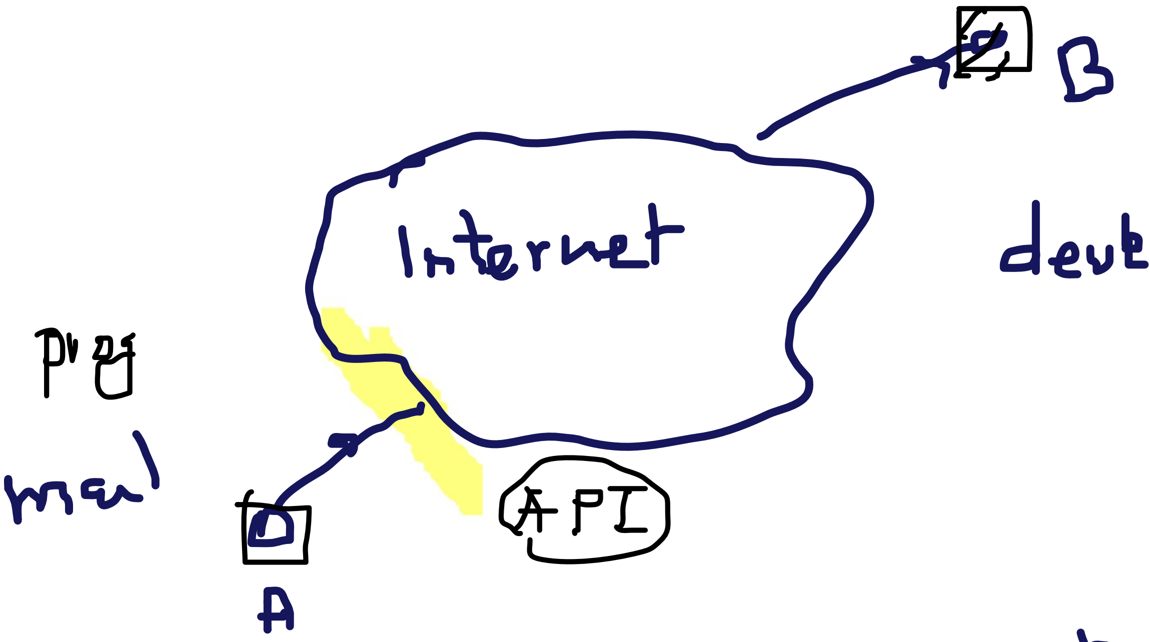


mail appl

user . Core - AB

developer . Core . BA

list of items → sort → algo

code mechanical → code

"lang" C

Network Appl.

Socket lib → most lang

Some network apps

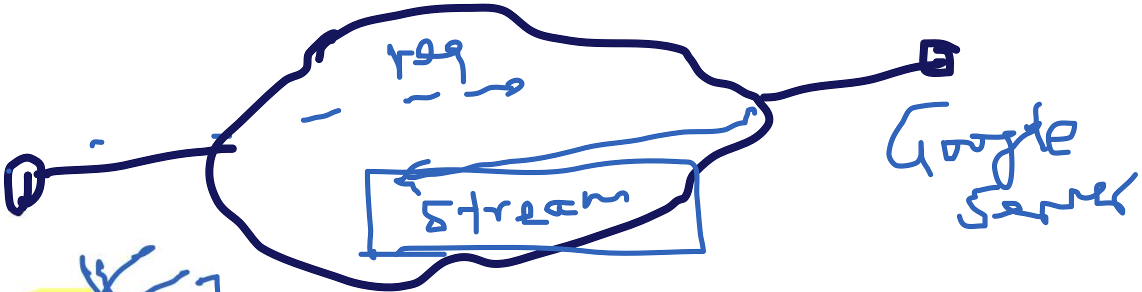
- e-mail Reliability
- web R
- text messaging R
- remote login R
- P2P file sharing torrent
- multi-user network games
- streaming stored video (YouTube, Hulu, Netflix)

- voice over IP (e.g., Skype)
- real-time video conferencing some noise is ok
- social networking
- search
- … * bitcoin / block-chain
- …

Video conf Meet vs. Youtube

→ up/down. ↑↓ download ↓

New requirements



Youtube → [Min. BW Bandwidth].

"Latency" delay between req/resp.

↳ [5 sec]

Meet/ → + Interactive appl

Chat

delay very low ≤ 200-300ms.

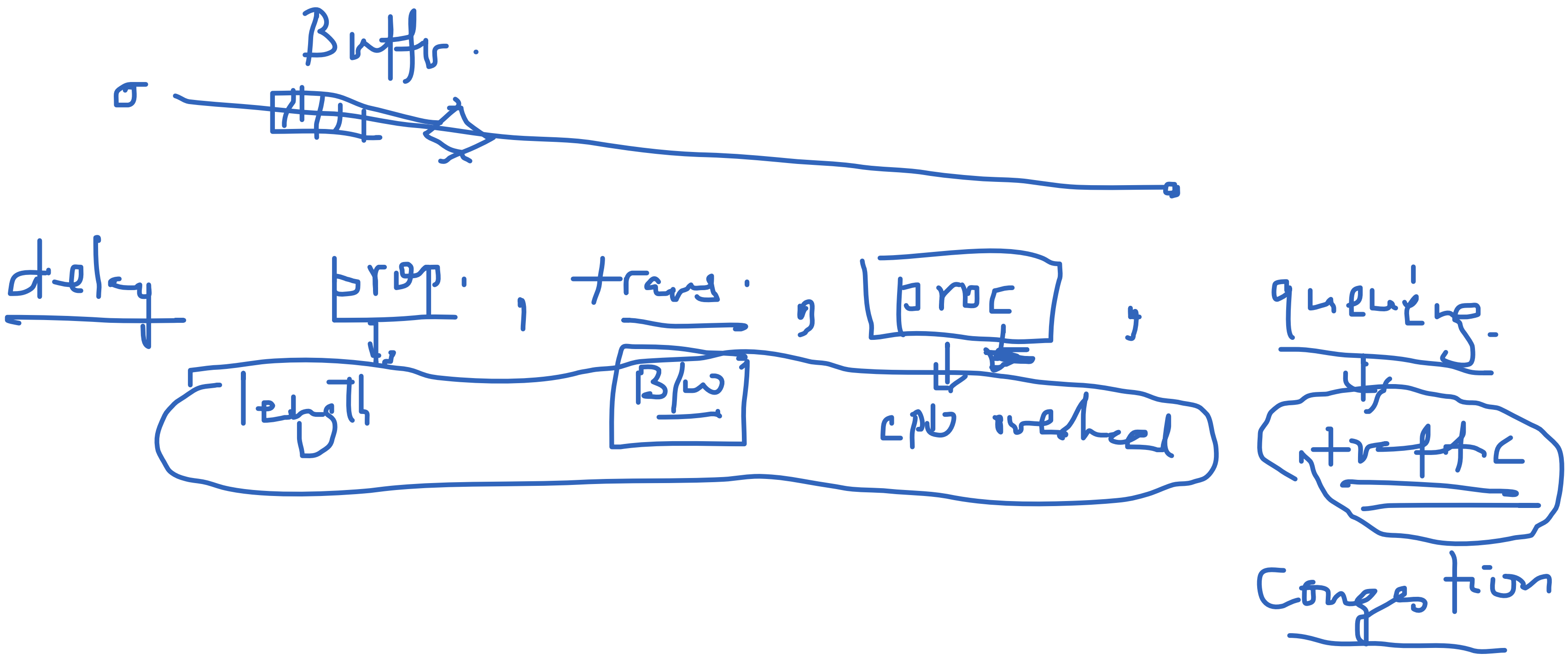
n/w is UNRELIABLE

[Router → drop packets
noise → bit errors
:
]

[Non-RT] → mail, Browsing

- Relibility ✓
- ✓ Real Time appl → + delay, B/W
  Stock xchange : ✓ , ✓ , + Rel.
- One to one appl..

Broadcast
Multicast appl
" group "


Buffer.
delay
prop.
trans.
proc
queuing.
length
B/W
cpu overhead
traffic
Congestion

N/w Arch.

Layered Arch.

"Appl Logic."

User Interface

Appl

N/w API

Socket

Reliability, delay

Transport

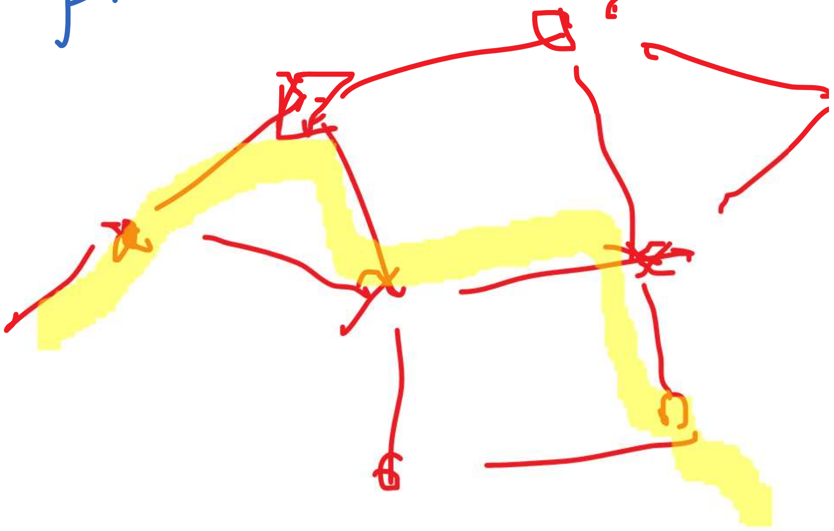
Routing, ! :

Network

Link Layer

signaling channel

"Physical Layer"



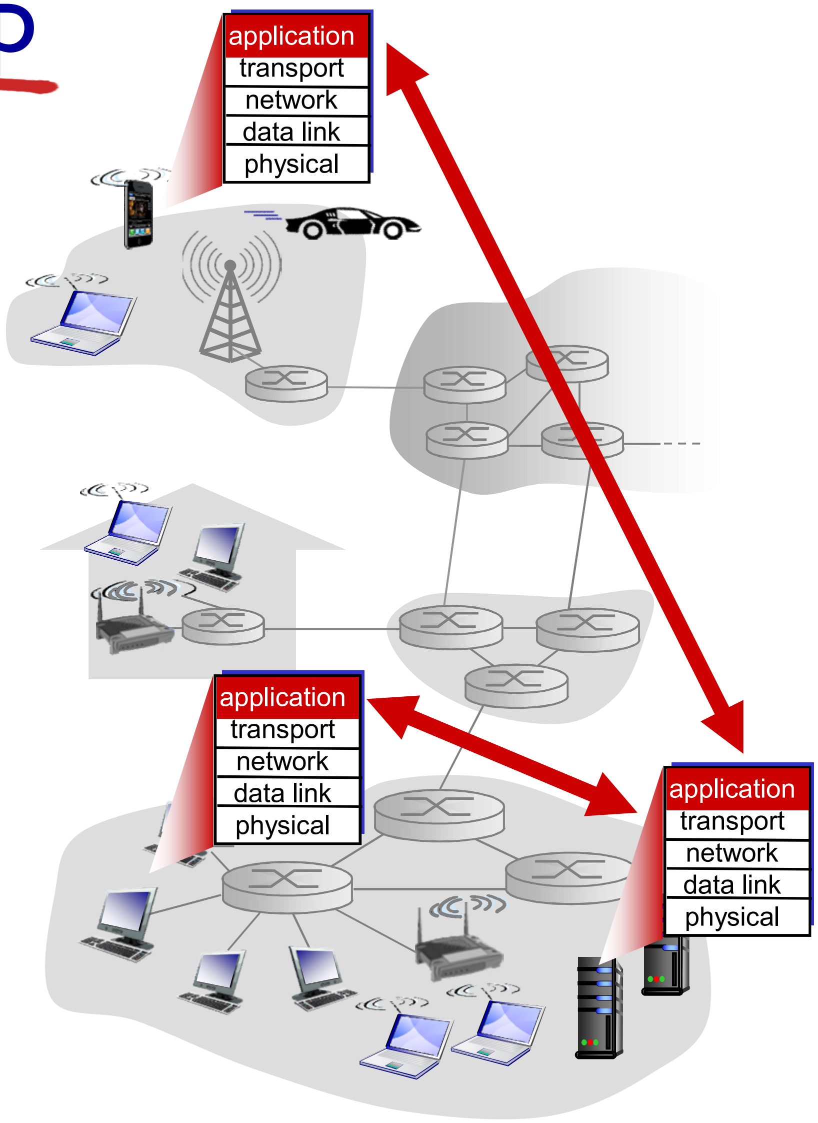
Creating a network app

write programs that:

- run on (different) *end systems*
- communicate over network
- e.g., web server software communicates with browser software

no need to write software for network-core devices

- network-core devices do not run user applications
- applications on end systems allows for rapid app development, propagation



Application architectures

possible structure of applications:

- client-server
- peer-to-peer (P2P)

Client-server architecture

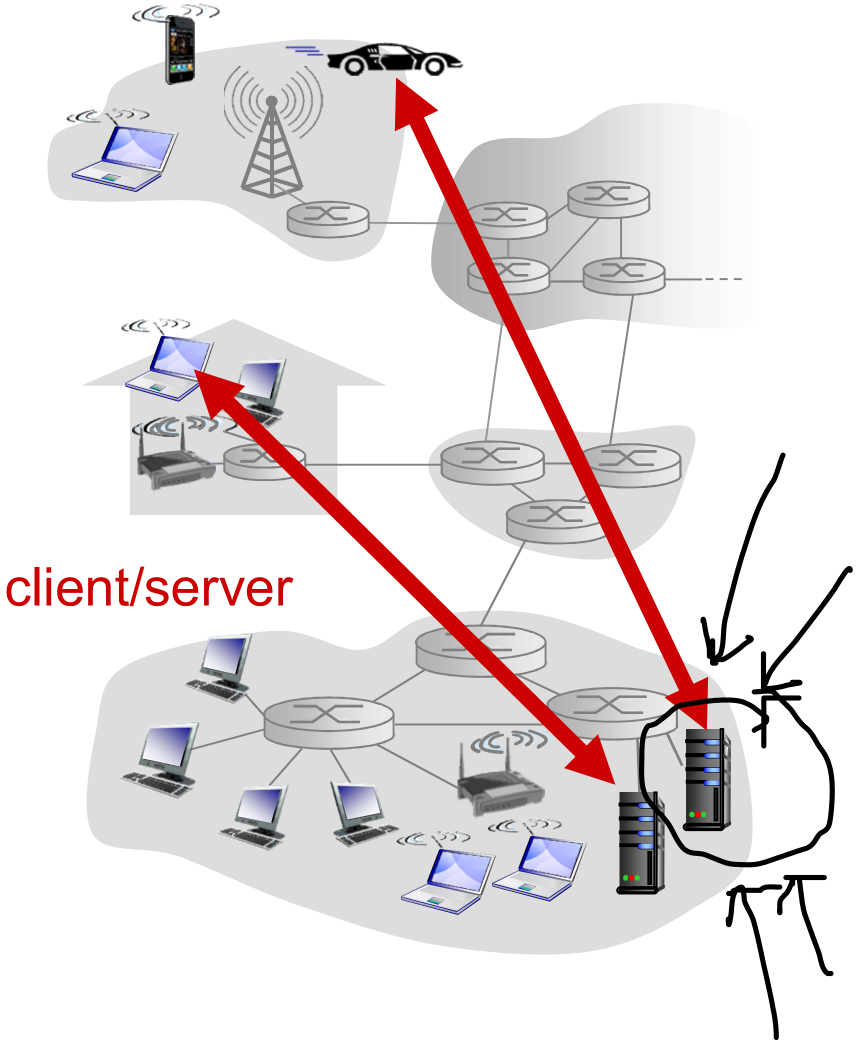
req

daiict

Active requests

passive

→ providing resource



server:

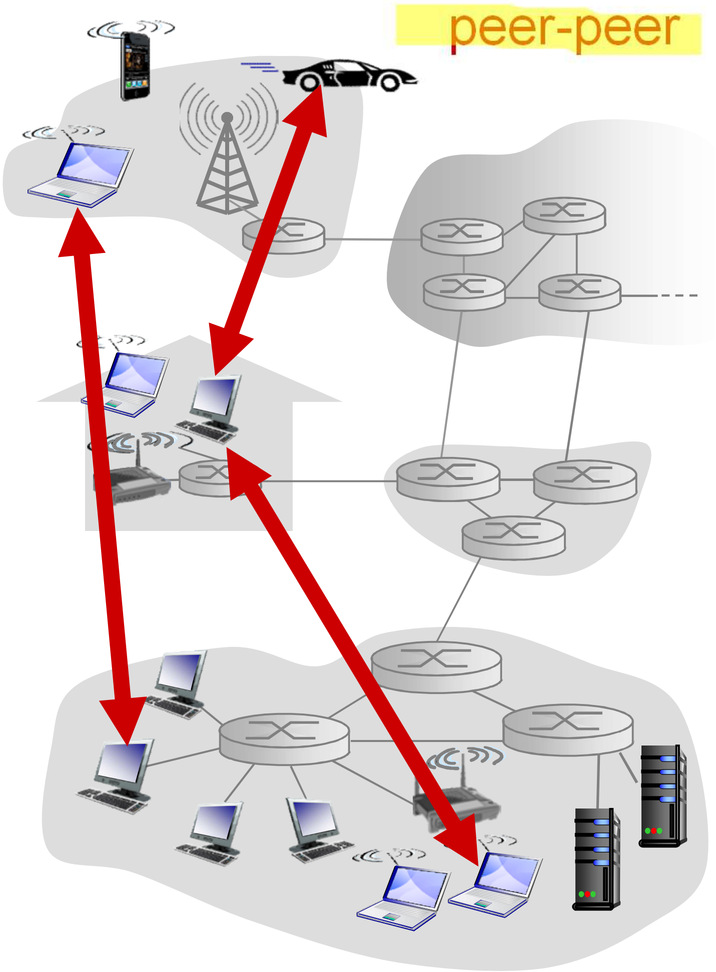
- always-on host
- permanent IP address
- data centers for scaling

clients:

- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other

P2P architecture

- *no* always-on server
- arbitrary end systems directly communicate
- peers request service from other peers, provide service in return to other peers
  - *self scalability* – new peers bring new service capacity, as well as new service demands
- peers are intermittently connected and change IP addresses
  - complex management



Processes communicating

process: program running within a host

- within same host, two processes communicate using inter-process communication (defined by OS)
- processes in different hosts communicate by exchanging messages

clients, servers

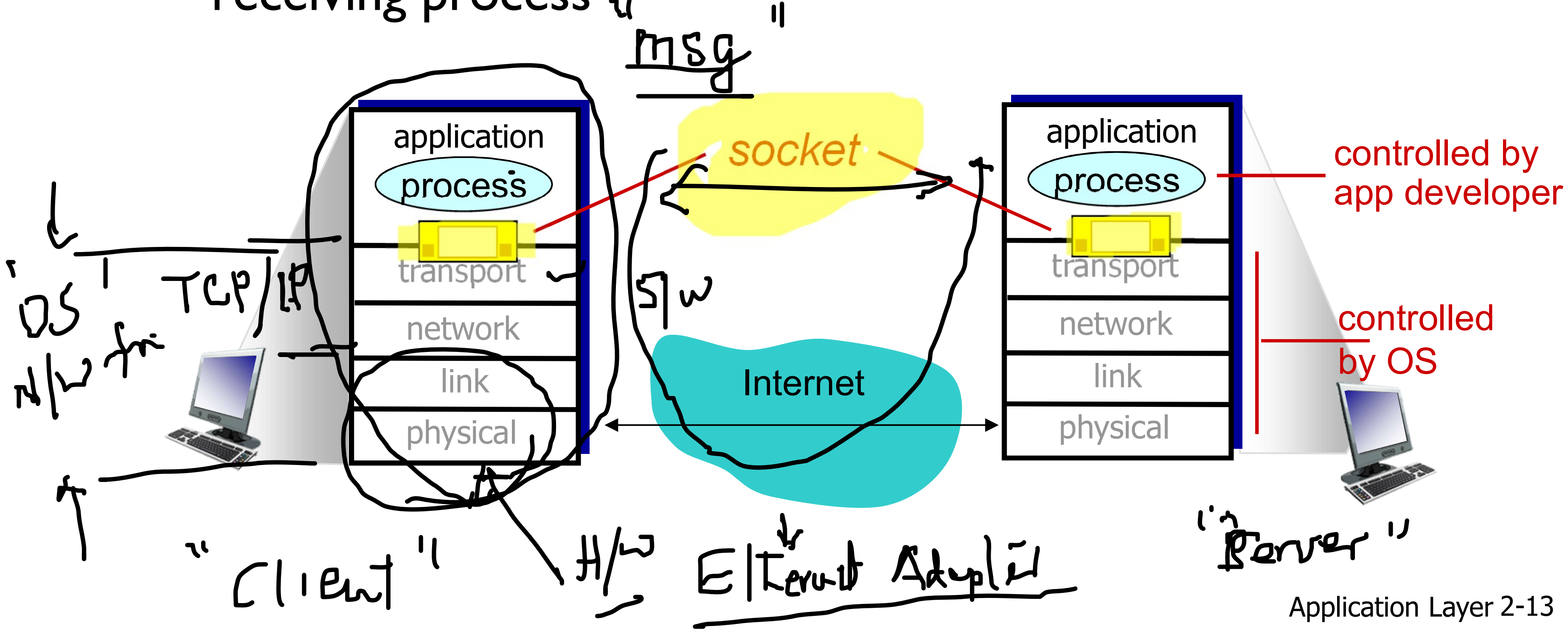
client process: process that initiates communication

server process: process that waits to be contacted

- aside: applications with P2P architectures have client processes & server processes

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



Addressing 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 gaia.cs.umass.edu web server:
  - IP address: 128.119.245.12
  - port number: 80
- more shortly…

DNS ↓ human convenient

addr: → www.daiict.ac.in.

32 bit

128.120.4.7 (IPv4)

0 – 255

port number

→ www.daiict.ac.in:8080

port

~ 10^{40} → 128 bit addr. (IPv6)

total # of addr ~ 2^{32} ≈ 4 billion ≈ 4×10^9

Curr Network devices ≳ 50 billion.

○ FTP (file Transfer Protocol) Appl.

'Msg.' →

Common set of rules

→ "msg"

→ "lang"

Client:

msg: get ␣ course_list.students → → SYNTAX.

"filename"

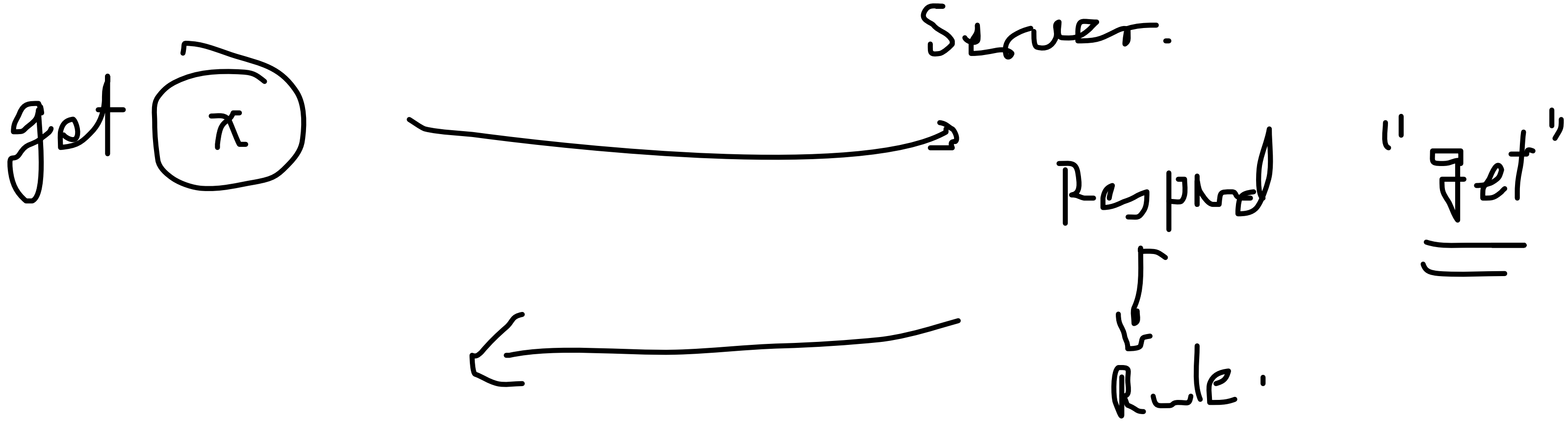
+ semantics.

↳ asking for/download

↳ grammar.

no of arguments

Command, separator, parameters/arg.



Search if 'x' exist.

if yes, "send" file.

if no, error msg.

"Appl. protocol design"

App-layer protocol defines

- 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
- allows for interoperability
- e.g., HTTP, SMTP

proprietary protocols:

- e.g., Skype