Algorithms

(for Game Design)

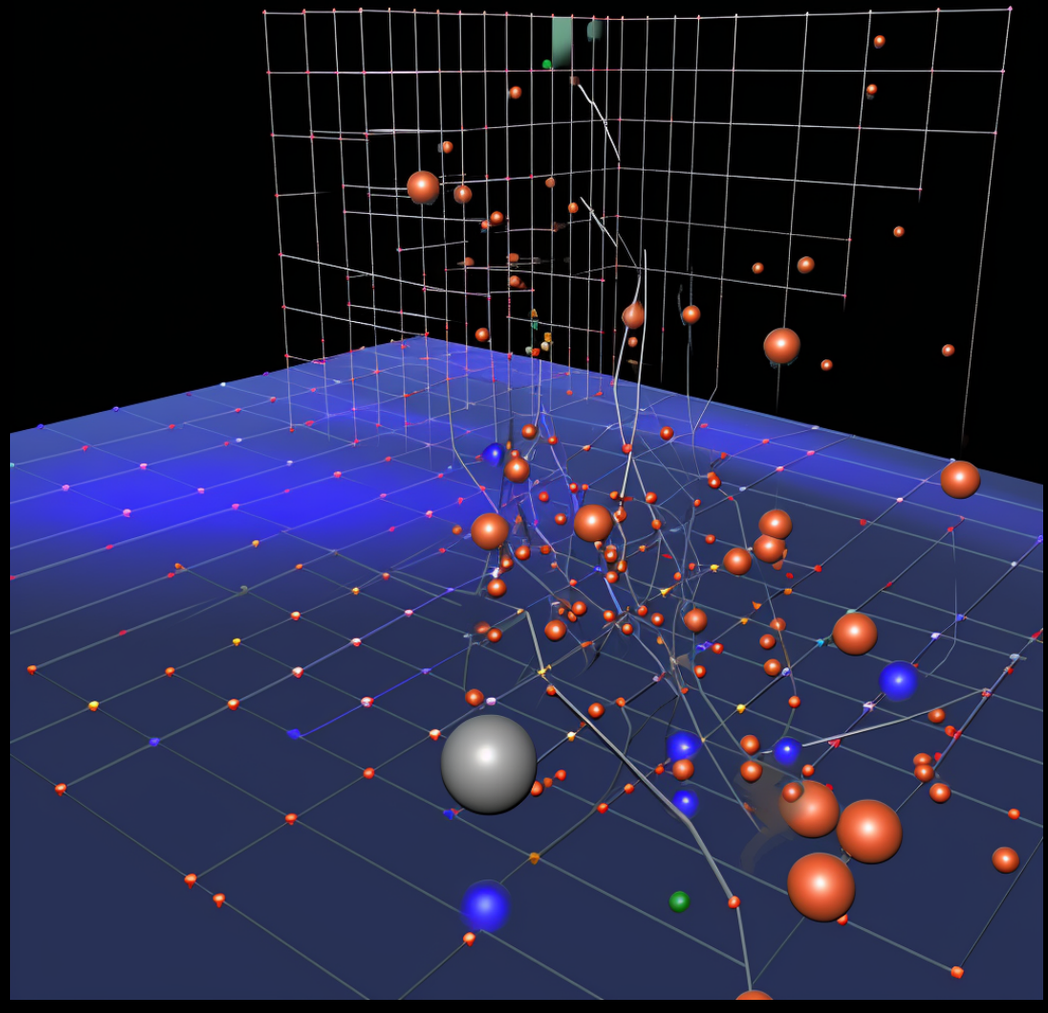
Session 6: Networked Games

Last time

- Collision Detection
- Problems with Naive Collision Detection
- Strategy for Collision Detection
- Many Tests

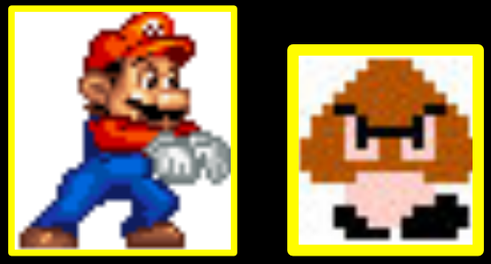
Collision Detection

- Also known as physics simulation
- Determining if/how objects interact
- Based solely on mathematical notations about those objects



Abstraction!

- What if we just check if their rects overlap?
  - Much simpler than creating the sets and calculating intersection



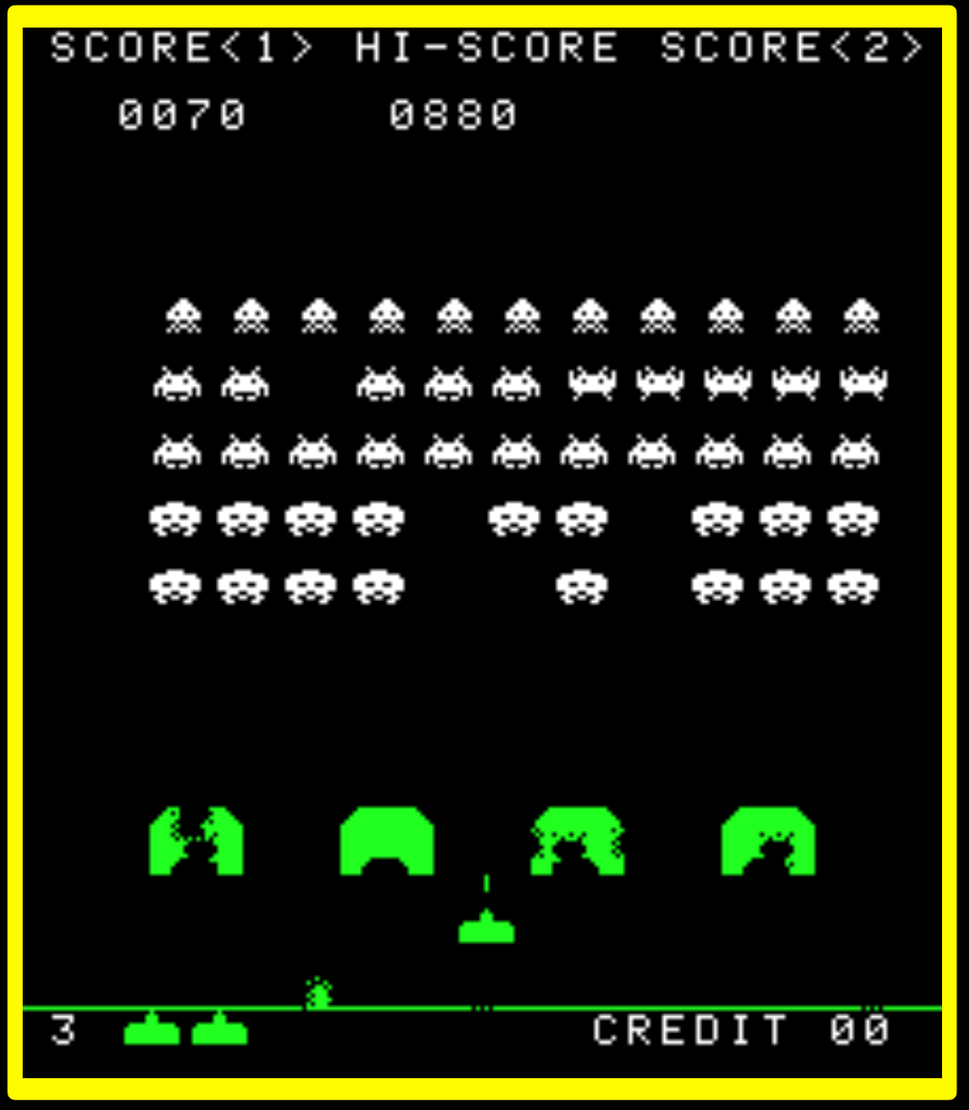
- Occasionally incorrect

Two Problems

- Tyranny of Numbers
  - Scale: Too many pairwise collision checks
- Move-through
  - Time is quantized by the game loop
  - Stuff might happen between frames

Historical Example

- Space Invaders
- 1978 Arcade Game
- Most money (infl. adj.)



Strategy

1. **Reduce** the number of checks that need to be made
    - This can be for *operational* reasons
    - Or because sprites are *geographically separated*
2. **Quickly test** sprites (pairwise): discard those that can't possibly collide
    - A very quick, but not very precise test
3. **Thoroughly test** remaining sprites to detect collisions
    - An in-depth test that is very precise
4. **Resolve collisions:** Do whatever -- explosions, bounces, etc.

Many Tests

- Circle Test
- Bounding Boxes: AABB, OOBB
- Capsules
- Polygon Tests
- Swept-spheres

Today

- Network Architecture
  - TCP vs UDP
- Network Programming
- Meshed Peer Topology
- Client-Server Topology
  - Client side prediction
- Starsiege: Tribes

We Like Networked Games

- Get to play against other humans
  - Maybe lots and lots of other humans
- Who aren't all at the same keyboard
- More social, more competitive!
- The very best AI still seems to be another human

Network Engineering

- Imperative that the game designer understands how networks operate
- Game requirements are often subtly different than other networked software

The Network is Great

- Worldwide collection of interconnected networks
- Any computer can communicate with any other

Network technology is fascinating!

... But complex

I encourage you to take a computer network class someday

Image credit: theconversation.com/how-the-internet-was-born-from-the-arpanet-to-the-internet-68072

What do we want?

- At the Application Level = Programmer's View
- All the data you send as a message will get instantly delivered to the other instances of your application, running on other player's computers
- A useful abstraction, but there are problems

Problem #1

- All the data you send as a **message** will get instantly delivered to the other instances of your application, running on other player's computers
- Message data is unstructured
  - The network just sends bytes ➙ no idea of its meaning
- You will have to serialize your object data into message bytes (and unserialize at the destination)
- Solutions: JSON, pickle (be careful!), others...

Problem #2

- All the data you send as a **message** will get **instantly** delivered to the other instances of your application, running on other player's computers
- The network has a limit to how fast it can transfer data
  - Bandwidth (or throughput)
  - Expressed in bits-per-second (bps)
    - Hopefully a big number, like 100Mbps
- If your game transfers lots of data, it is going to take a long time

Problem #3

- All the data you send as a message will get **instantly** delivered to the other instances of your application, running on other player's computers
- The network can't operate at faster-than-lightspeed rates
- *Latency* is the time to get a message to get from one computer to another
  - Latency is commonly in the 10-200 milli-second range
- Remember that 60 frames per second is 16.6 ms per frame
- Note that this time is independent of the bandwidth-related time from the previous item

Problem #4

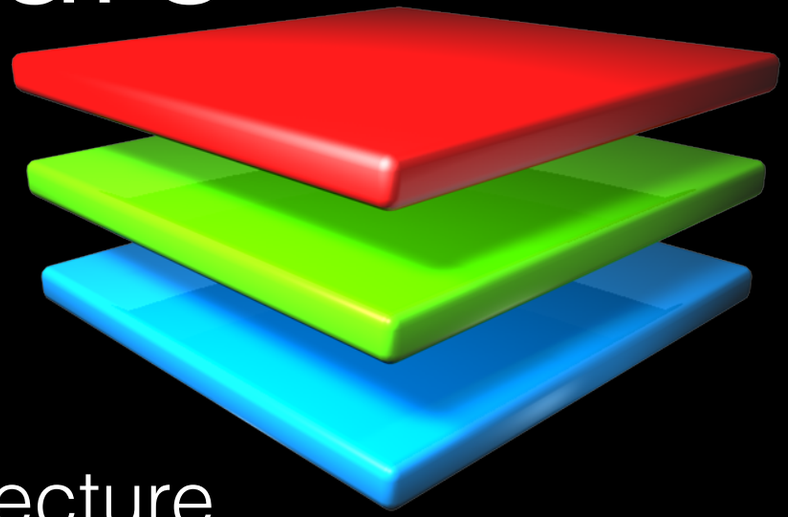
- **All the data** you send as a message will get instantly delivered to the other instances of your application, running on other player's computers
- The network is not reliable: Bits are flipped. Data is dropped, duplicated and re-ordered
    - Solutions exist: Error correction codes, protocols like TCP
    - Occasionally, the network partitions into two pieces that cannot communicate
        - No solution exists
        - Luckily, this is rare at large scale

Problem #5

- All the data you send as a message will get instantly delivered to the **other instances** of your application, running on other player's computers
- How does a sending computer know where other copies of the game are being played?
- Data sent from your game needs to know an *address* for the destination computer
- In some games, the user is asked to type in this address
- More commonly, a central service maps usernames to computer addresses

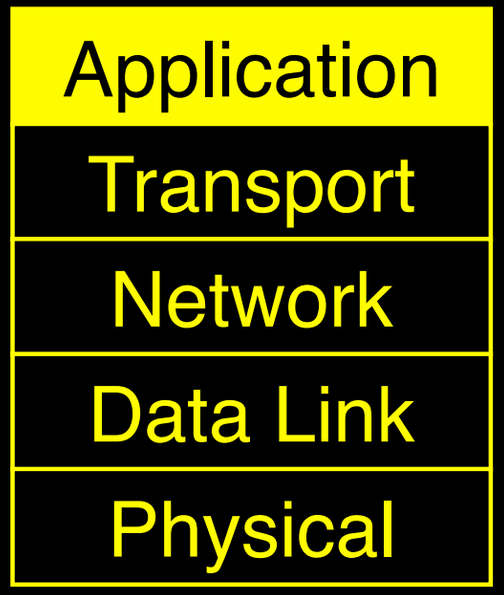
network architecture

Architecture



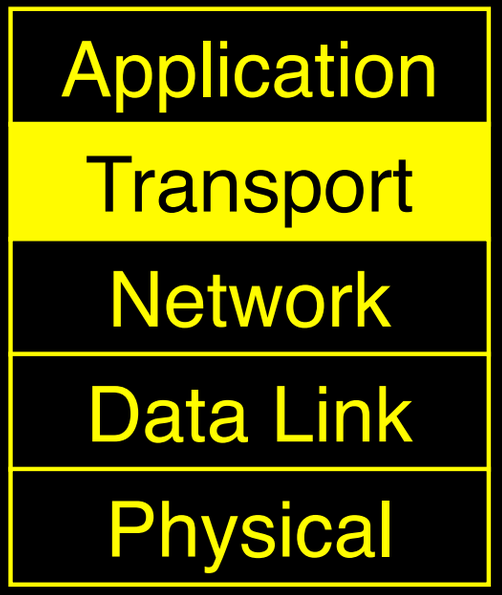
- How to arrange components in a complex system
- Network uses a **layered** architecture
  - Components are stacked
  - Each component uses the services of the layer underneath...
  - ...to provide service to the layer above

Application Layer



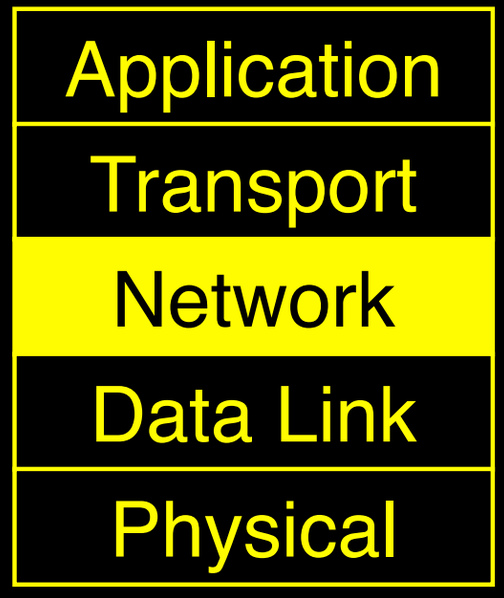
- Where your game resides. It is a networked application, providing network access to the user
- Mission: Provide network access to users of various network-enabled applications
- Datatype: Message
- Protocols: Vary widely
  - HTTP for web traffic, TLS for security, DNS for naming services
  - Plus, your protocol to communicate from player to player

Transport Layer



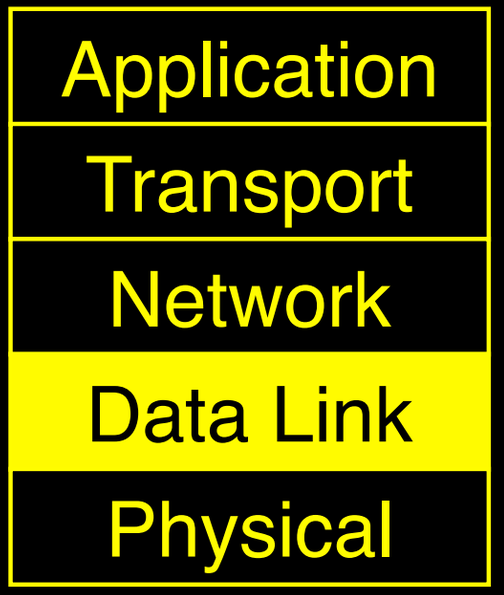
- Mission: Connect applications with a logical connection
  - This means your game can use the transport layer to send messages to other players
- Datatype: Segment
- Protocols: TCP and UDP are common

Network Layer



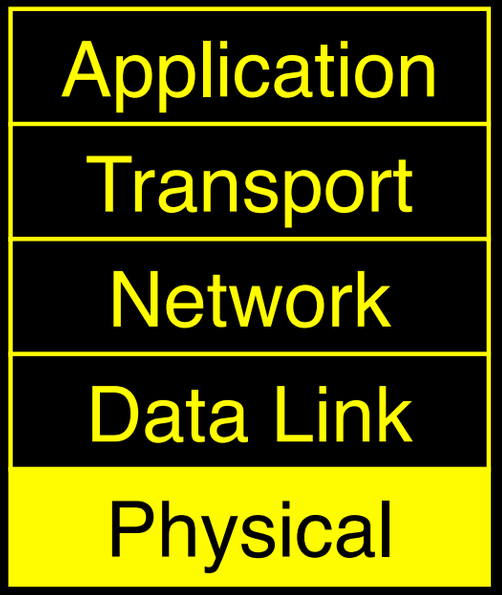
- Mission: Connect computers in the network
  - Responsible for getting data from any computer to any other computer
  - Operates on all the routers and end-hosts of the internet
- Datatype: Packet
- Protocols: Internet Protocol (IPv4, IPv6) for data delivery
  - Routing protocols include: BGP, RIP, ISIS, OSPF

Data Link Layer



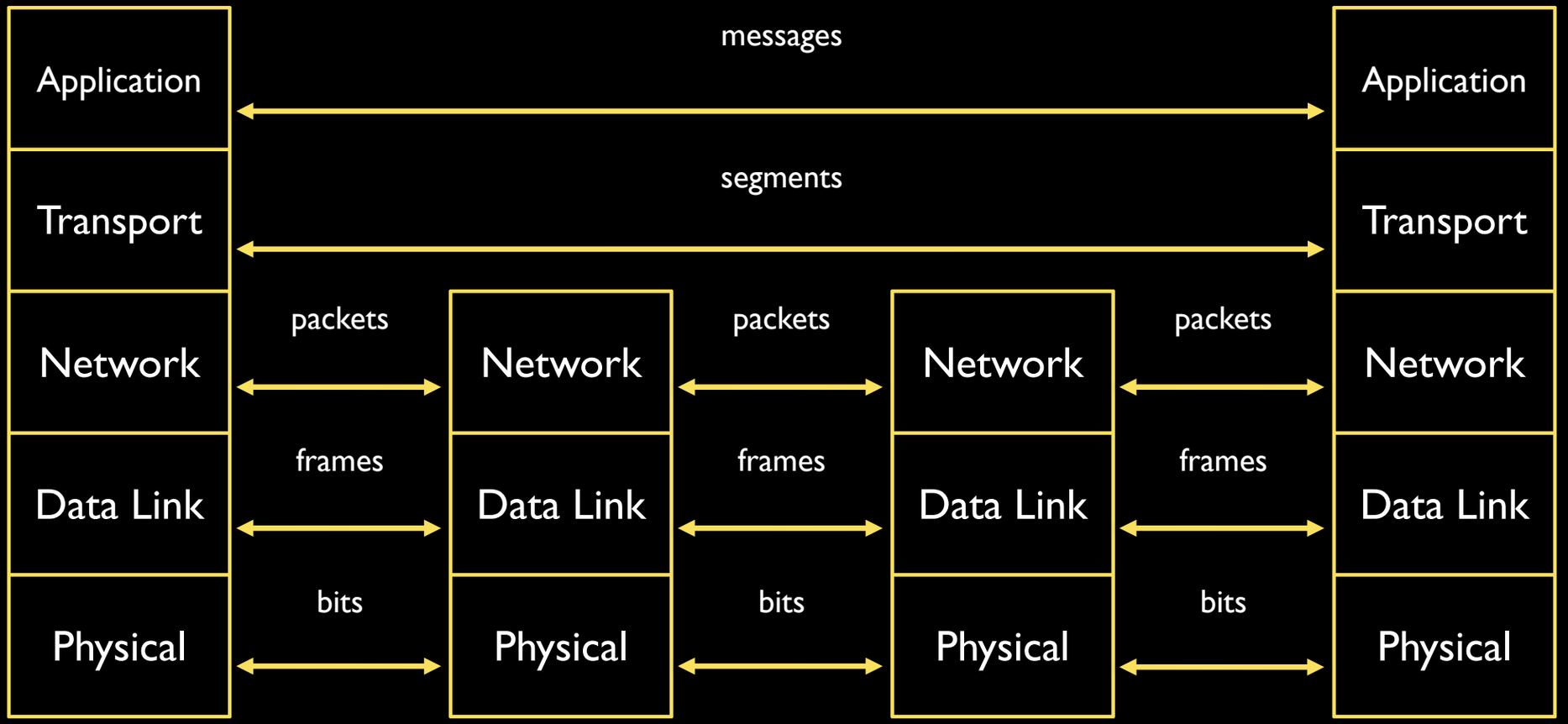
- Mission: Connect two adjacent computers or routers
- Datatype: Frame
- Protocols: Ethernet, WiFi, others
    - Other technologies lead to other protocols

Physical Layer



- Mission: Move bits from one end of a wire / radio channel / media, to the other end
- Datatype: Bit
- Protocols: Various modulation strategies
- Note: This is a very electro-magnetic oriented field, and is dominated by communication and electrical engineers

All Together!



Whew!

- Lots of names and acronyms
- Game developers are mostly be concerned with the interface from the application layer to the protocols of the transport layer
- A fuller understanding of the network stack will help to decipher the phraseology of the documentation and debug problems

TCP vs UDP

UDP: User Datagram Protocol

- A thin layer over top of IP (Network Layer), with few additional services
- Each UDP segment is designed to be transmitted in a single IP packet
  - IP packet goes to a particular computer
  - IP address: something like 128.2.3.103 or fe80::208f:faff:fe0d:a177
- UDP segment goes to a particular application
  - UDP Port number: something like 51987
  - 65000 available
  - <1024 are "well-known" ports

UDP Guarantees

- There are no guarantees ➙ only "best effort"
- You can lose your segment (maybe 1-5% chance, though bursts happen)
- Segment can be duplicated
- Segment can be delivered out of order

Checksum

- UDP (and TCP) use a checksum to combat bit errors
- Checksum is an "error detection" mechanism
  - Any single-bit error is detected
  - Many multi-bit errors are detected (but not all)
- Statistically, this may not be enough protection
- Encrypting your data helps detect additional bit errors

TCP: Transmission Control Protocol

- TCP does everything UDP does (checksums, port addressing, segmenting...)
- Also adds reliability guarantees
  - Protection against loss, out-of-order delivery, or duplication
- Sounds great! Why would anyone not want this?

TCP Reliability

- TCP works by adding a feedback mechanism
- Every segment that is sent has a sequence number attached
- Receiver sends back an "ACK" with that sequence number*
- If sender doesn't get an ACK "in time," it assumes a loss and re-transmits the segment

Sounds great!

- Yeah, but not necessarily for all apps (and especially not for games)
- On the receiver side, the sequence numbers are used to ensure all the segments are collected
  - Duplicates are discarded
  - Segments put in the correct order
  - Wait until lost segments are re-transmitted

Here's the problem!

- Then, deliver the data to the appilcation -- in the correct order

Reliability through Delay

- TCP creates reliability, but uses delay to achieve it
  - Delay is almost never the right choice for real-time, action-oriented games
- Each segment includes data about player actions: locations, movement, swinging swords, shooting, opening doors, ...
- If there is a loss, TCP will wait until that data is re-transmitted and received (BTW, additional losses can happen)
- When that data is received, it is often out-of-date

TCP Conclusion

- Don't use TCP for action-oriented, real-time games
  - At least, be careful about the data you send using it
  - TCP Is great for bulk transfer of static data
- It may be fine for a board game, but not for Fortnite
- This is a fundamental feature of TCP ➔ you won't be able to program around it

other problems

NAT: Network Address Translation

- Game users in residential settings are almost certainly behind a Network Address Translation (NAT) device
- The world is running out of IPv4 addresses, so most homes share a single address for all computers
- This strategy works beautifully for web surfing and email
  - The desire to connect starts within the home
- But, with games, the desire to connect starts on your friend's computer at his house

NAT Solutions

- STUN: Session Traversal Utilities for NAT
  - RFC 5389: tools.ietf.org/html/rfc5389
- ICE: Interactive Connectivity Establishment
  - RFC 8445: tools.ietf.org/html/rfc8445
- TURN: Traversal Using Relays around NAT
  - RFC 5766: tools.ietf.org/html/rfc5766

Security

- Internet connected games open up opportunities for those with malicious hearts (cheaters and otherwise)
- Make sure your game validates that all incoming segments actually were sent by other game users
- Make sure your data is encrypted, so it can't be read in transit
- Make sure you don't allow your games to send data to those who haven't asked for it

Important! Security!

- If you are building a game for wide-scale release,
- Get someone who knows internet security on your team
- Hopefully very early in your development process

network programming

BSD Sockets

- An abstraction of the network, used in many OSes and languages (including Python)
- The socket API provides functions to connect, send/receive data, disconnect
    - An abstraction: Hides lots of the complexity of using DNS, TCP and UDP

Server Side

- A server will `open( )` a "listening" connection to the socket API
  - Lets the socket library set up some state
  - Tells the socket library if you want to use TCP or UDP
- Then, it will `bind( )` to a particular port number
  - Port numbers are Transport Layer addresses
- Then, it will call `listen( )` which sets aside buffers for connections
- Then, it will call `accept( )` which won't return until a client connects

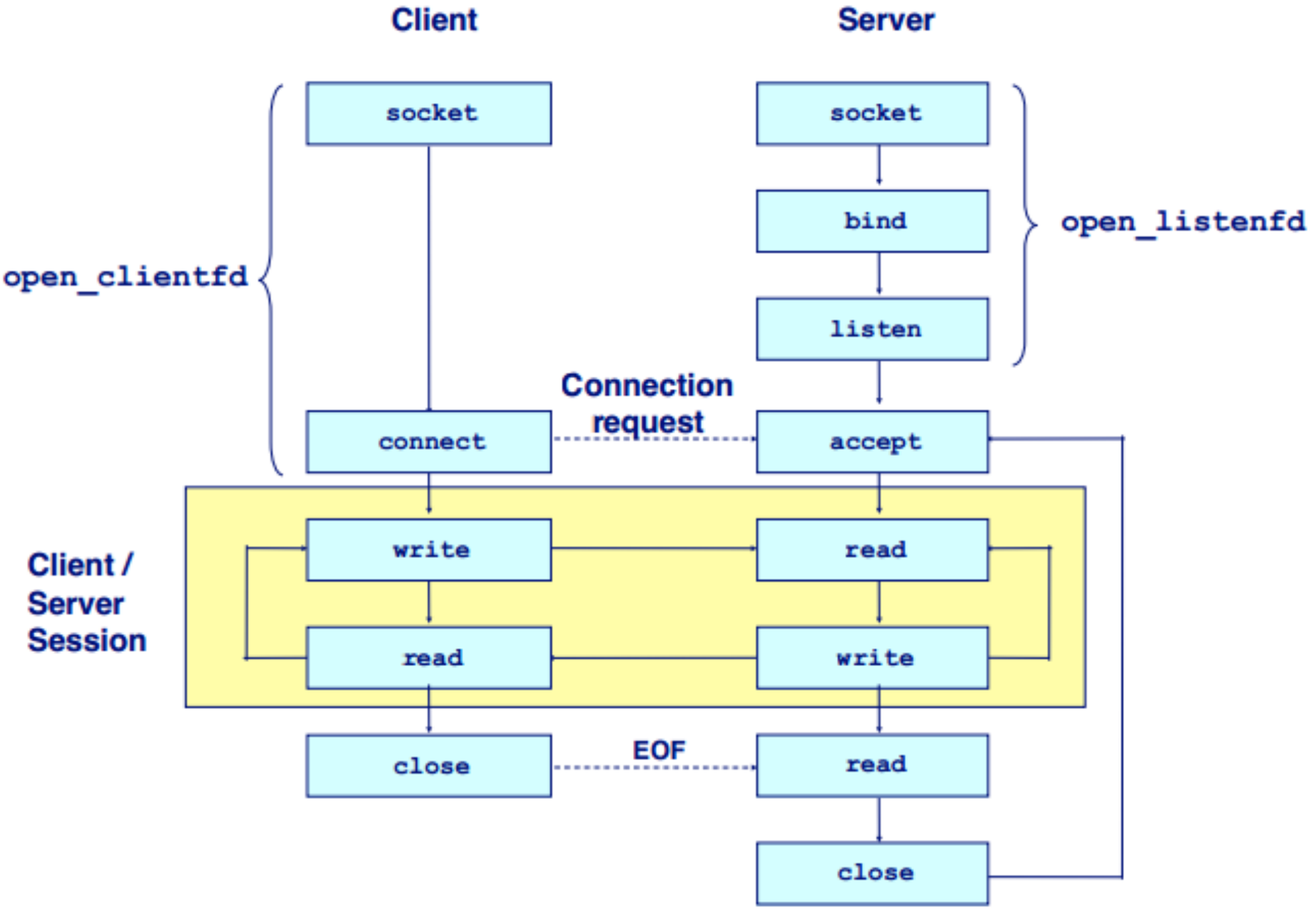
Client Side

- Client `open( )`s the connection to the socket API
- Client calls `connect( )` to actually start sending segments to a target server
  - Must specify an address and port number
- If connect is successful, the client and server can write / read data over the connection
  - Use `send( )` and `recv( )`

Finishing Up

- The connection is bi-directional
- Either side can `close( )` the connection whenever they have no further data to send
- When both sides have called `close( )`, the connection is ended
    - Also, some timeouts force a close if one side is silent for too long

Socket API Operation Overview



webclient_socket.py

```
#!/usr/bin/env python3
import sys, socket

# hard-wire the port number for safety's sake
# host and filenames are on the command line
port = 80
host = sys.argv[1]
filename = sys.argv[2]

FORMAT = "---------------\n{}\n---------------"
```

```
# create a TCP/IP socket object
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect((host, port))

# Send an HTTP GET request
request = f"GET {filename} HTTP/1.1\nHost: {host}\n\n"
print("\nSending an HTTP Request\n")
print(FORMAT.format(request))
totalsent = 0
while totalsent < len(request):
    bytes_array = request[totalsent:].encode('utf-8')
    sent = sock.send(bytes_array)
    if sent == 0:
        print("connection broken during send")
        sys.exit(-1)
    totalsent += sent
```

I want an IP and TCP socket

Use that socket to connect

send some bytes

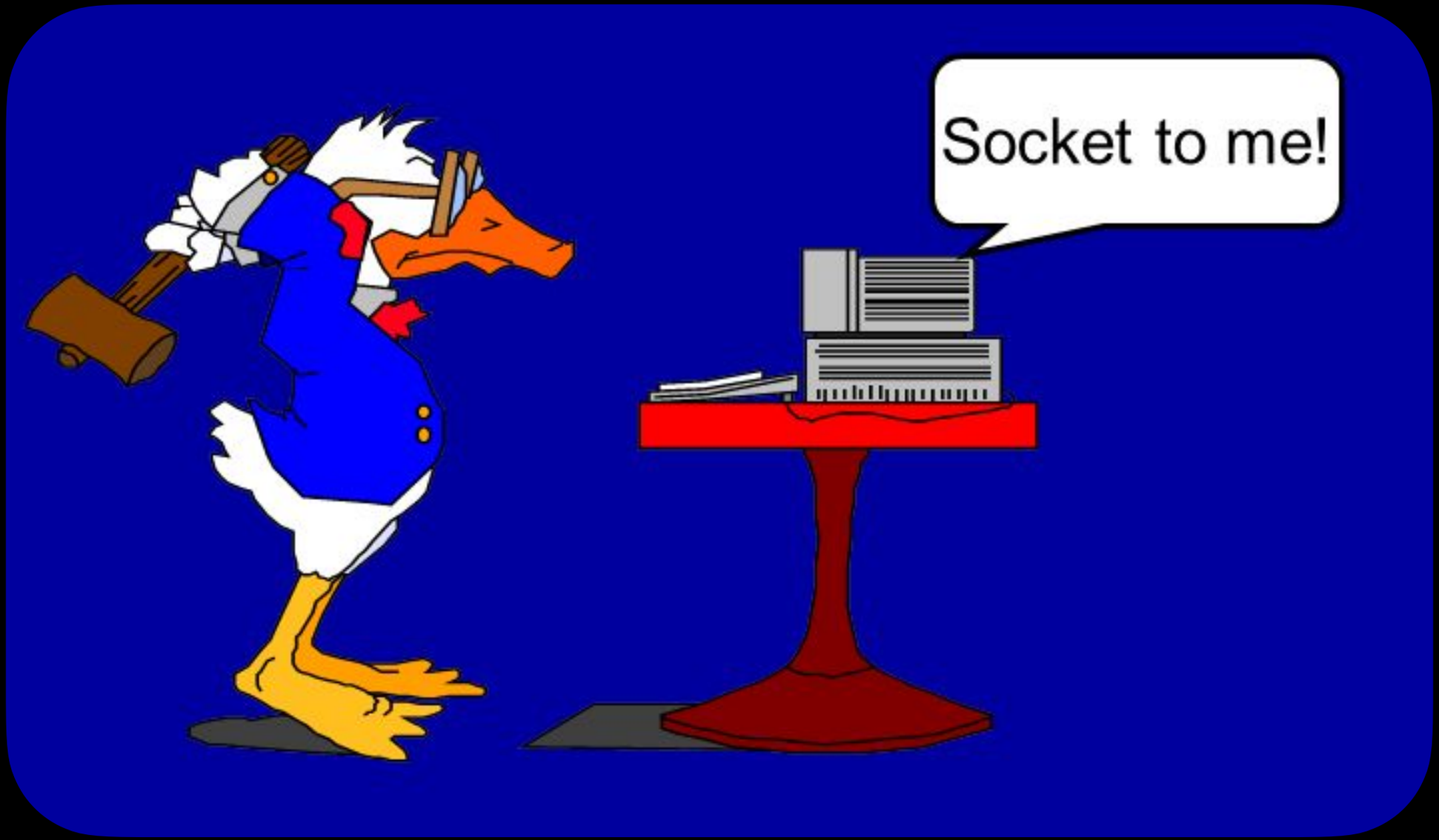
```
# Read and print out the response
print("\nReading the response\n")
chunks = []
while True:
    chunk = sock.recv(4096)
    if chunk == b'':
        break
    chunks.append(chunk)
byte_array = b''.join(chunks)
response = byte_array.decode('utf-8')
print(FORMAT.format(response))

sock.close()
```

receive up to 4096 bytes

close up shop!

The Verdict

- Sockets are conceptually easy, but numerous error possibilities make them hard to operate in practice
- Also, no assistance is provided for the transfer of structured information


Socket to me!

Sockets??

- Numerous libraries and frameworks exist to make network programming more robust and less complex
- Sockets are still the foundation of those libraries, powering almost all network interactions

Other Frameworks

- Features you might want:

1. A way to discover players on a local network automatically
2. A way to connect to players or servers outside your network
3. A way to punch through NAT devices (and, perhaps, firewalls)
4. Serialization of network data
5. Efficient transmission of network data in sizes that make sense for your game.
6. Concurrency controls: Network I/O is asynchronous and you don't want your game code waiting while the network is checked.
7. Error handling

Look at: Twisted, RakNet, asyncio

socket example

An Example

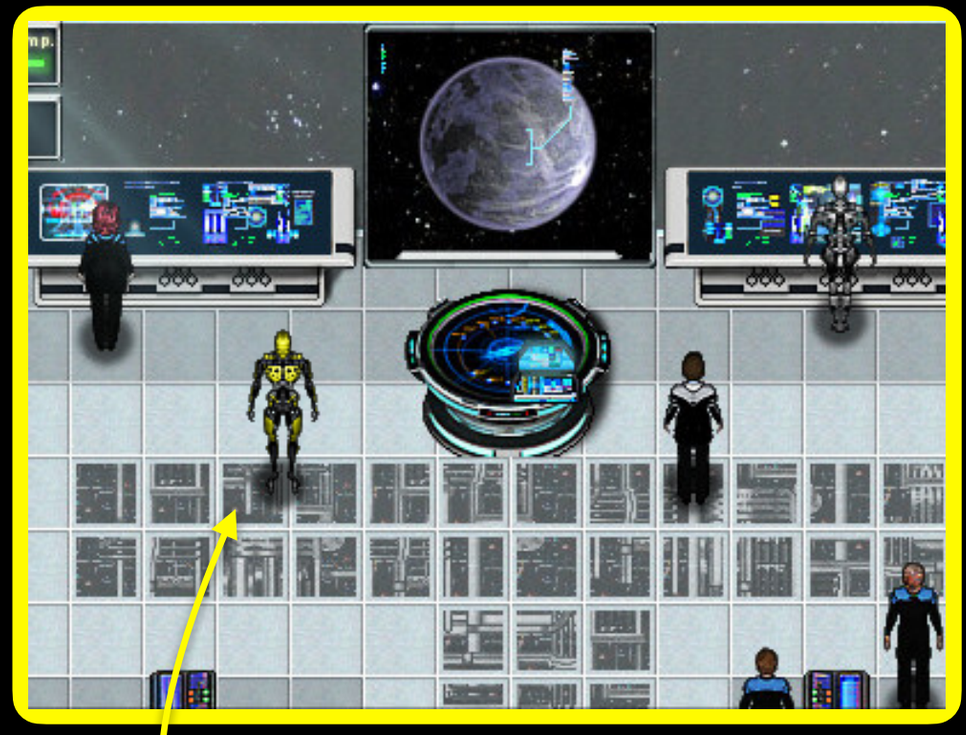
- The web is littered with examples of socket-based communicating code examples
  - Many are an "echo server" that will send any incoming data back out
  - Few are game oriented or have the game-loop problem
- My example in `UDPClient.py` and `UDPServer.py` uses Pygame and sockets

Execution: Server

- Run `UDPServer.py` first
  - It is terminal based, non-Pygame
- Server keeps track of location of all clients
  - Responds to any movement with an updated list of all client locations

Execution: Client

- Client will show a sci-fi background
- Each client will have an android avatar
  - Move with arrow keys
- Run the clients on the same computer as the server



Network Protocol

- When an arrow key is pressed, client sends a UDP segment with "u" followed by the direction

```
conn.sendto(b"ur", addr_port) #right
```

- Notice the odd string notation -- it's converting a string into a "byte-array" because the network doesn't know anything about strings

Network Protocol: Server

- Whenever a move is received, send a list of locations to all clients
  - Put the list in a string with | separators

```
"x0,y0|x1,y1|x2,y2..."
```

- Then convert the string to a byte-array with

```
the_string.encode('utf-8')
```

Unconvert

- When the client gets the string (with a list of locations and | characters)

```
msg, addr = f.recvfrom(32)
```

- It must be un-encoded from a byte-array

```
str(msg, 'utf-8')
```

- Then lots of parsing to get the data back into a list, get the coordinates out as integers, etc

Other Ugliness

- The IP address of the client and server are hard-coded into the source ➙ Bad, bad practice
  - Ooops, the port numbers are hard-coded as well
- The IP address is `127.0.0.1`, which is known as the "loopback" address
  - This is always the IP address of the local computer
  - You can easily send segments to yourself

Yuck!

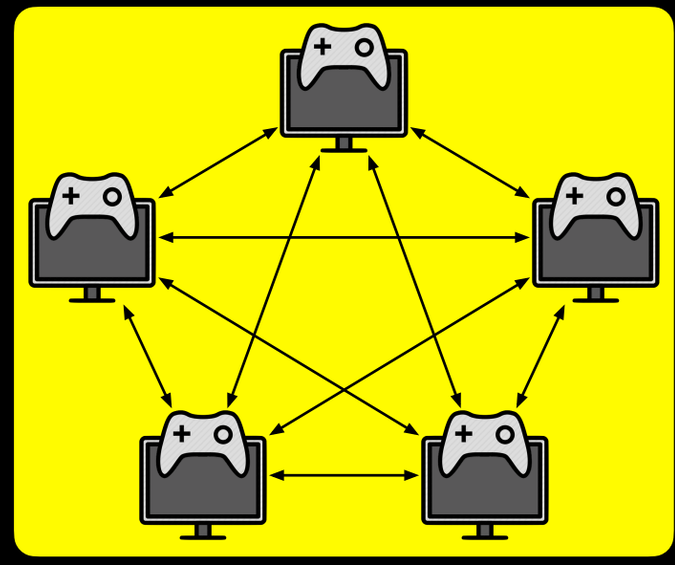
- This code looks like ..... Ugh. Yuck.
- You don't want this sprinkled all around your game
- That's why you'll need to use a framework or library (or, build your own methods to send the game-specific messages you desire)

meshed-peer topology

Clients and Servers and Peers

- Each application of the game sends messages through a connection
- What is the *topology* of these connections?
- Remember, each line represents many routers moving data along that connection

Meshed Peer Topology



- Early days! Each player sent and received data from all other players
- Peer-to-Peer Lockstep: Game play happens in a series of turns
- In each turn, all peers send player input to all other peers ➙ sequence of commands
- Each player's machine executes all the commands in the same order
  - Identical outcomes on all computers!

Problem 1: Latency

- Latency! It takes time to send messages
- Before processing the turn, each peer must collect all the commands from all other peers
- Any peer that is late will slow down everyone
- The game goes as slow as the slowest network connection ➙ **lag**

Problem 2: Determinism

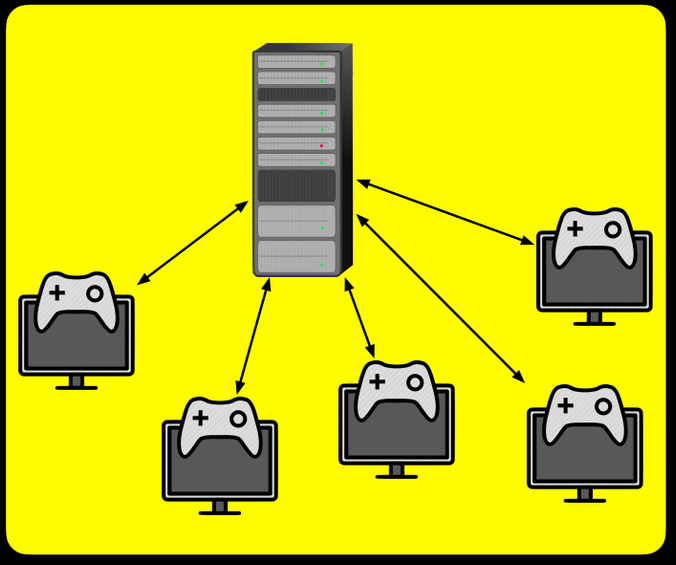
- More subtle: It is very difficult to ensure every machine does exactly the same thing with the commands
- Differences build up to complete desynchronization over time
- Great example from Age of Empires:
- "1500 Archers on a 28.8" by Bettner and Terrano

Authority

- The real problem with meshed-peer is a lack of authority
- There is no single source of truth
- Authority is distributed and assumed to be synchronized
- BTW: Cheating is harder to deal with as well

client-server topology

Client-Server



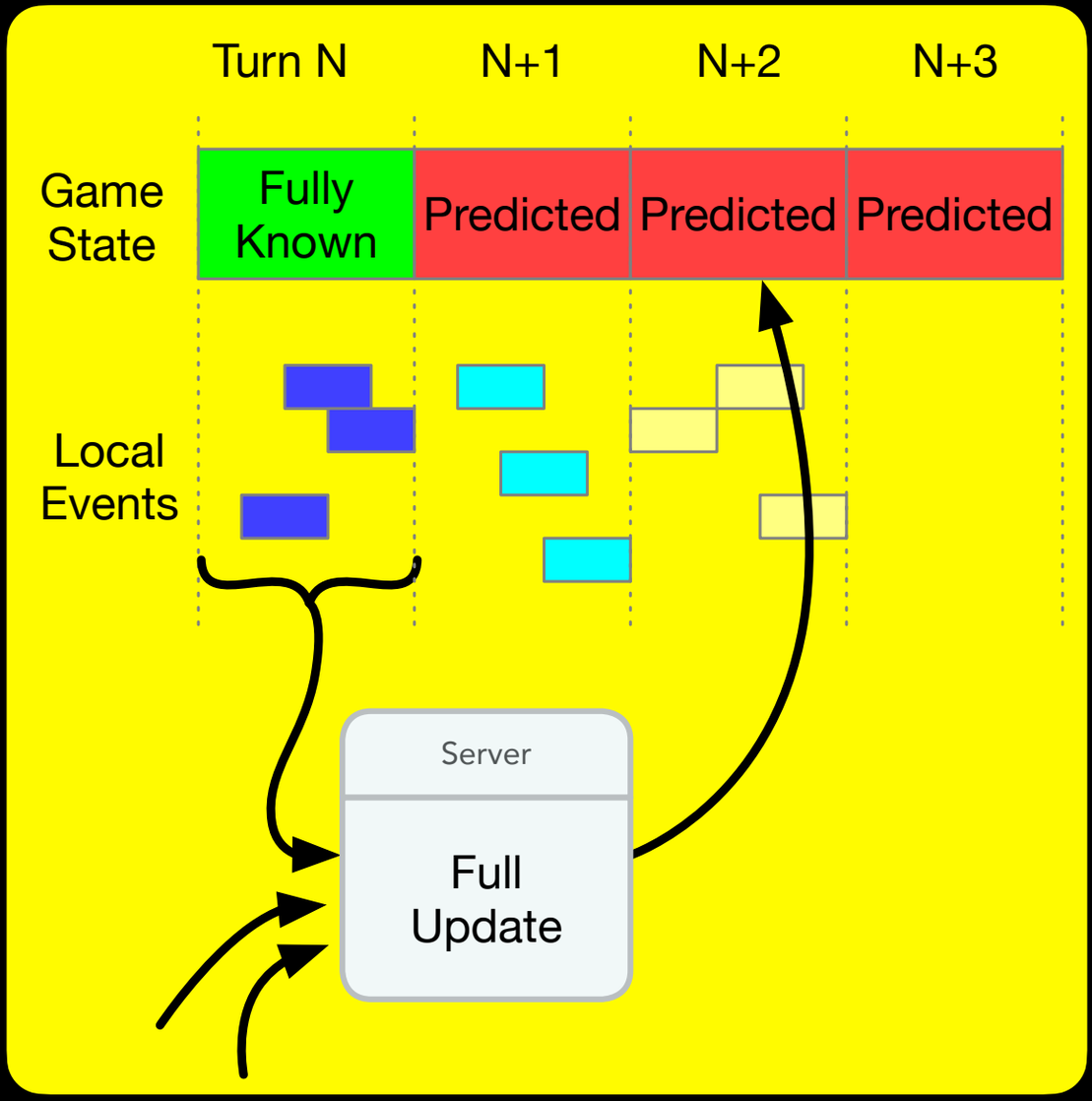
- Player's computers are *clients*
- Connections are to a single *server*
  - Often operated by the game company and in a data center
- Server is **authoritative**: knows the entire game state
- Clients send player inputs to server
  - Which sends back enough info to render the frame

- Latency is no longer a problem, but must be managed
- A slow player will not hold all other players up
- Originally, key and mouse events were sent to the server
  - 1996: John Carmak and Quake
  - But, latency is visible ➙ RTT is > human perception
- When a player hits a key to fire a gun, the gun doesn't actually fire until after the server hears about it and sends a response back to the client

Client-side Prediction

- Solution to the latency issue
- Client will predict the effects of the player's action
  - And, show those predictions to the player
- But, other player's actions could change the game state
- Then, the predictions aren't correct

- Solution: Client keeps track of all inputs and when they happened
- When an update comes from the server, the events are replayed against the past game state
- Known as **Deterministic lockstep**



Starsiege: Tribes

History Lesson

- A pioneering networked game
- Networking architecture is still relevant today
- Sci-Fi FPS from 1998

Network

- Tribes had fast-paced combat system
- Large number of players (up to 128)
  - Exceptionally good for the time
- LAN play or internet play
- 28.8 Kbps with several hundreds of ms latency

Ouch! Today's numbers are 100Mbps at tens of ms

Delivery Guarantees?

- Built on unreliable data networking (like UDP)
- Designer's built application protocols with some guarantees

1. **Non-guaranteed data:** This is data that would never be retransmitted, should it be lost in the first place
    - In fact, if the game is bandwidth-starved, it could choose not to even bother sending this data in the first place

Sounds like UDP

Sounds like TCP

2. **Guaranteed data:** This data must be retransmitted if lost and delivered to the receiver in order

   - Used for the most critical data in the game

3. **Guaranteed Quickest data:** This data is the highest priority stuff and will be transmitted with a delivery guarantee

   - Same as category 2, except it is higher priority and should go before #2 data

   - Usually, this is data which is only relevant for a short window and needs to get to everyone quickly

4. **Most Recent State data:** This data is about some state, wherein only the latest value is of importance

   - If the message is lost, then it shouldn't be re-transmitted
   - Instead, the newest value of that state should be sent
   - The location of a player 10 seconds ago is not as important as its current location

- "The TRIBES Engine Networking Model or How to Make the Internet Rock for Multiplayer Games" by Frohnmayer and Gift
- Goes into more detail
- Includes the networking architecture (layers)
- Great discussion of "Ghosting"
  - How to distribute hints of state information

What did you learn today?

- Computer networks are complex
- Using them in games is challenging
- Don't be scared: Computer Scientists have studied how to build distributed systems for 60 years
- Do be careful: Build on their knowledge with careful study. Don't just hack stuff