Final Review

FRIDAY
12/15 3:00PM
MICHELANGELO

Final Exam

- Same format as before: about equal parts **vocabulary**, short-answer **concepts**, and long-response **application** of ideas.
- One 8.5"x11" *handwritten* page of notes
- About 66% from post-midterm, 33% pre-
- About 66% from lectures, 33% from reading assignments (entry tickets)
  -
  - 24% of final grade

“

Threat modeling is a process by which a system is methodically analyzed from an attacker’s perspective, to identify attack goals, evaluate the risks they pose and mitigate their vulnerabilities.

Zero-Days and “Hats”

- **Zero-days** are security vulnerabilities that have not yet been revealed: Vendors have had “zero days” to mitigate them.
- **Hackers** are just people who use systems in unintended ways. The term is neither good nor bad.
- **White hats** disclose zero-days to vendors before making them public.
- **Black hats** keep zero-days secret to exploit them themselves.
- **Gray hats** make zero-days public.

Threat Actors

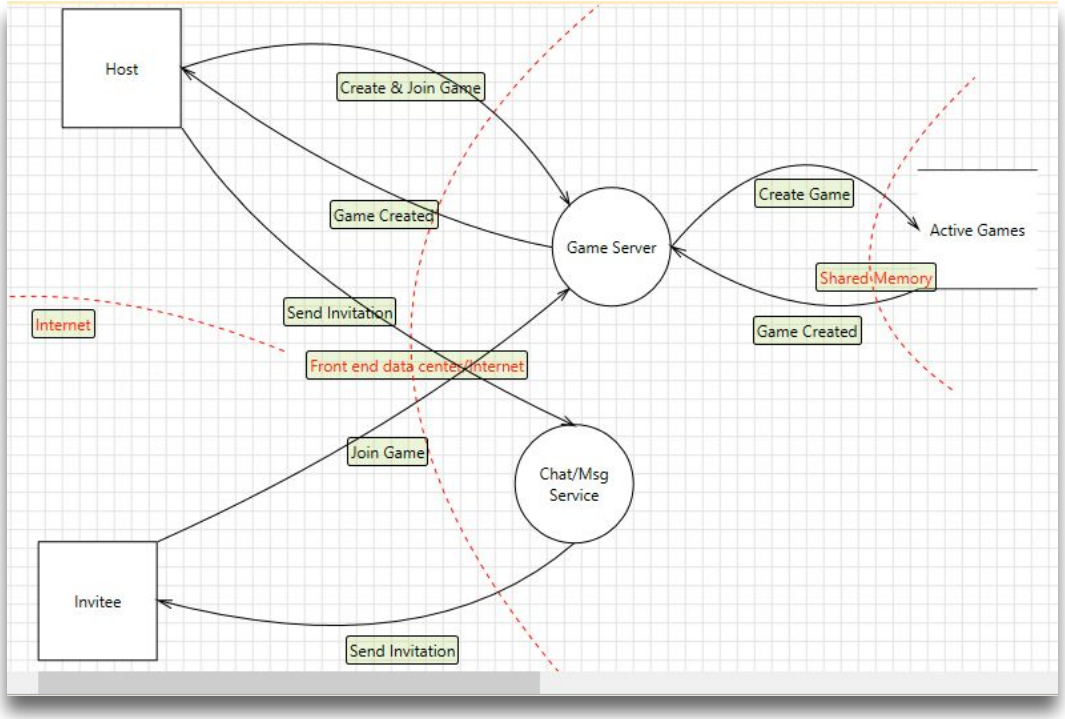
- **State actors** have government (i.e. “infinite”) resources, but varying motives.
- **Organized crime** have resources proportional to the economic value of the attack.
- **Insiders** have access to more information, but may be defeated by anti-repudiation measures.
- **Hacktivists** have diffuse resources but varying skill and motivation.
- **Script kiddies** are plentiful but have low skill..

STRIDE

- **Spoofing**: Pretending to be another user
- **Tampering**: Modifying data outside of normal usage
- **Repudiation**: Erasing the history of an action
- **Information disclosure**: Reading secrets
- **Denial of service**: Preventing normal operation
- **Elevation of privilege**: Performing forbidden actions

Data Flow Diagrams

- **Threats come from data**, so to create a threat model we must document what our data is, where it comes from and goes to, and how it is handled along the way.



Data Flow Diagrams

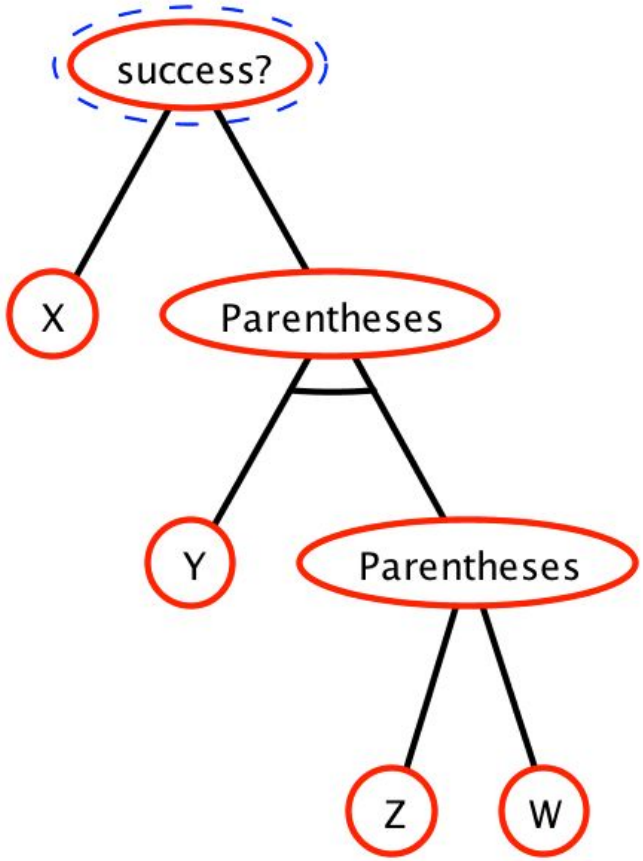
- **Processes**: Code (not an OS process)
- **External Interactors**: A source or sink of data that's outside your control (e.g., the client)
- **Data Stores**: Something that holds data—memory, a file, a database
- **Data Flow**: The transfer of data from one element to another
- **Trust Boundary**: Border between two elements that do not trust each other

STRIDE x Elements

	S	T	R	I	D	E
Processes	√	√		√	√	√
External Interactors	√					
Data Stores	√	√		√	√	
Data Flows		√	√	√	√	

Attack Trees

- Answers the question "What has to be true for an attacker to successfully perform this attack?"
- Root node is the attack itself
- Siblings connected with an arc must all be true (AND)
- Siblings with no arc just need one to be true (OR)



Risk Rating

- **Perfect security is impossible**, so we aim for “good enough”. Making that decision wisely requires balancing costs versus benefits.
- **Risk = Likelihood x Cost**

REAL Rating:

- **Reward**: What’s it worth to the attacker?
- **Effort**: How little does the attacker have to work?
- **Audience**: How many people will be affected?
- **Level of Skill**: How many attackers have the skill required to carry out the attack.

Mitigations

- A **vulnerability** is where an attacker has one or more paths from the bottom of an attack tree up to the top.
- A **mitigation** blocks a node, preventing the attacker from making progress.
- A tree is **fully mitigated** if there are no paths from the bottom to the top.
- A tree has **defense in depth** if it is still fully mitigated even when one or more mitigations are removed.

Mitigation Toolbox

- **STRIDE categories imply mitigations**:
- Spoofing —> **authentication**
- Tampering —> **signing**
- Repudiation —> **auditing**
- Information disclosure —> **encryption**
- Denial of service —> **auditing & filtering**
- Elevation of privilege —> **defensive coding**

Auditing

- **Keeping a log** of all important activity, so that it can be reviewed after an incident
- To be effective against repudiation, auditing requires authenticating users
- To be effective against denial of service, either requires authenticating users or accepting the risk that some innocent users will be punished

Filtering

- **Rejecting invalid or malicious input as quickly as possible**.
- Can disclose information when done naively.
- Can become a denial of service vector against legitimate users when done without authentication.

Authentication

- An entity can be authenticated by **something it is**, **something it has** or **something it knows**
- In information security, most often we rely on the last. In other words, the entity must demonstrate possession of a **shared secret**.

Hashes

- A **hash function** reduces an arbitrary message to a fixed-size representation
- A **cryptographic hash** is one that can't* be reversed; where the message cannot* be modified without changing the hash; and where a **collision** cannot* be created.
- "Cannot" in cryptography means "cannot in less than **brute-force time**"
- Sometimes an algorithm gets **broken**, meaning that less-than-brute-force attacks against it have been discovered [for example, MD5]

Hash Algorithms

- **MD5** [**broken**, but still in use], **SHA1** (128 bits), **SHA256** (256 bits), **SHA512**
- Take blocks of data and perform simple bit operations—shifts, rotates, XORs—on them repeatedly
- Around 300,000,000 blocks a second on modern CPUs… 100 times that on GPUs.
- As computing power increases, hash algorithms must become more complex
- Collisions can be found in square-root time thanks to the **birthday paradox**

Signing

- A **signature** seeks to prove that a document has not been **tampered with** and came from an **authentic** source
- A cryptographic hash algorithm can prevent tampering
- Including a shared secret in the hash can prove authenticity

Encryption

- An **encryption algorithm** transforms a document so that it cannot* be read without the **encryption key** (a shared secret)
- In a **symmetric** algorithm, one key both encrypts and decrypts
- In an **asymmetric** algorithm, there are two keys: What one key encrypts, the other decrypts (and vice versa, usually)
- "Cannot" in cryptography again means "cannot in less than **brute-force time**"

Symmetric Encryption

- **DES**, **3DES**, **AES**
- Take blocks of data, XOR them against a key, mutate the block and key and XOR again, repeatedly
- Around 200,000,000 blocks per second on modern hardware
- The major challenge of symmetric encryption is **key distribution**—communicating the shared secret of the key between the parties of the conversation

Asymmetric Encryption

- **RSA** or **elliptic curve**
- Vastly (100,000x) slower than symmetric encryption, and only suited to small messages
- Primarily used for **key exchange** (generate a random symmetric key and then encrypt it asymmetrically) and **message authentication** (calculate a hash and then encrypt it asymmetrically)

Creating an Authentication Scheme

- Use a hash to demonstrate possession of a shared secret without revealing the secret itself
- Incorporate a **nonce** in the hash to prevent the hash from being reused as if it were just a different shared secret
- A nonce must be **known to both parties**, must **change over time**, and **cannot be influenced by an attacker**… but it **does not need to be secret**
- Decouple servers by using **tokens**, which are effectively encrypted messages from one server to another passed through the user

Cryptographic Protocols

- AES, RSA, SHA are cryptographic **algorithms**—mathematical procedures used to transform data.
- A cryptographic **protocol** is a series of steps and message exchanges between entities to achieve a specific cryptographic result… in other words, recipes that tell us how to apply algorithms.
- Protocols are effectively templates for writing programs, and so are created the same way we threat model programs: **identify objectives**, **model data flows**, **look for potential attacks** and **mitigate them**.

WPA2 Protocol

- "**Wireless Protected Access**, version **2**"
- Goals are to verify both client and router, and encrypt traffic against *external* (but not internal) listeners
- Router and client each generate nonces, then demonstrate knowledge of shared secret (the wifi password) by calculating hashes of the nonces
- The password, nonces and hashes are all combined to generate a symmetric encryption key

SSL / TLS

- "**Secure Sockets Layer**", which was renamed to "**Transport Layer Security**" when legacy-breaking changes were introduced
- Goals are to verify identity of the server and encrypt traffic against all listeners
- Client generates a key, then submits it to server encrypted with public half of server's asymmetric encryption key…
- …but how does the client *know* that's the server's public key?

TLS Certificate Chain

- The owner of foo.bar verifies their identity to Verisign (e.g.)
- The owner of foo.bar creates an asymmetric key pair, and asks Verisign to encrypt the hash of the public key with Verisign's private key
- The client calculates the hash of foo.bar's public key, gets Verisign's public key, decrypts the encrypted hash and sees that they match…
- …but how does the client *know* that's Verisign's public key?
- The root of the chain of trust lives on each PC… and can be tampered with, maliciously or incompetently.

Bitcoin

- Goals are to transfer value between entities without possibility of tampering or repudiation, and without requiring a central authority
- The history of all transactions is called a **ledger**. Ultimately, Bitcoin relies on everyone agreeing which copy of the ledger is the valid one.
- Each block of changes to the ledger is hashed. A block is only valid if its hash begins with a certain number of zeroes.
- Discovering a valid block is like winning the lottery. Searching for valid blocks is called **mining**.

Bitcoin Repudiation

- To repudiate a transaction, Alice would have to construct an alternate copy of the ledger without that transaction.
- A ledger copy is only valid if all the hashes are correct, so Alice would have to mine herself.
- The *one* accepted copy of the ledger is the one with the most hashes in it, so Alice would have to not only replace the existing one, but also replace all the work that got done to the true ledger while she was doing her mining.
- Thus, Alice would need more hashing power than the rest of the world combined to keep up.

How Passwords Get Stolen

- A minor danger of passwords is an attacker brute-force trying random passwords. Not possible with a well-implemented site.
- A major danger of passwords is an attacker downloading the password database. So the developer stores hashes, not passwords…
- …so the attacker generates a rainbow table of hashes…
- …so the developer stores **salted hashes**.
- A password that is strong enough to resist modern GPU cracking is probably too hard to remember… “correcthorsebatterystaple” can’t be trusted.

Passwords Suck

- The randomness contained in a password is called **entropy**. More is better.
- People are bad at generating entropy.
- People are bad at remembering high-entropy passwords.
- Failing to remember passwords requires sites to have password-reset mechanisms, which are themselves the #1 way accounts get compromised.
- Use **two-factor authentication** wherever possible.
- Use a **password manager**.

Script Injection

- Do not run other people's code in your system.
- Watch out for places where user input might be interpreted as code. For example: **SQL injection** and **script injection**.
- Making sure that user input does not contain injected code is called **sanitizing** it. This can be done through **blacklists** (lists of known-bad constructs), but is better done through **whitelists** (lists of known-good constructs).
- Unsafe input can either be deleted/discarded, or can be rendered safe by **escaping** it.

Buffer Overrun

- By the nature of its memory management, C/C++ is highly vulnerable to **buffer overruns**: reading or writing beyond the space allocated for an array or buffer.
- Anytime the attacker can control the contents of your memory, they have the potential ability to put their own code there and trick you into executing it.
- Use every tool available to find and prevent these errors: strict warnings, lint, etc.
- If performance is less important than security—and it usually is for Internet software—use another language.

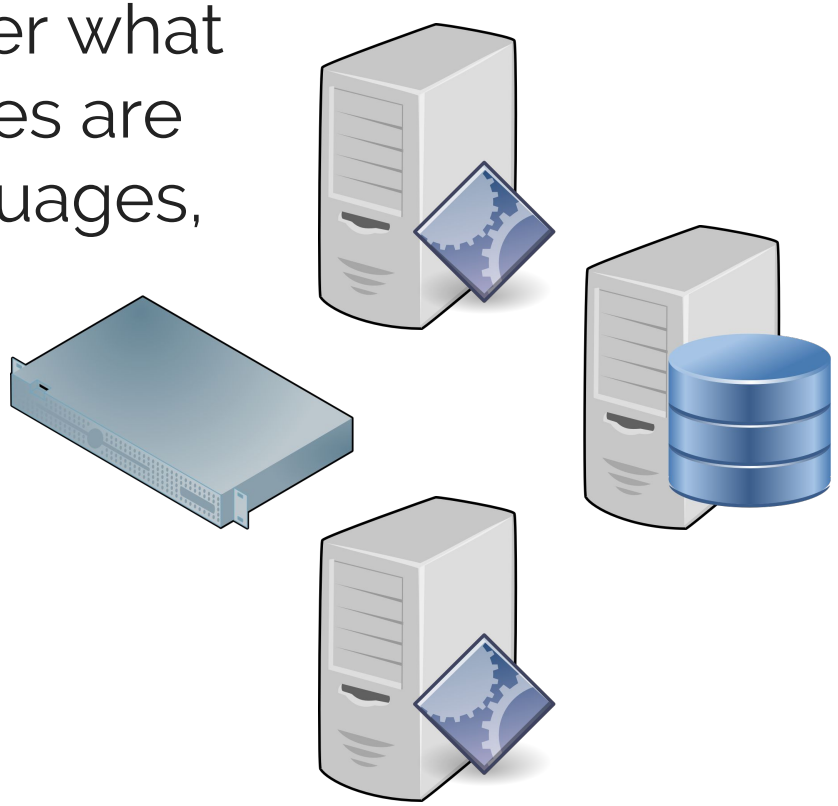
Error Handling

- It is almost always safer to crash outright than to continue in an unexpected or misunderstood state.
- When logging errors, be careful not to **leak information**.
- Likewise, when logging errors, be careful about string processing and accessing the hard drive.

*A **distributed system** is one in which two or more processes work together to achieve a computational result.*

Distributed System Tasks

All distributed systems perform the same tasks, no matter what the scale. All that changes are the tools (software, languages, hardware, etc.) used



- **Balance load**
- **Handle socket traffic**
- **Interpret requests**
- **Perform calculations**
- **Query data**
- **Cache data**
- **Compose responses**

Server Quantity

- Even small numbers of inexpensive servers can handle 10s to 100s of simultaneous users
- “Simultaneous users” will always be a mere fraction of total users
- Facebook serves about 1.5 billion users with about 250,000 servers
- **Cloud computing** just means “renting servers from someone else, by the day, hour or even minute”
- Microsoft and Google have 1M+ servers, Amazon 2M+.
- Netflix has zero.

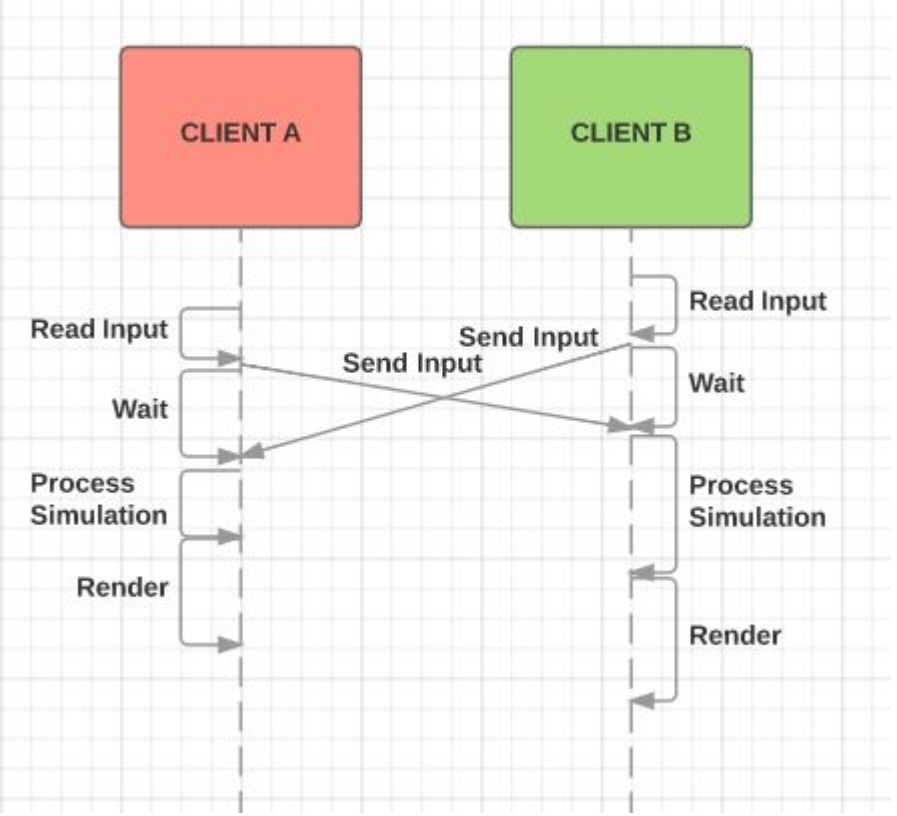
CAP Theorem

- The total information processed by a system is known as its **state**
- Ensuring that all processes in a distributed system have the same view of the state is known as **synchronization**
- The **CAP Theorem** tells us that when the possibility of a **P**artition exists (which it always does), we can either choose to be **C**onsistent or **A**vailable, but never both at the same time.
- Juggling consistency versus availability is the key challenge of synchronization

Lockstep Synchrony [Doom]

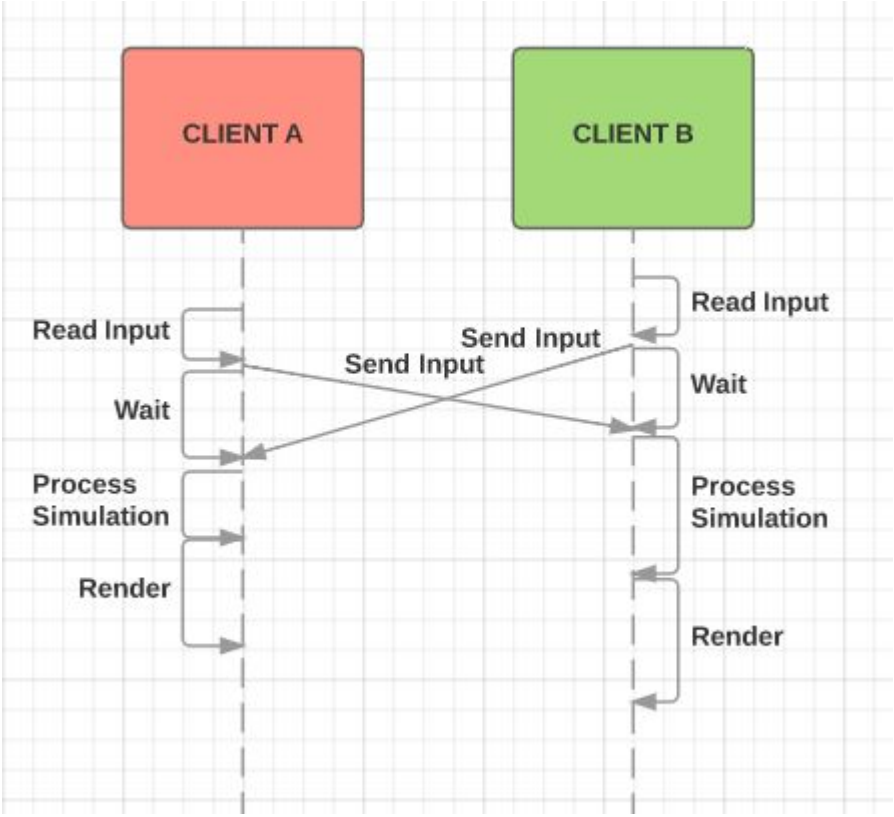
Simplest possible distributed system:

- Every node has a complete copy of state
- Every requested change is sent to every node
- All nodes apply the changes at the same time once every node has received every change



CONSISTENT but not AVAILABLE

Sequence Diagram

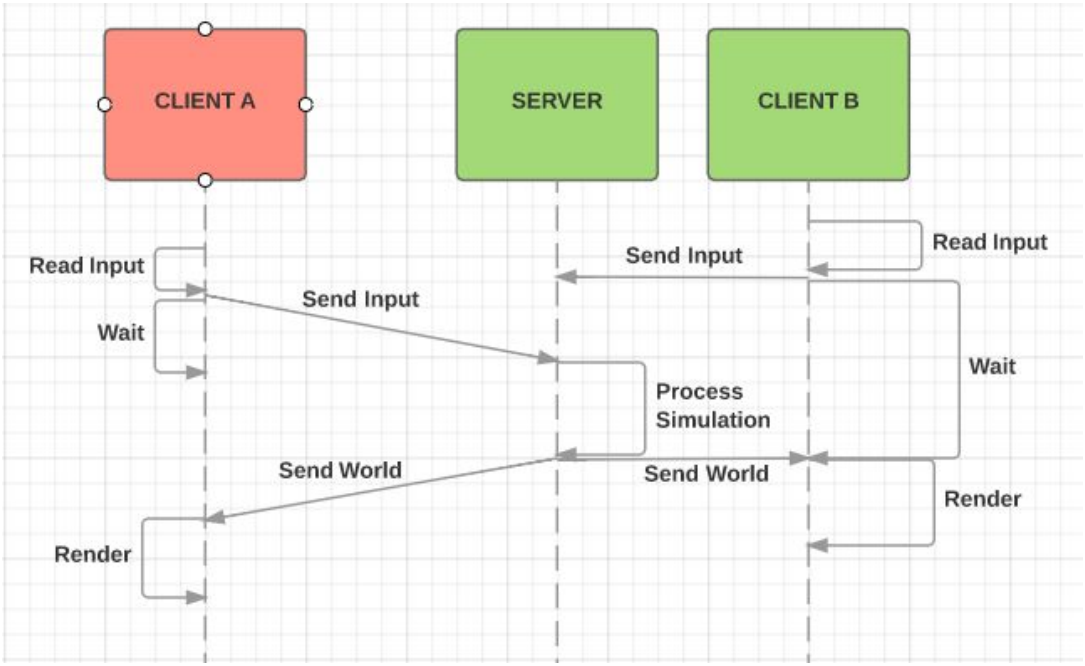


- Each line represents a process
- Time runs from top to bottom
- Each arrow is a function call or message
- Slope of arrow represents **lag** (delay between request and result)

Client-Server Synchrony [Quake; HTTP]

More secure:

- Only one copy of state (at host)
- All nodes send change requests to host
- Host sends new state to nodes
- Nodes render new state

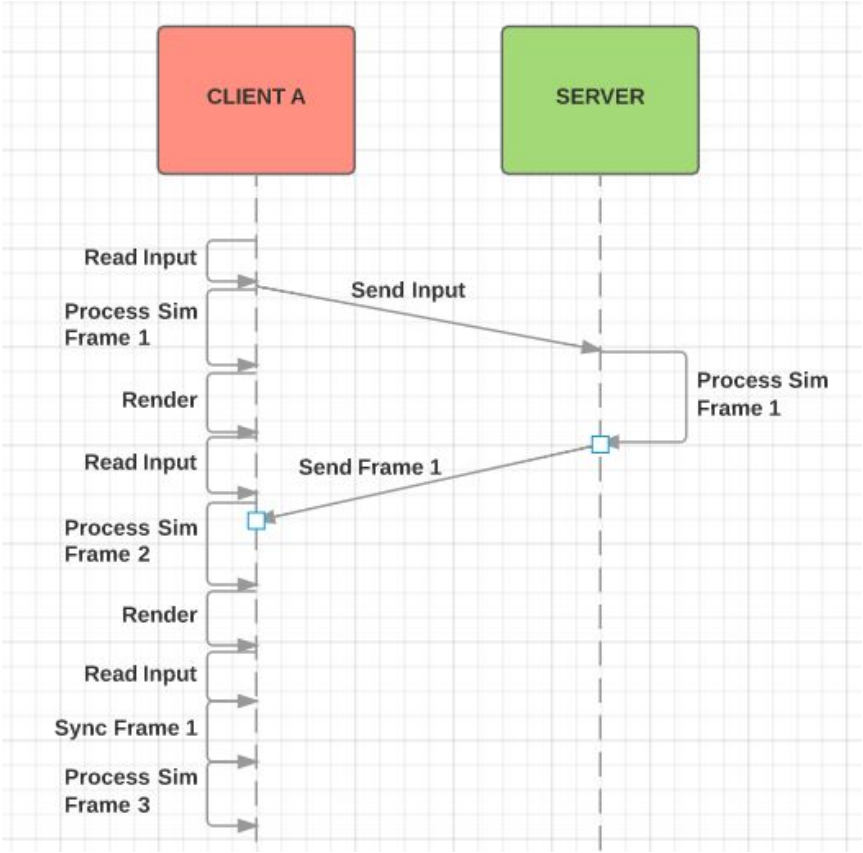


CONSISTENT but not AVAILABLE

Optimistic Synchrony [all the cool kids]

More responsive:

- One master copy of state, but each client has local copy
- All nodes send change requests *but also* apply change to local copy
- Local state is always out of date, so **prediction** is necessary
- Conflicts are resolved as they are detected



AVAILABLE but not CONSISTENT

Networked Game “Easy Mode”

- Use “client/server” (“dumb client”) networking…less burden on precise determinism, and known-good server for (re)join-in-progress
- Use TCP… if packets are only very rarely dropped, TCP and UDP are basically identical
- Packets shouldn’t get dropped if you only run on ethernet

Difficulties

- Lag is caused by **latency** plus **transmission time**
- Latency is increased by **distance** (both **logical** and **physical**), **congestion** and **packet loss**. It is usually measured by **ping time**, which is double the actual amount.
- Transmission time is effectively packet size divided by bandwidth. Available bandwidth is always lower than claimed ISP speeds. Xbox Live requires games to run with as little as **8KB/sec**.
- Packet loss shouldn't happen on wired networks, but is endemic on wireless ones
- **Jitter** is unpredictable swings in lag

HTTP

- HTTP was invented to fulfill the vision of the **memex**: a networked system that *fetches documents* and renders *connections between them*
- Those inter-document connections are called **hyperlinks**, and documents containing them are called **hypertext**.
- Hyperlinks consist of **uniform resource locators** (URLs). An URL describes **where to find a document** and **how to retrieve it**.
- The description of where to find a document is a **uniform resource identifier**. All URLs are URIs.

HTTP Protocol

- Fundamentally **client-driven**: The client opens a connection, makes one or more requests, then closes the connection.
- Connection is made to an IP address, but URLs feature domain names. Thus, the HTTP protocol requires specifying the target domain name as part of the request.
- Requests consist of a **verb**, an **URL**, **headers** and an optional **body**. Responses consist of a **status code**, **headers** and a **body**.

HTTP Protocol (continued)

- Verbs: **GET**, **POST**, **HEAD**, **PUT** (rarely), **DELETE** (almost never)
- Status codes: **2xx** = success, **3xx** = redirection, **4xx** = bad request, **5xx** = server error
- Headers: Control various aspects of the request and response, such as target **host**, the **content type** of the body, how long to **cache** the response, user **authentication**, etc.
- Cookies: A special type of header used to attach state to future client requests.

Web Services

- The HTTP protocol has become in effect a new transport layer on top of the ATIN stack.
- A basic challenge of writing **web services** is deciding what tasks require **push** from server to client, versus which can settle for the much-easier **pull** request from the client.
- Web services generally focus on storing data objects, and providing **CRUDI** operations for them: **create**, **read**, **update**, **delete** and **index**.

FRIDAY
12/15 3:00PM
MICHELANGELO

Credits

Special thanks to all the people who made and released these awesome resources for free:

- Presentation template by SlidesCarnival
- Photographs by Unsplash

Presentation design

This presentations uses the following typographies and colors:

- Titles: **Merriweather**
- Body copy: **Raleway**

You can download the fonts on this page:

https://www.google.com/fonts#UsePlace:use/Collection:Merriweather:400,400italic,700,700italic|Raleway:400,700

Click on the “arrow button” that appears on the top right

- Dark gray **#222222**
- Wine **#a8122a**
- Cream **#f5f1e0**

You don’t need to keep this slide in your presentation. It’s only here to serve you as a design guide if you need to create new slides or download the fonts to edit the presentation in PowerPoint®

SlidesCarnival icons are editable shapes.

This means that you can:

- Resize them without losing quality.
- Change fill color and opacity.
- Change line color, width and style.

Isn't that nice? :)

Examples: