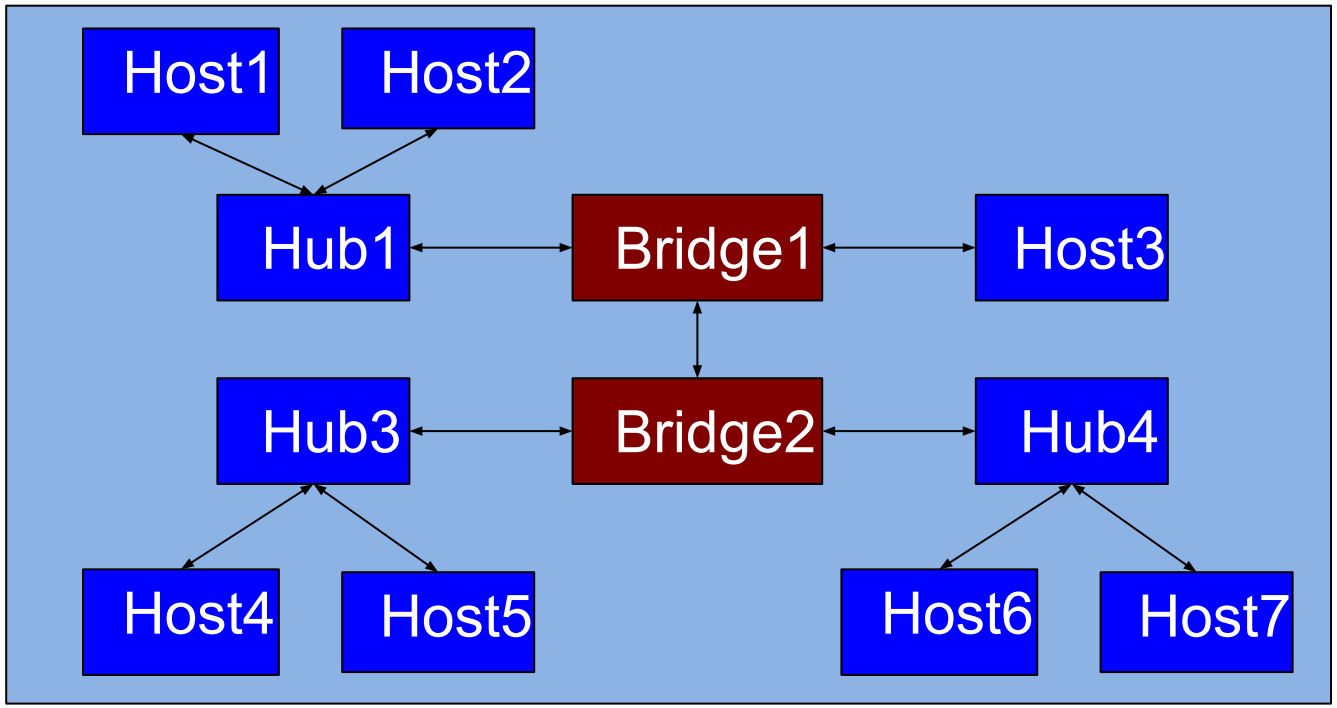
Receives frame from adapter; doesn't examine header; copies to all others

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

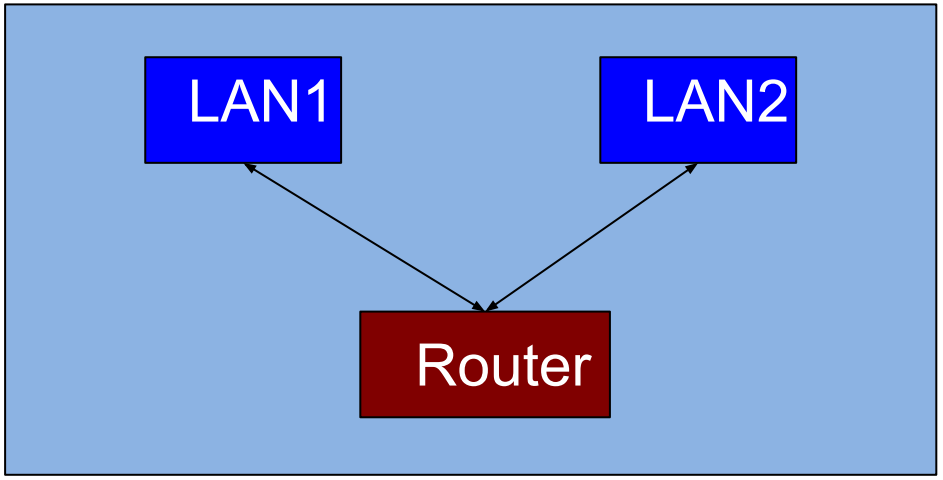
host1 → host2: hub1 → bridge1 → (discard)

host1 → host7: hub1 → bridge1 → bridge2 → hub4

Local Area Network (LAN) = hosts + hubs + bridges

Scope: one building; one campus

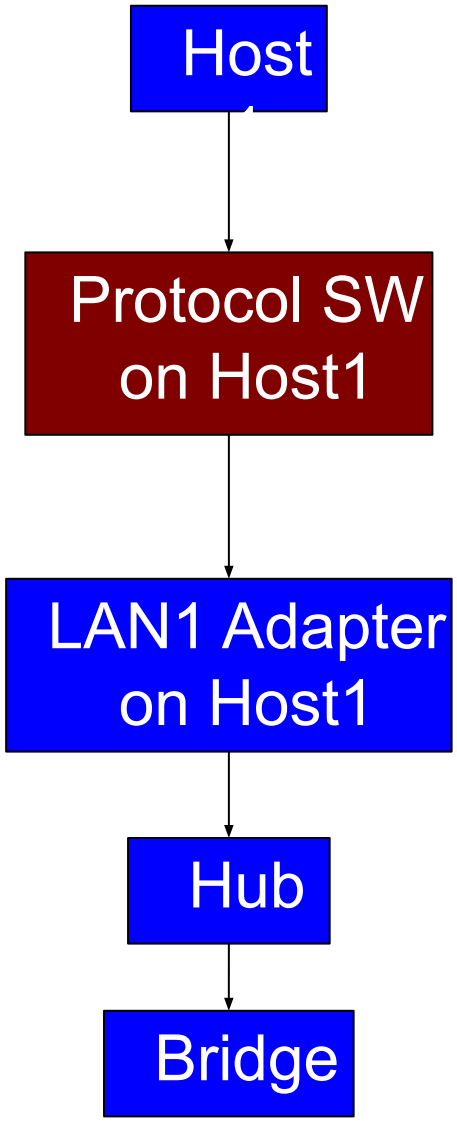
Network Basics



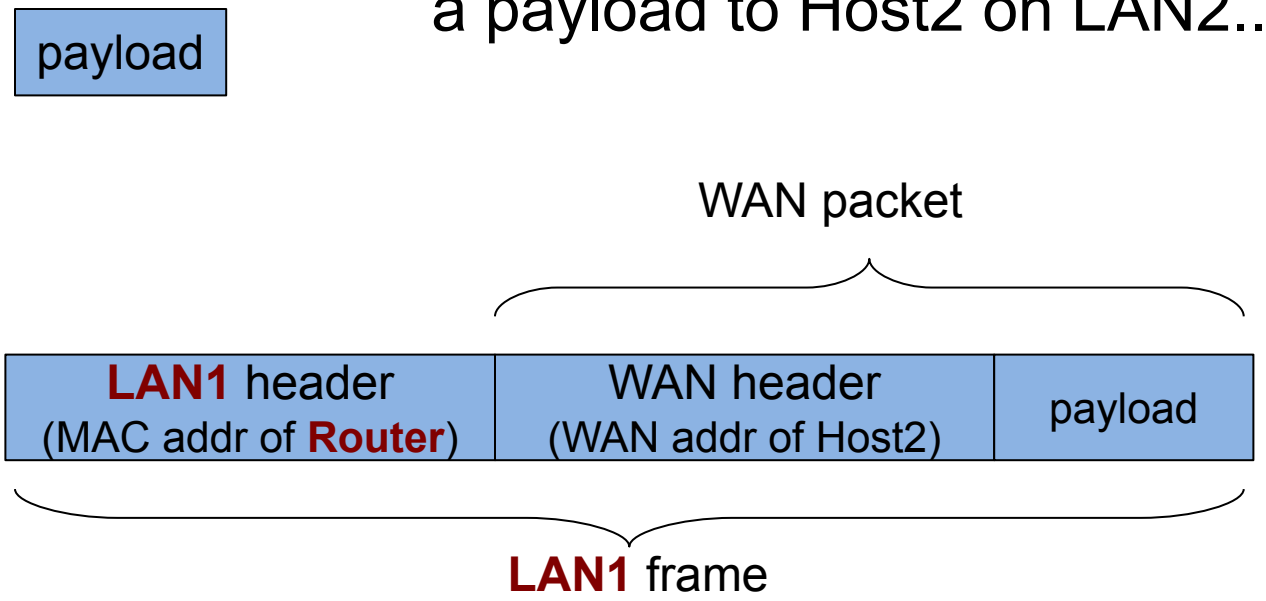
WAN

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

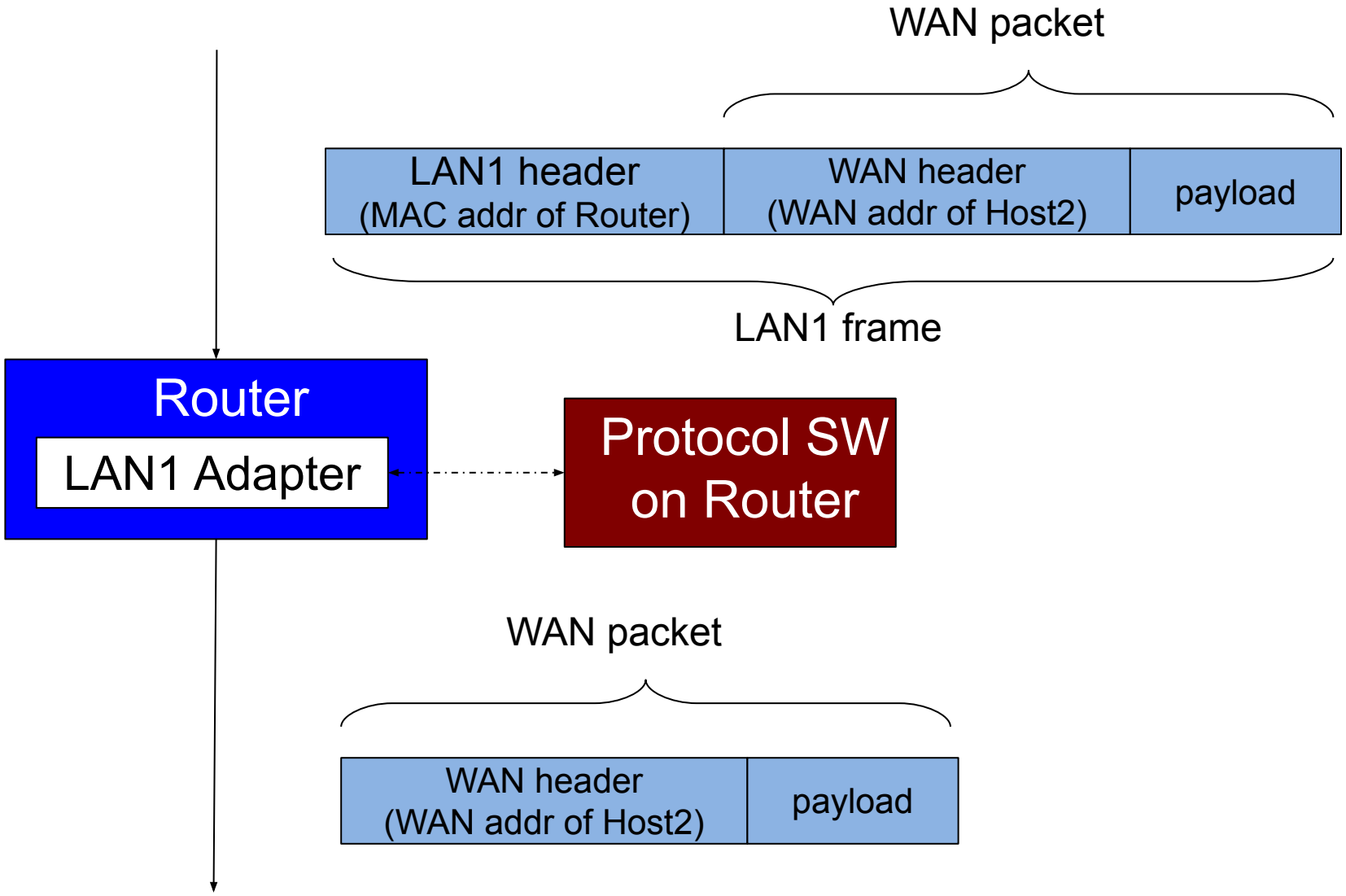
Network Basics



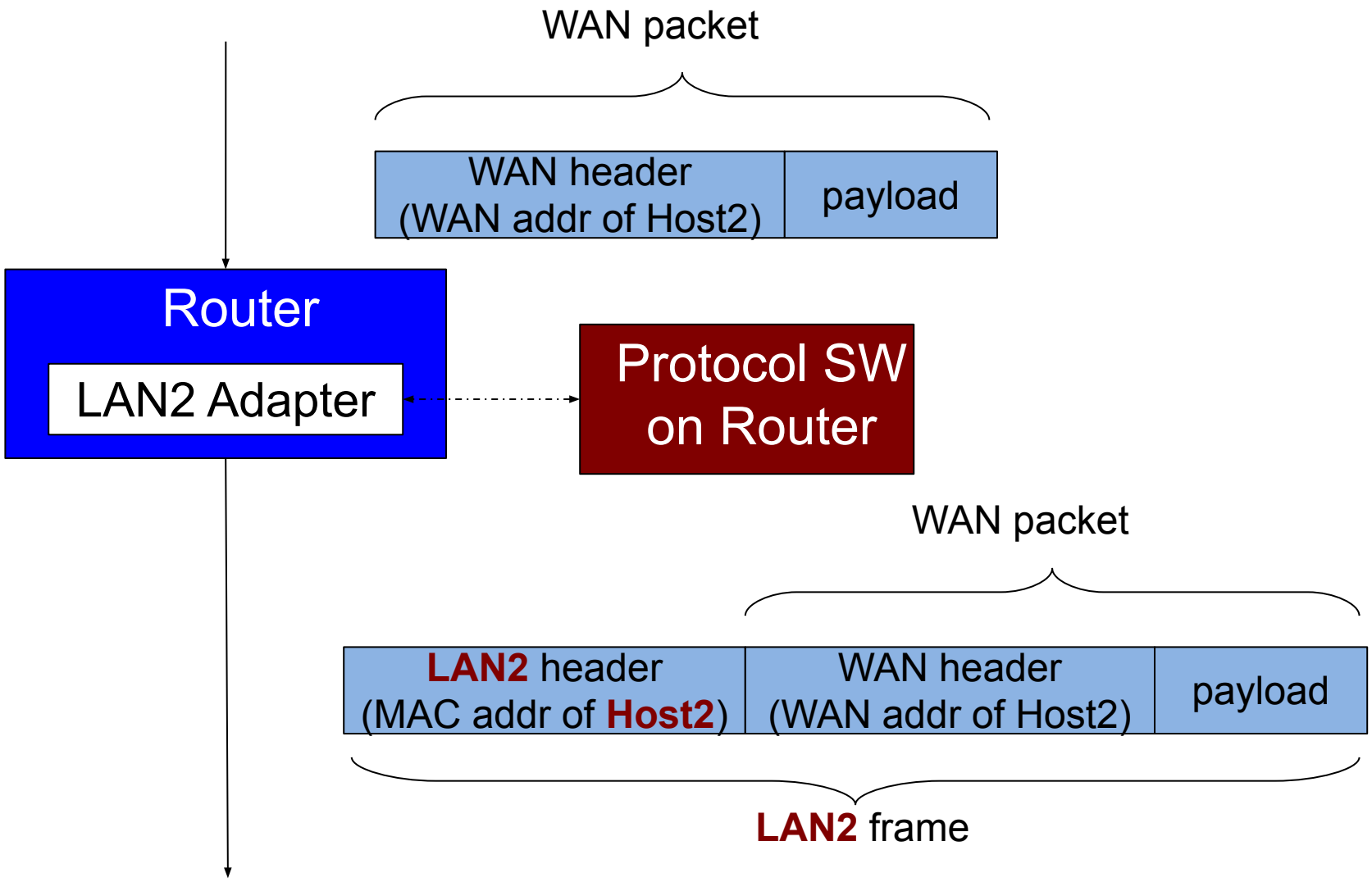
When Host1 on LAN1 sends a payload to Host2 on LAN2...



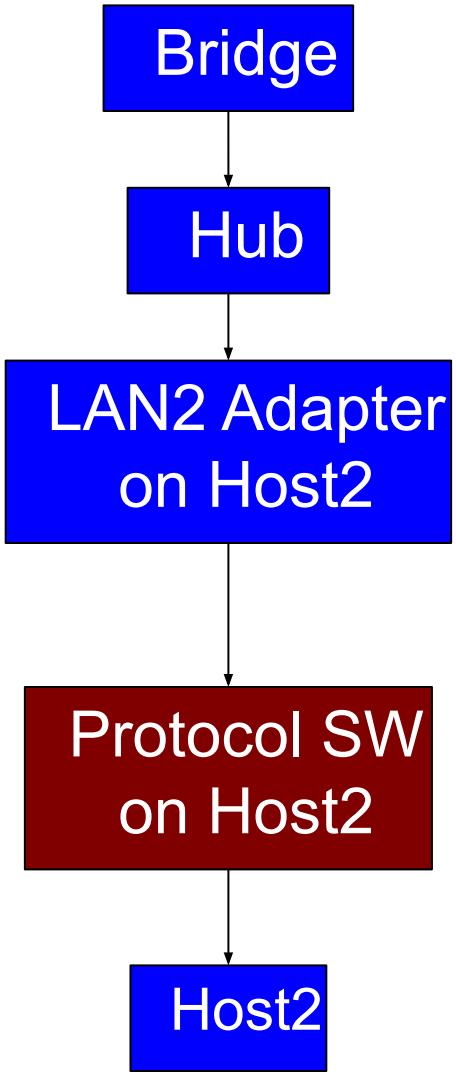
Network Basics

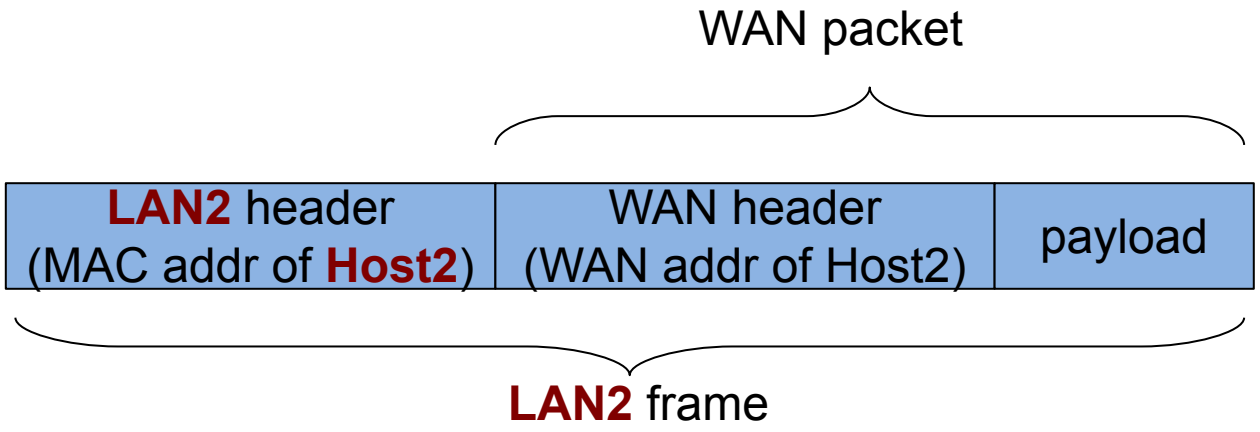


Network Basics



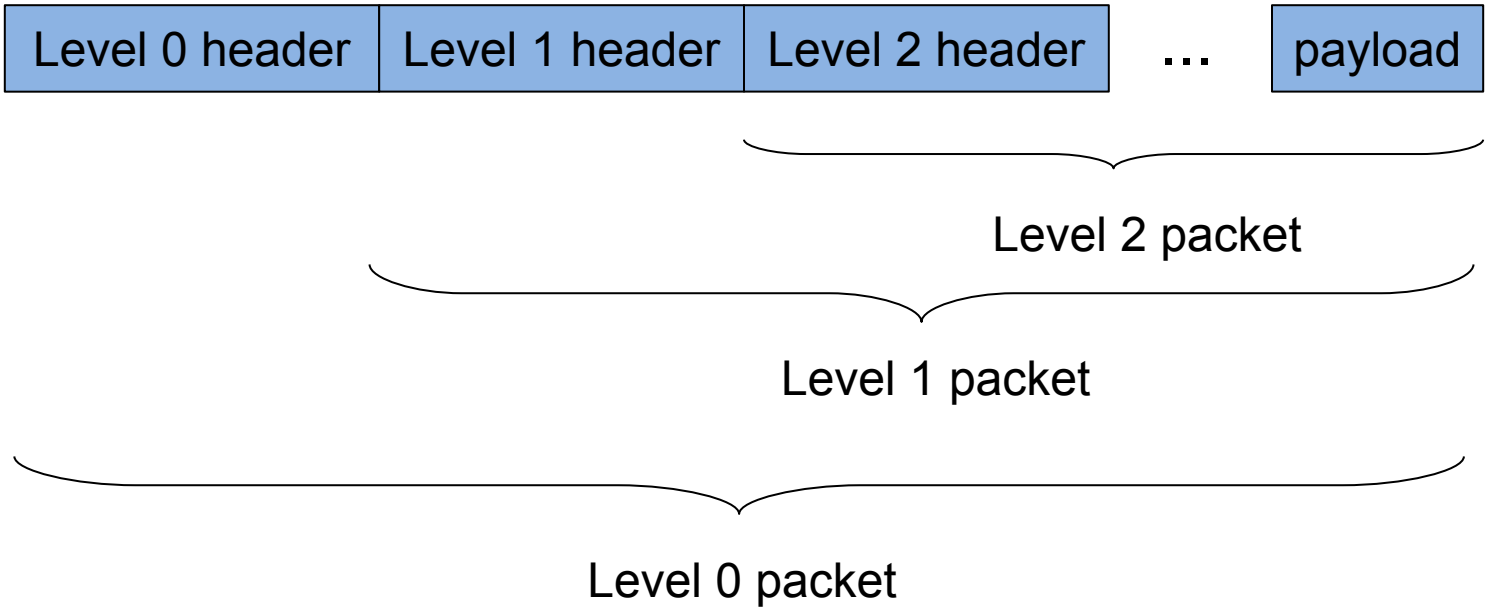
Network Basics





Network Basics

Generic Packet Structure



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

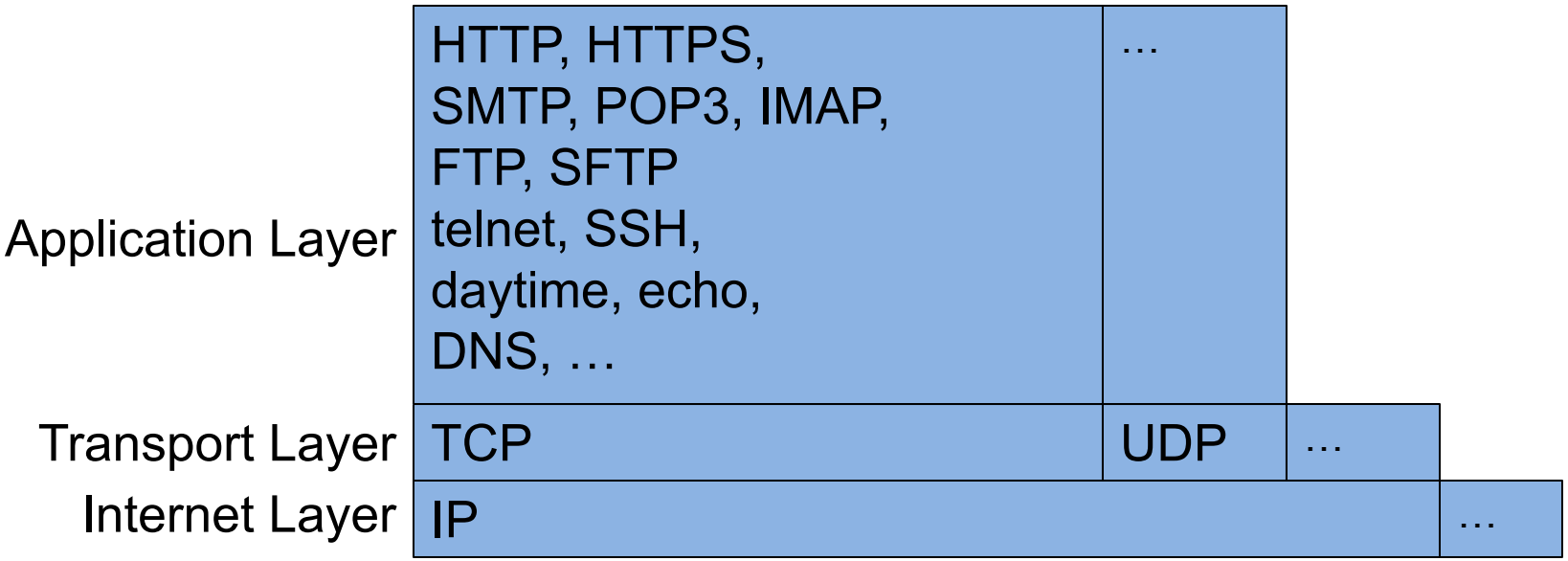
Appendix 2:
The Internet

The Internet

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

The Internet

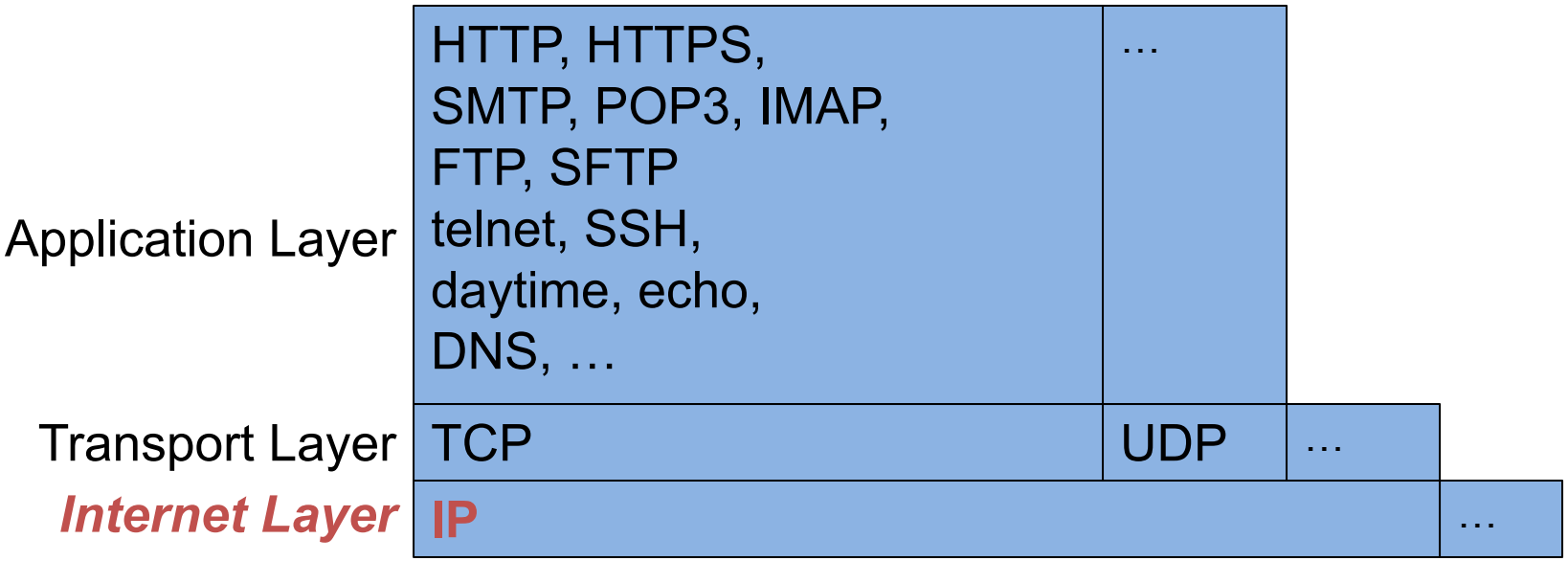
The Internet Protocol Stack



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
128.112.136.61	Princeton CS Dept computer
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: 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***

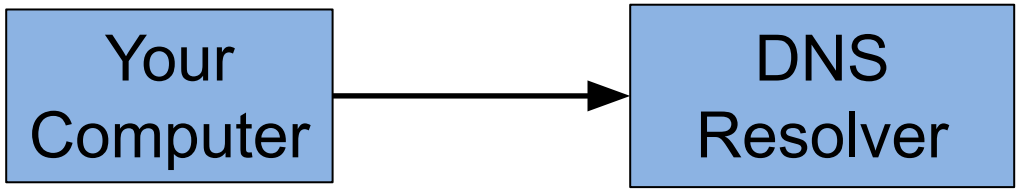
The Internet

- ***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

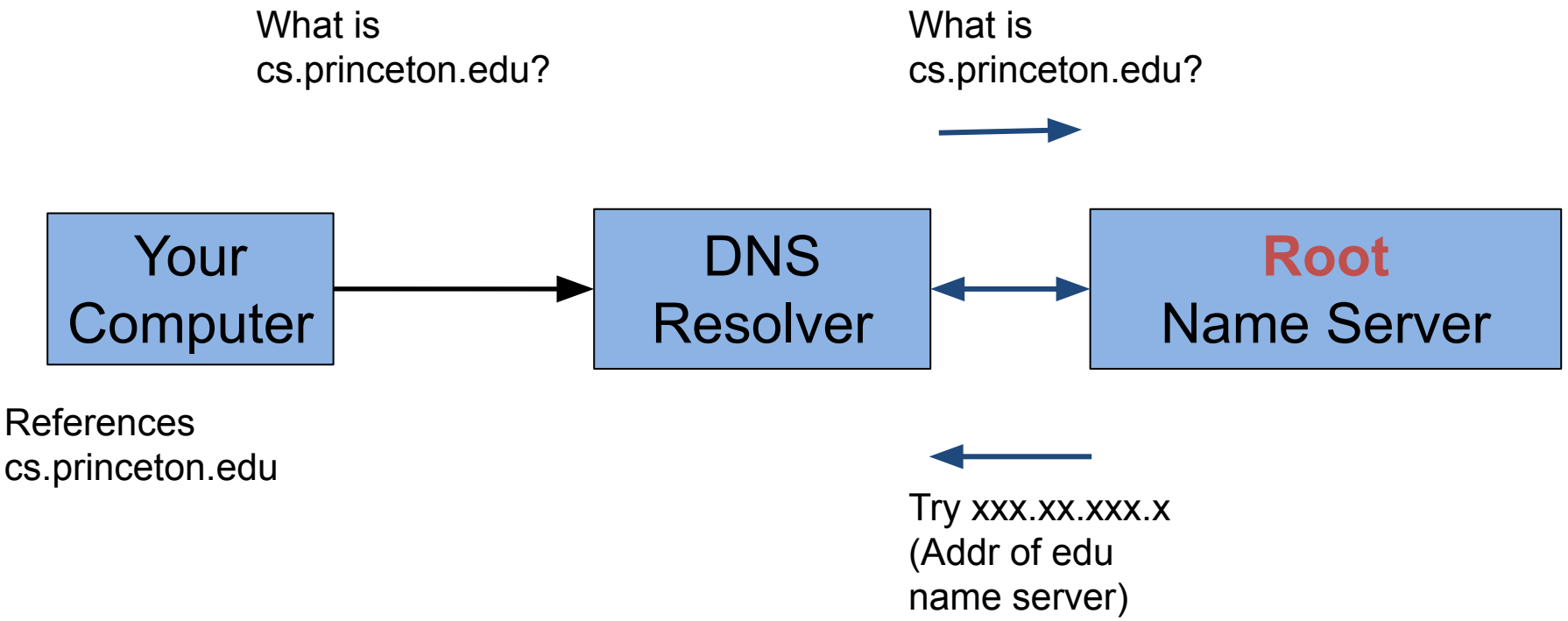
What is
cs.princeton.edu?



References
cs.princeton.edu

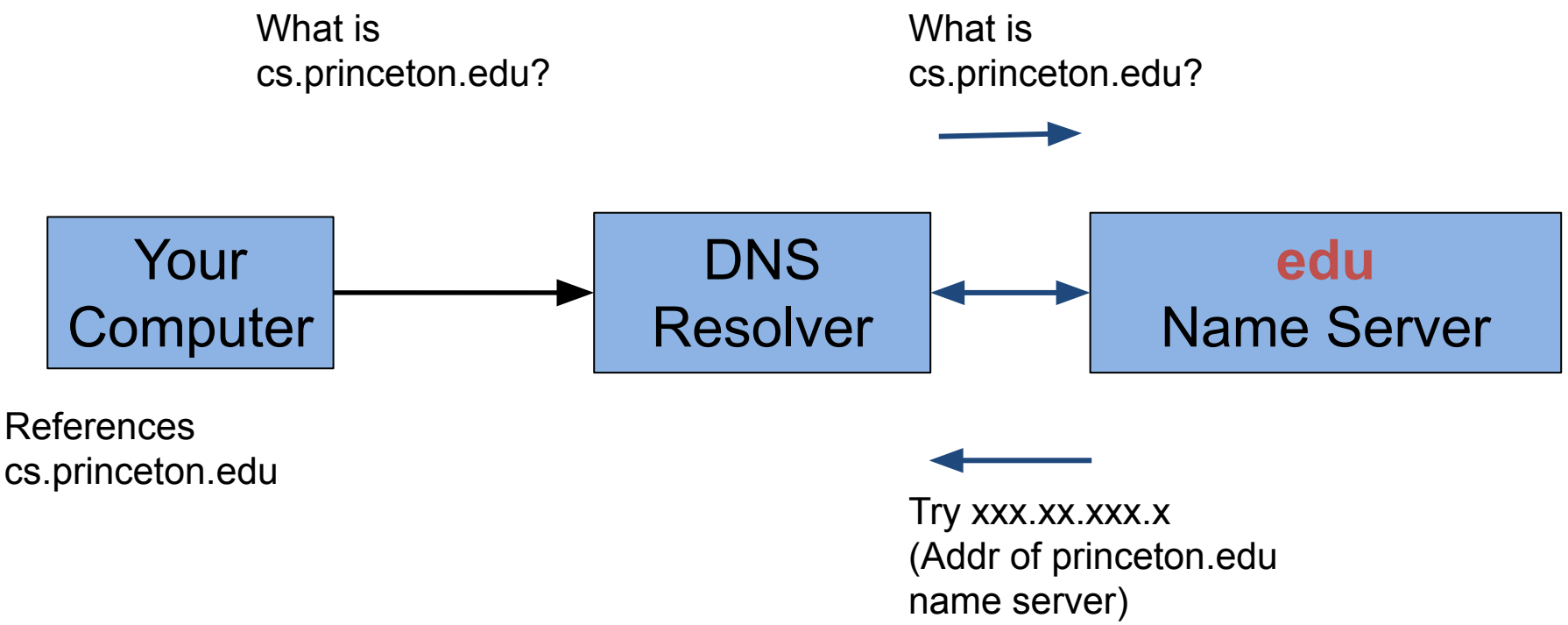
The Internet

The Domain Name System



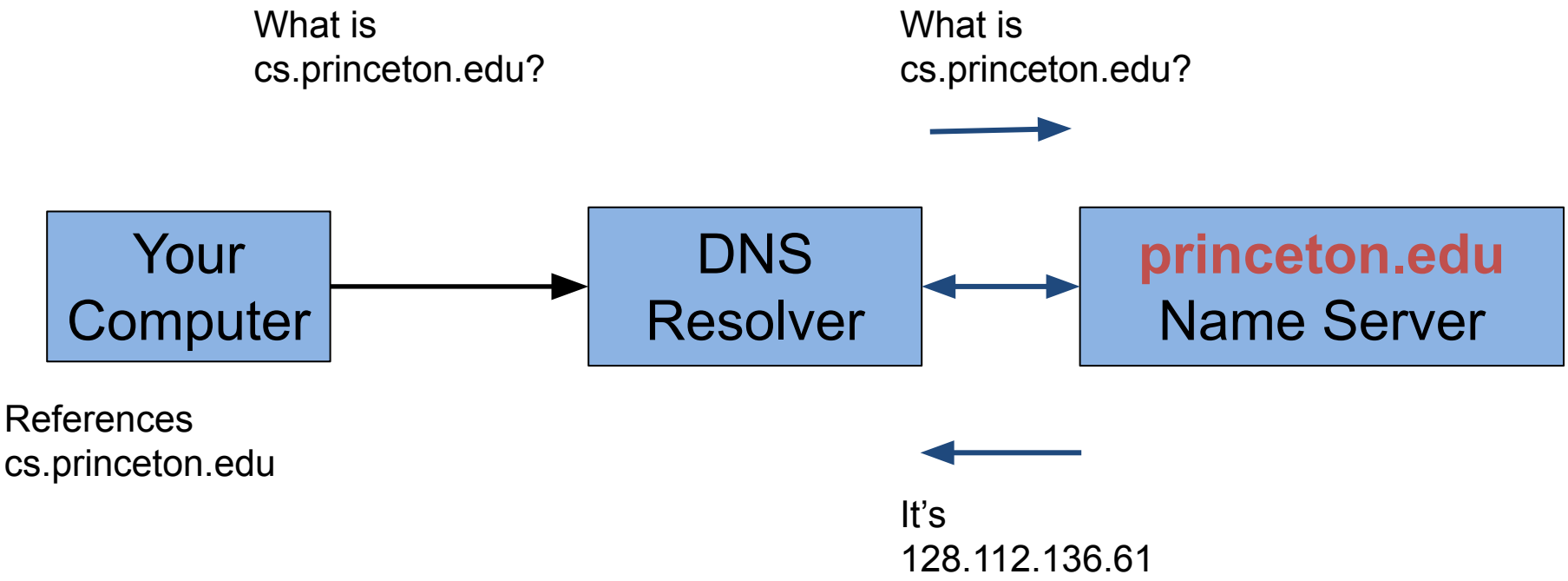
The Internet

The Domain Name System



The Internet

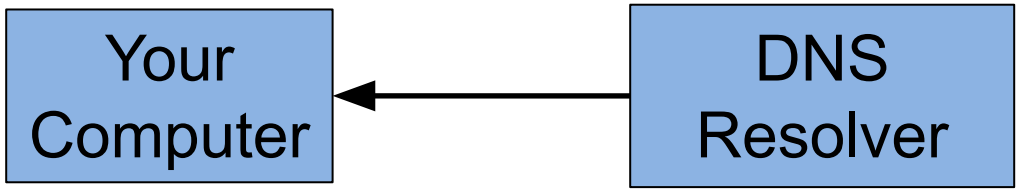
The Domain Name System



The Internet

The Domain Name System

It's
128.112.136.61



References
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

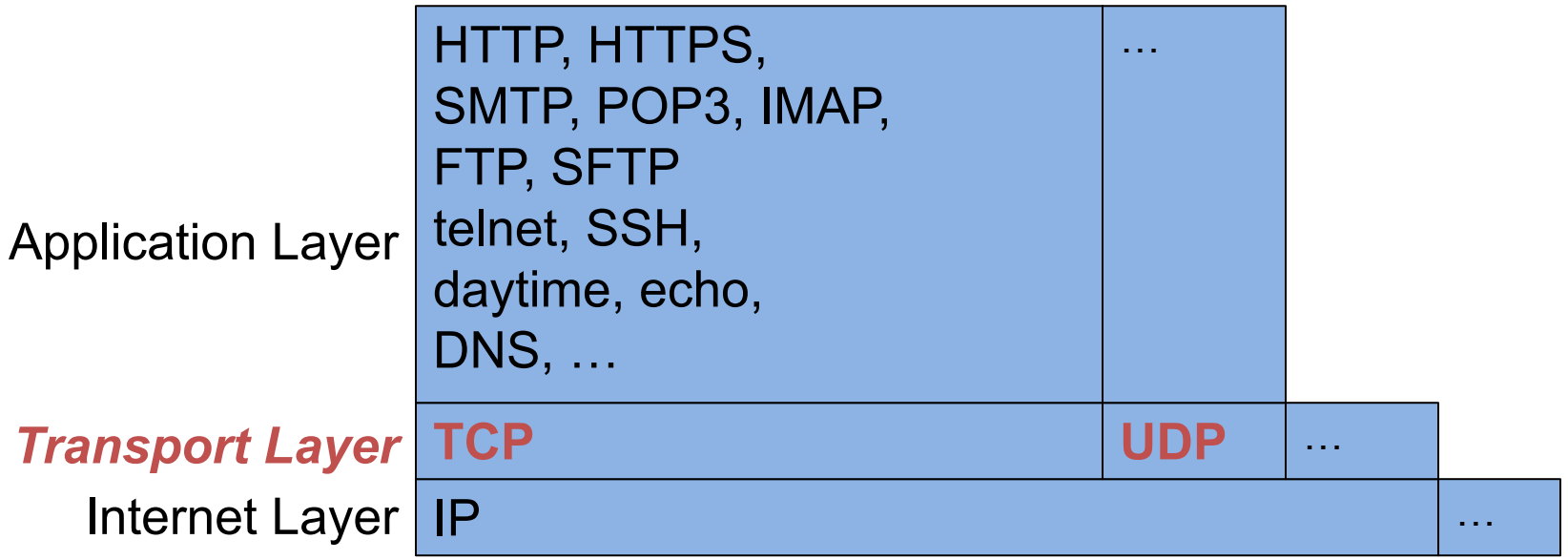
- Try (on Linux, Mac, MS Windows):
  - `nslookup cs.princeton.edu`
  - `nslookup time-a.nist.gov`
  - `nslookup nonexistingdomainname`

The Internet

- See **ipaddress.py**
  - `python ipaddress.py 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

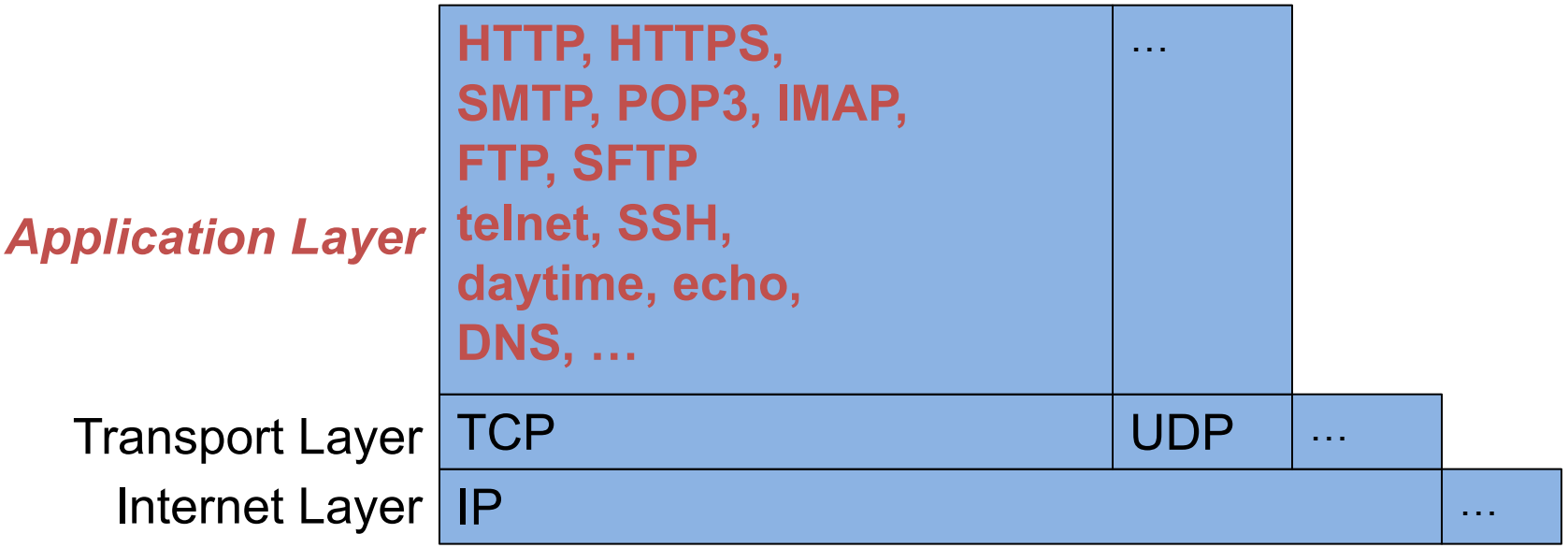
Port	Kind of Process
7	echo
13	daytime
20	FTP-control
21	FTP-data
22	SSH, SFTP
23	telnet
25	SMTP
53	DNS
80	HTTP
110	POP2
143	IMAP
443	HTTPS

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