Blockchain Basics

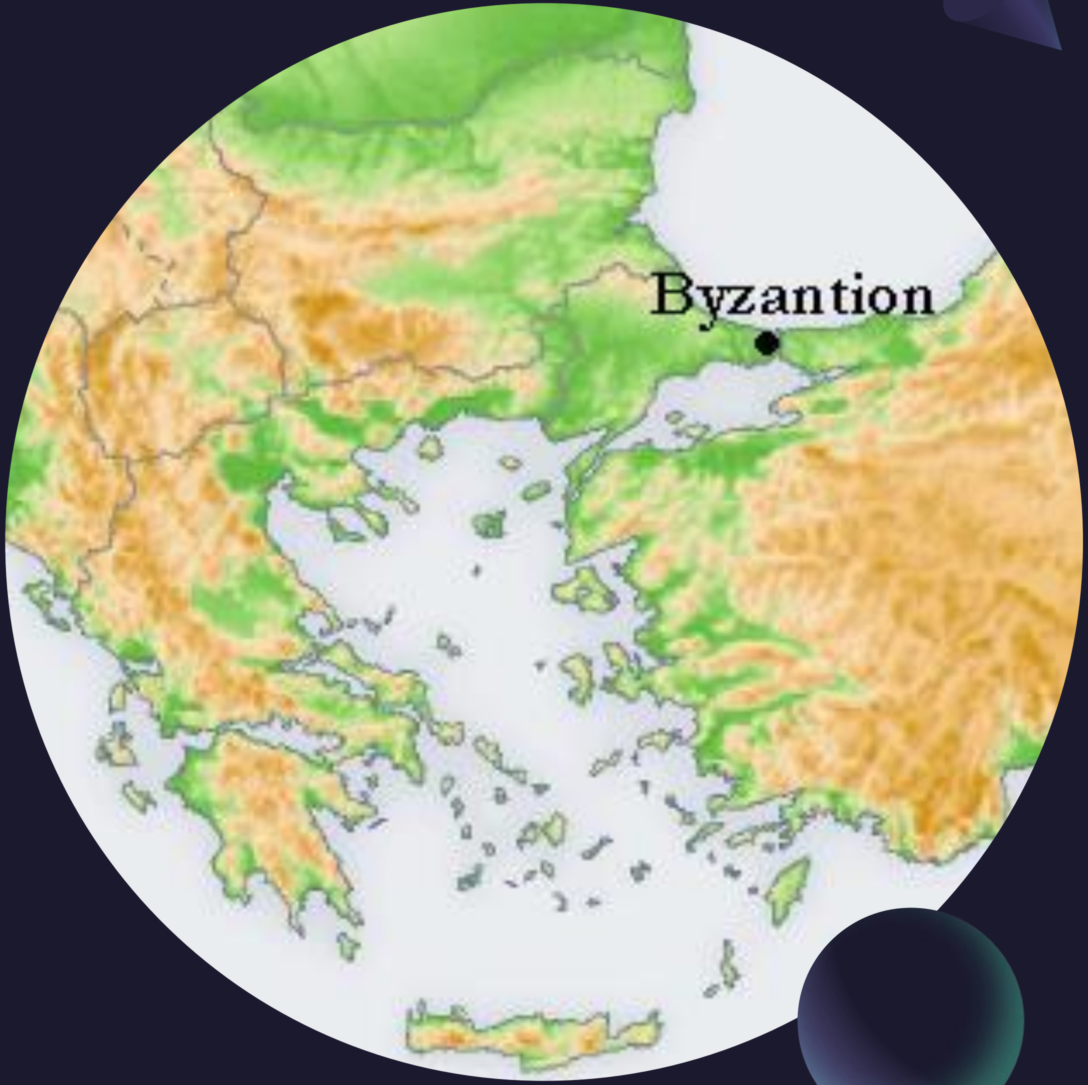
Kavinga Yapa Abeywardena

Department of Computer Systems Engineering

Today

- It's blockchain time!
- We're going to learn about the Byzantine Generals Problem
  - What is it and how does it affect computing?
- We're going to learn about Blockchain
  - What is it
  - How does it work
  - How do you make one
- And also how to deploy it
  - Where can you put blockchain?
  - When should you use blockchain?

Byzantium



Byzantium

Also known as Constantinople

Also known as Istanbul

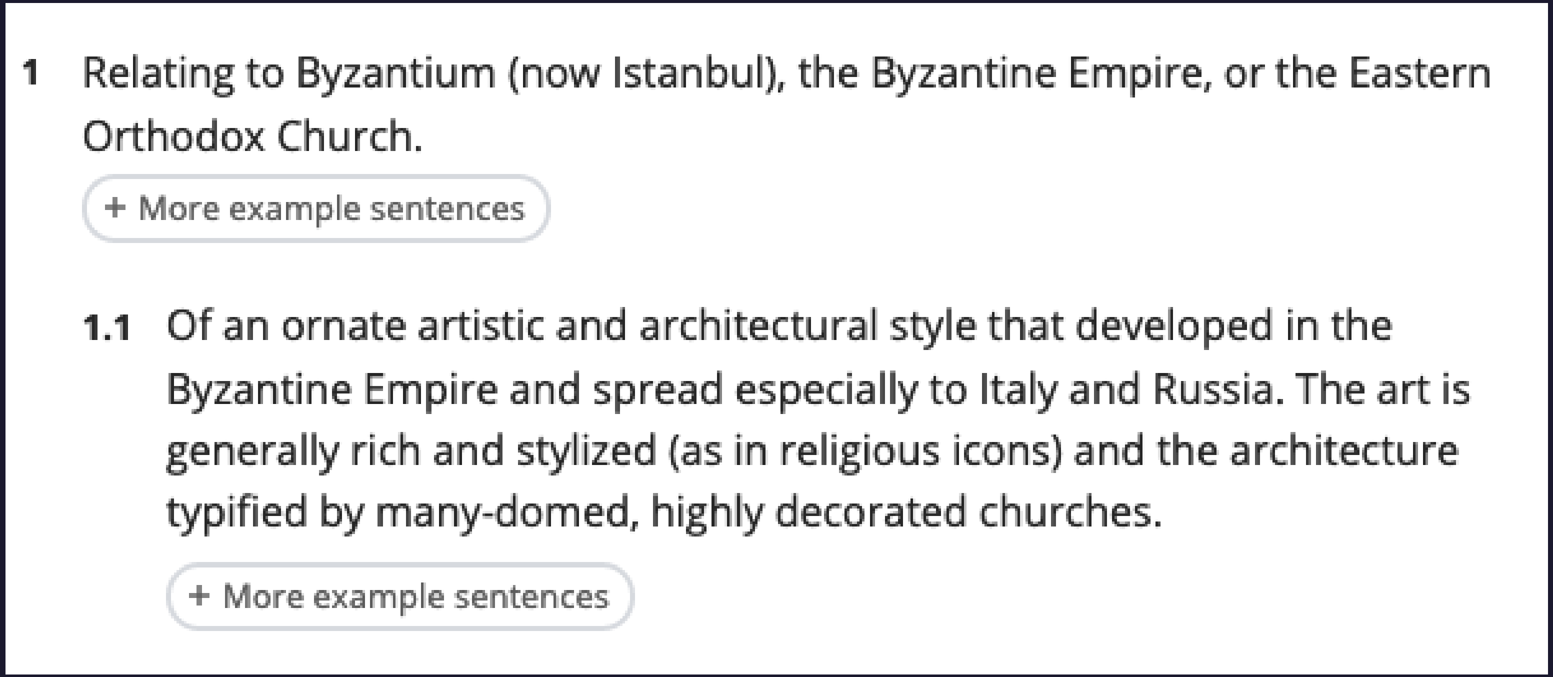
Also known as the capital of the Byzantine Empire

- Named for, you guessed it, Byzantium
- Also known as the Eastern Roman Empire

Basically a city of many names

And a city of *many* bureaucrats back in Byzantine times.

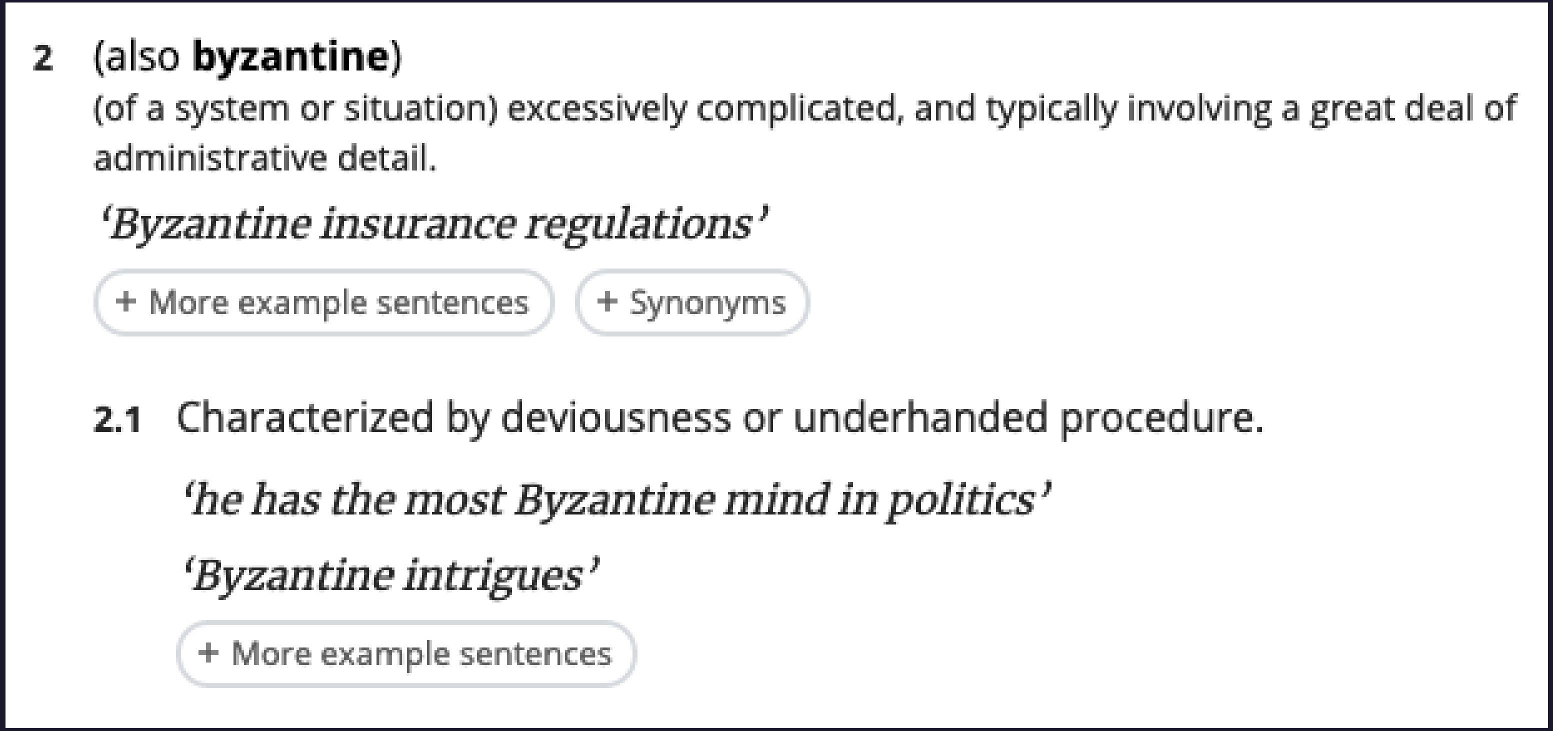
Byzantine



1 Relating to Byzantium (now Istanbul), the Byzantine Empire, or the Eastern Orthodox Church.

1.1 Of an ornate artistic and architectural style that developed in the Byzantine Empire and spread especially to Italy and Russia. The art is generally rich and stylized (as in religious icons) and the architecture typified by many-domed, highly decorated churches.

Byzantine (also)


2 (also **byzantine**)
(of a system or situation) excessively complicated, and typically involving a great deal of administrative detail.

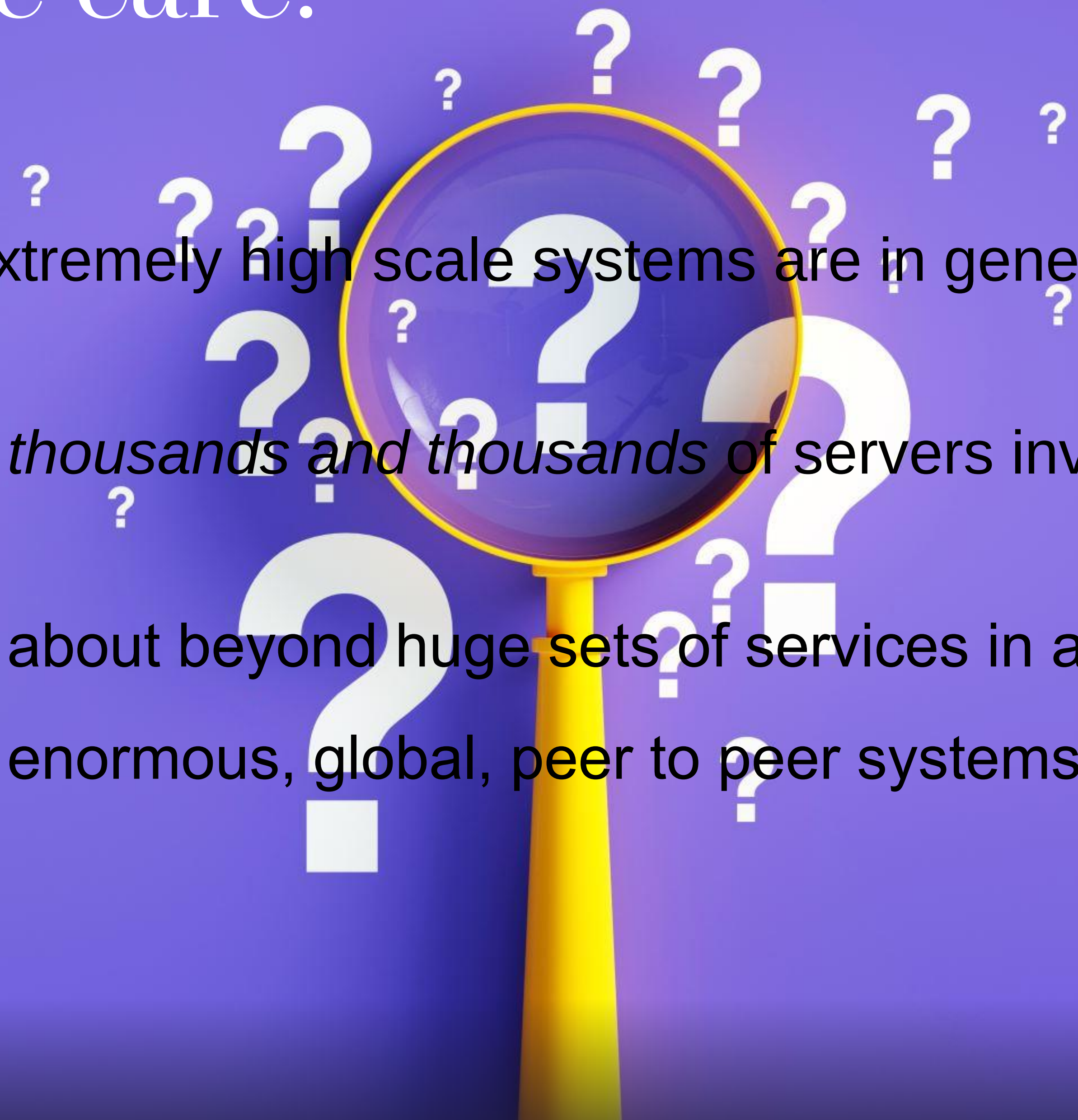
'Byzantine insurance regulations'

2.1 Characterized by deviousness or underhanded procedure.

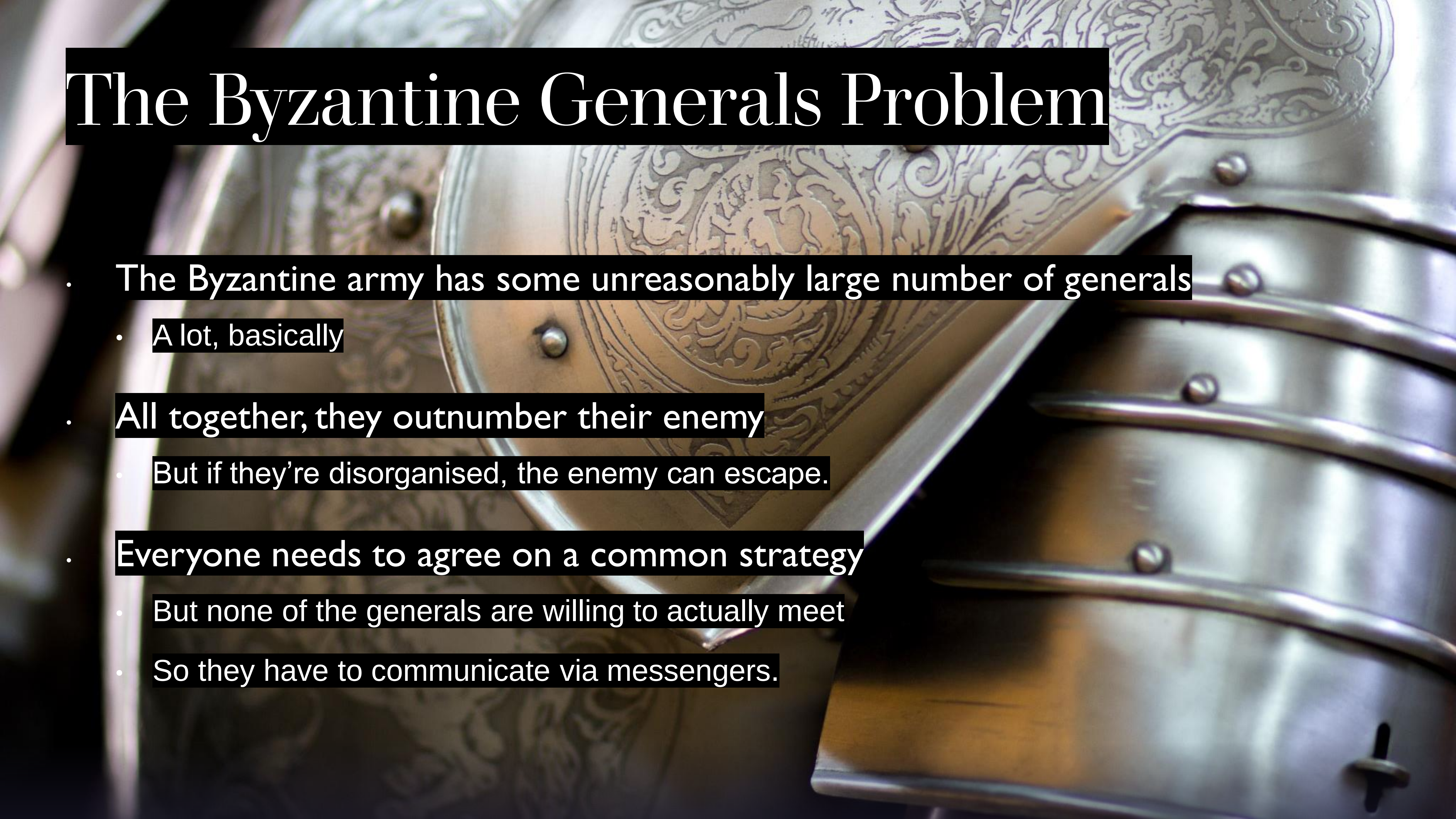
'he has the most Byzantine mind in politics'

'Byzantine intrigues'

Why do we care?



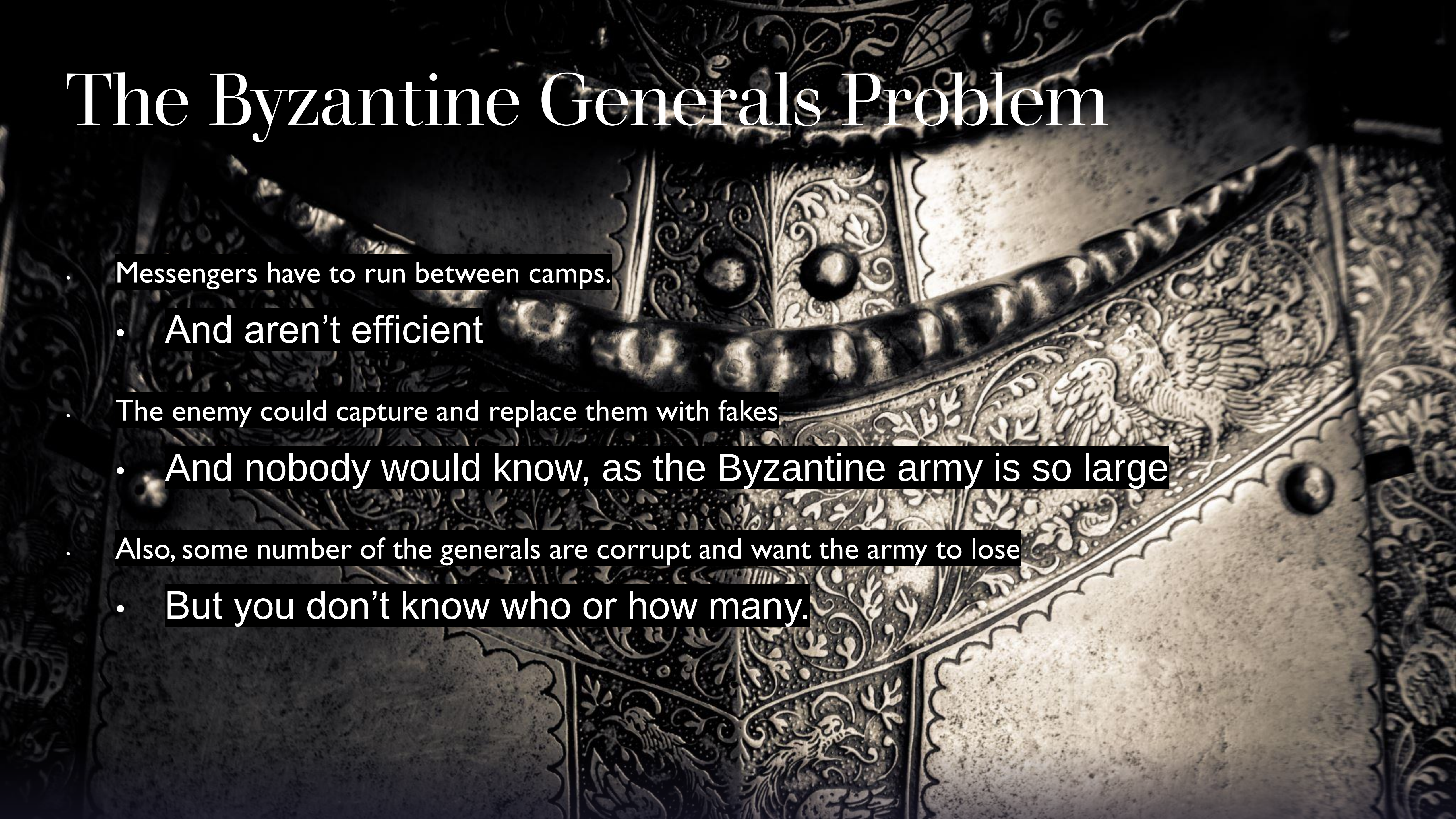
- Simply put, extremely high scale systems are in general, byzantine in nature.
- We’re talking *thousands and thousands* of servers involved in your service.
- We’re talking about beyond huge sets of services in a massive system.
- We’re talking enormous, global, peer to peer systems.



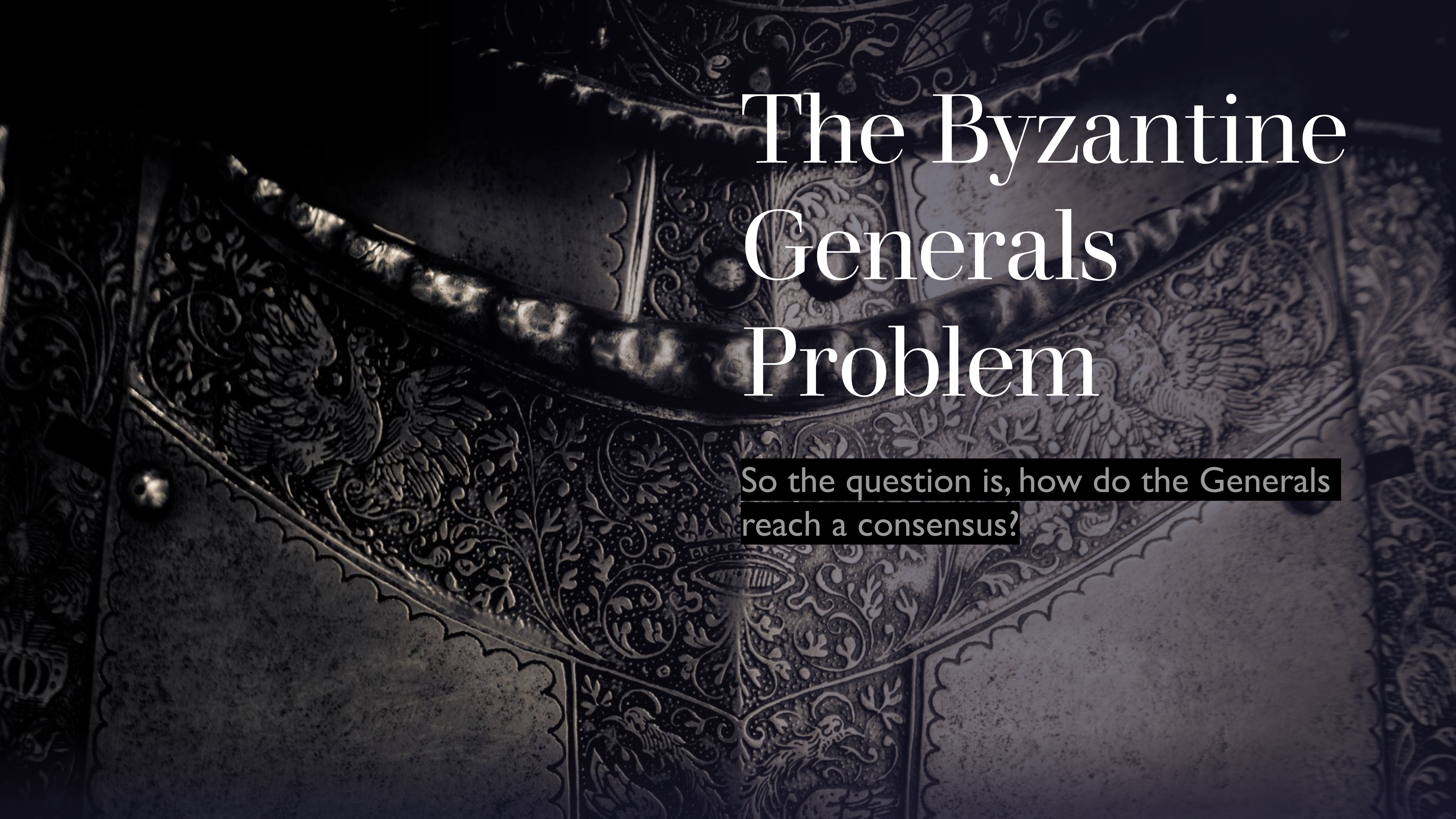
The Byzantine Generals Problem

- The Byzantine army has some unreasonably large number of generals
  - A lot, basically
- All together, they outnumber their enemy
  - But if they’re disorganised, the enemy can escape.
- Everyone needs to agree on a common strategy
  - But none of the generals are willing to actually meet
  - So they have to communicate via messengers.

The Byzantine Generals Problem



- Messengers have to run between camps.
  - And aren't efficient
- The enemy could capture and replace them with fakes
  - And nobody would know, as the Byzantine army is so large
- Also, some number of the generals are corrupt and want the army to lose
  - But you don't know who or how many.



The Byzantine Generals Problem

So the question is, how do the Generals reach a consensus?

A Byzantine Fault

- Essentially a kind of distributed system heisenbug.
    - The nature of the error changes depending on where you are in the system
    - Some machines think everything is fine, some machines believe the world is ending
    - It is not possible to determine which is true and which is false
    - It's also hard to determine what part of the system actually broke.
- Basically the worst kind of error in Distributed Computing

When does this happen?

- At tremendous scale.
  - Can happen at smaller scale, but rare and much easier to fix
- When a system needs to keep a unified state distributed across many systems
  - And failure to do so would be catastrophic
- The kind of situation a distributed data tier would find itself in.

What goes wrong

- In complex distributed data tiers, things get out of sync
  - Pretty much all the time
- And when that happens, the data tier as a whole needs to decide on what is the definitive data set
- This is made significantly more difficult if the database is time-critical.

What has to happen?

- When a member of the consensus has an update, they broadcast it.
- Other members then need to determine the validity, and confirm with each other that it is a valid update.
- Any communication involved with this could be compromised.

Byzantine Fault Tolerance

- BFT is the ability to resist Byzantine Faults
- Systems for this exist all over the place
  - In computers
  - In planes
  - In rockets
  - In basically anything with a bunch of component systems that could fail in such a manner.

How to be BFT?

- In order to be Byzantine Fault Tolerant, the system must be able to come to a consensus even if some clients are wrong
    - Or evil
- There are two ways of solving this
    - The Commander and Lieutenants method
    - The Unforgeable Signatures method

Commander and Lieutenant method

- One potentially corrupt commander
- Several loyal lieutenants
- Lieutenants share the messages the commander gave them amongst themselves
- Can determine if the Commander appears corrupt amongst themselves
    - Doesn’t fix the communication problem
    - What if a lieutenant is corrupt?

Unforgeable Signatures Method

- All messages have signatures that cannot be forged by the enemy.
- It is immediately noticeable that any alterations to a message have been made.
- Everyone knows when something has gone wrong
  - This still doesn’t deal with corrupt generals

What to do?

- Commander/Lieutenant model solves *one* of the corrupt generals problem
- Unforgeable signatures stops all inadvertent errors in communication
- But neither is a full package!
- So… why not combine them?

The Trust No-One model

- All members use unforgeable signatures
- All members assume everyone is corrupt, and try to use their neighbours as lieutenants.
- Assuming enough members are loyal, a consensus can still be reached
  - By progressively redoing all lieutenant checks involving untrustworthy members.
- So long as there are at least 1/3 good members, consensus will be valid!

BFT Systems

BFT is hard to achieve

It's very complicated and requires it's own set of protocols on top of your usual distributed system

Essentially necessary at large scale

Especially with distributed data tiers

There are better ways to handle this at lower scale!

But what if you really need this? What do you do?

Blockchain

Blockchain is a highly adaptable method of maintaining a BFT database.

Can be used in both Server-Client infrastructure, and Peer-to-Peer.

Unfortunately, it is also the most hyped technology around atm

- Even more so than Deep Learning

So what it *actually does* can be elusive.

Blockchain

Basically acts as a unified ledger of transactions

Database state is the sum of those transactions

Transactions are grouped into “Blocks”

These blocks are then ”chained” together

This chaining process is the key to the whole thing

The chaining process

Each block has a specific hash associated with it.

These hashes are included in the next block

In this way the blocks are mathematically predicated on the previous block

This means each chain is uniquely identifiable from a single hash

- Even if you have the same transactions in the block!

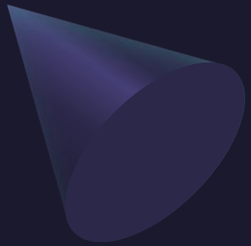
What chains let you do

By chaining blocks like this, an out of sync member of the swarm can tell if they're out of sync.

And can tell if they've got a valid chain.

A member can identify the correct chain from neighbours.

Extremely efficient.



How to deploy a Blockchain

- Database must be a ledger of transactions
  - Otherwise, blocks too big
- Data tier must connect to each other in peer to peer network
  - Structured or unstructured, not really important
- Peers need to be semi-specialised
  - Some are miners, some are storers
  - These can be unified into miner-storers
- Then, implement a blockchain and communications mechanism.

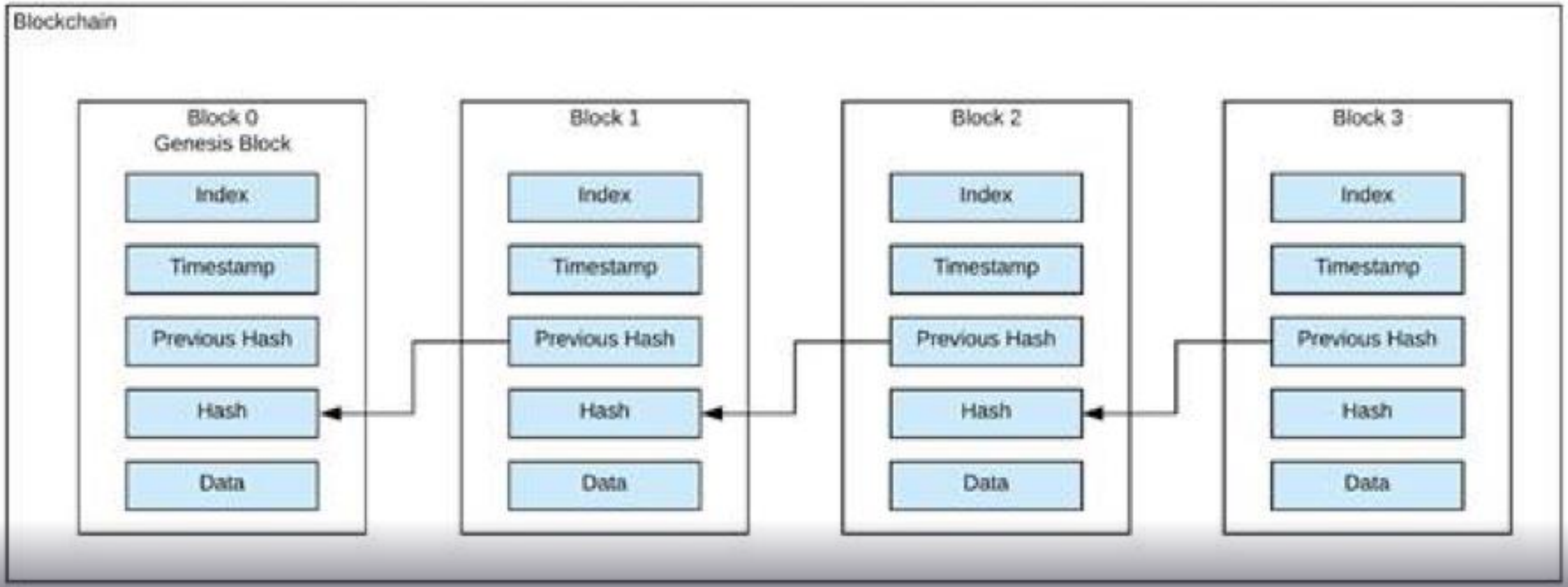
Implementing a Blockchain

Each block needs to contain:

- Some number of transactions
- An identifier
- The last block's hash
- A nonce
- The block's hash

This is the minimum number of things required.

What this should look like



The Nonce

Sometimes it’s worth increasing the difficulty of creating the hash.

This may be due to the sheer number of transactions.

This may also be because the system needs to be more secure, and synchronisation needs more time to happen.

The Nonce

- One way of increasing difficulty is with a hash challenge
- Valid hashes must follow some sort of rule
  - Some rules are easy, some rules are hard
- The only way a system can affect the outcome of the hash is with a variable, the Nonce.
- Sometimes the Nonce has special rules too!

Do you need to use a Nonce?

- Not at all!
- If you are certain you control all members of the swarm, you don't even need cryptographic hashes
  - Something simple like a CRC checksum will do
  - As long as you follow the rules of how a blockchain is made, it will be highly resilient to Byzantine Faults
    - Just not ones caused by malicious members of the swarm.ß

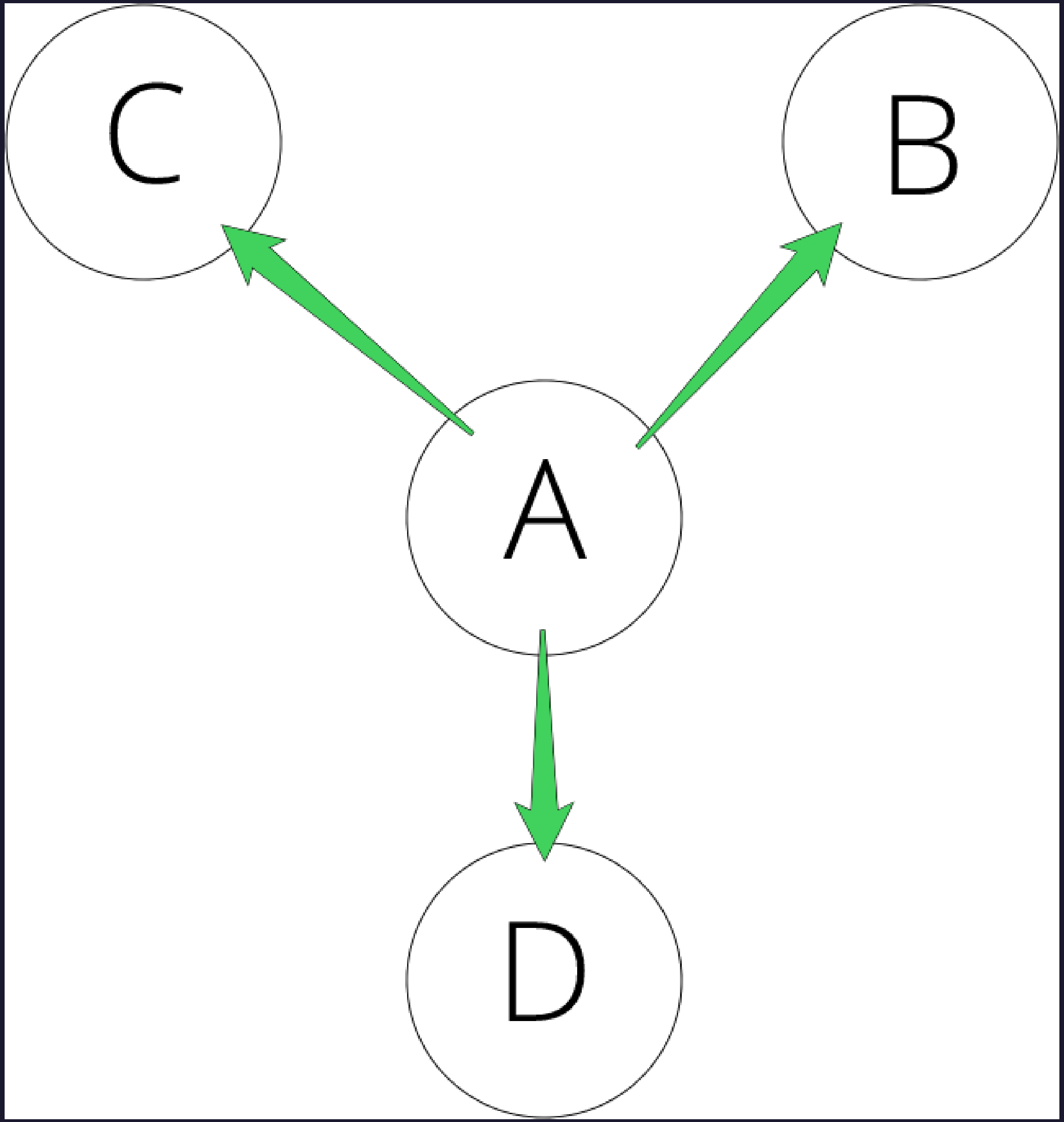
Getting your Blockchain from A to B

- There are many ways a new transaction can be added
- Let’s assume for this lecture that they are submitted to one of the data tier machines.
- When the transaction is added, then what?
- Let’s assume that the data tier server produces a hashed block for the transaction (inefficient, but whatever)

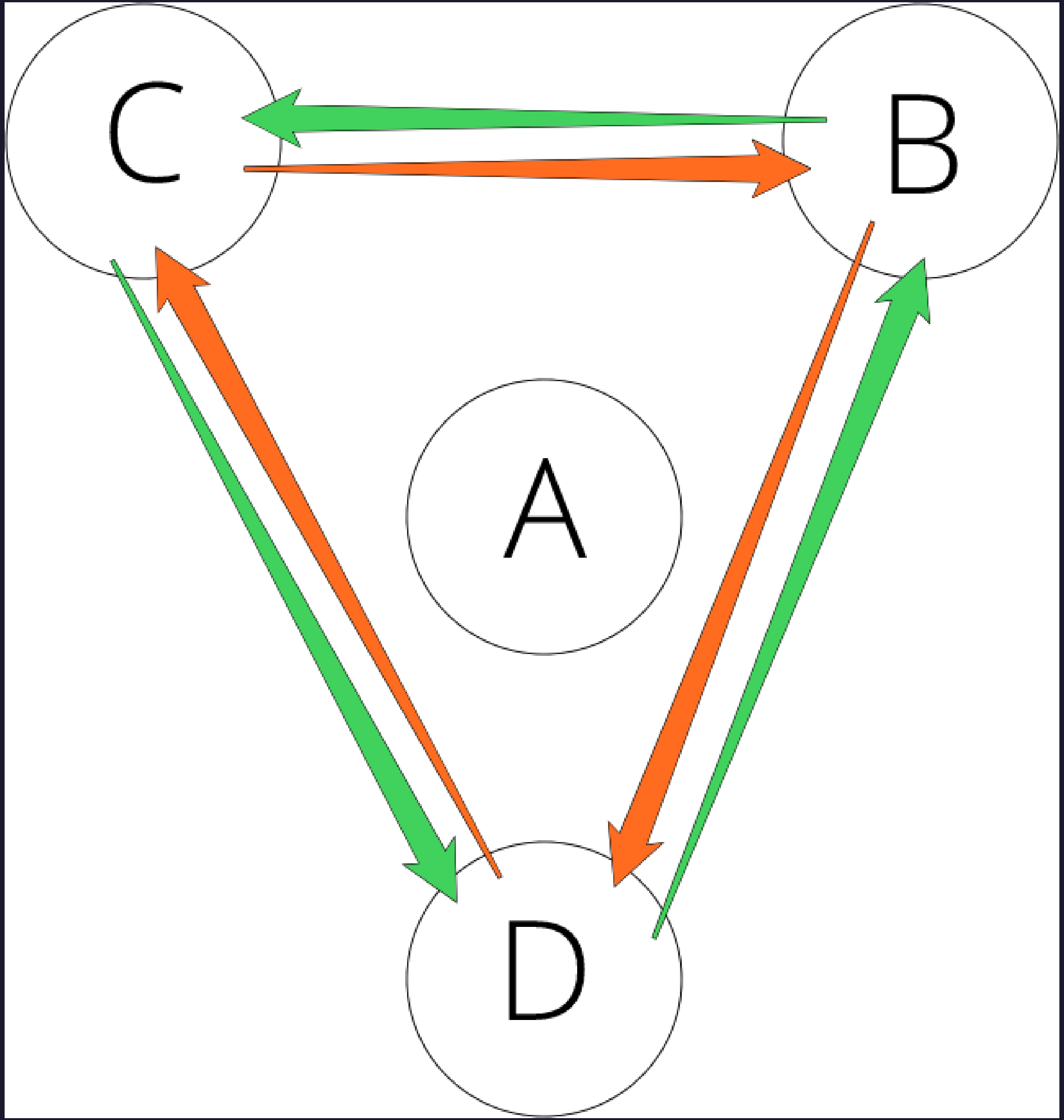
Moving Blocks

- The server then broadcasts to it’s neighbours the new block.
- The neighbours can then confer between each other to see if the block is valid
    - All rules are followed
    - The same block was sent to everyone
    - The previous hash is the current hash they have in their chains
- They then add the block to their chains, and broadcast it to their neighbours.

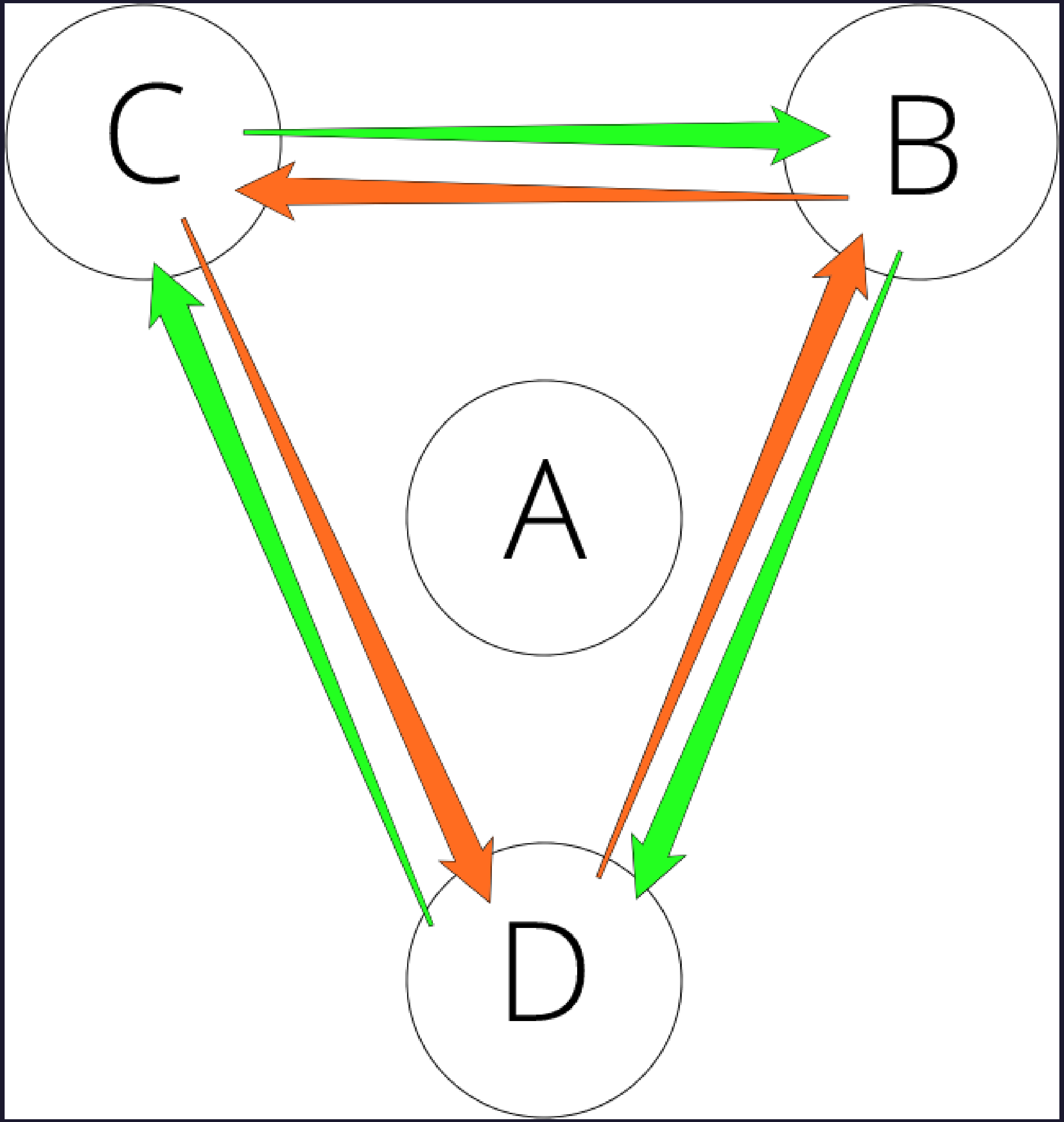
An Example


$$\begin{tikzcd} C & & B \\ & A \arrow[ul] \arrow[ur] \arrow[d] & \\ & D & \end{tikzcd}$$

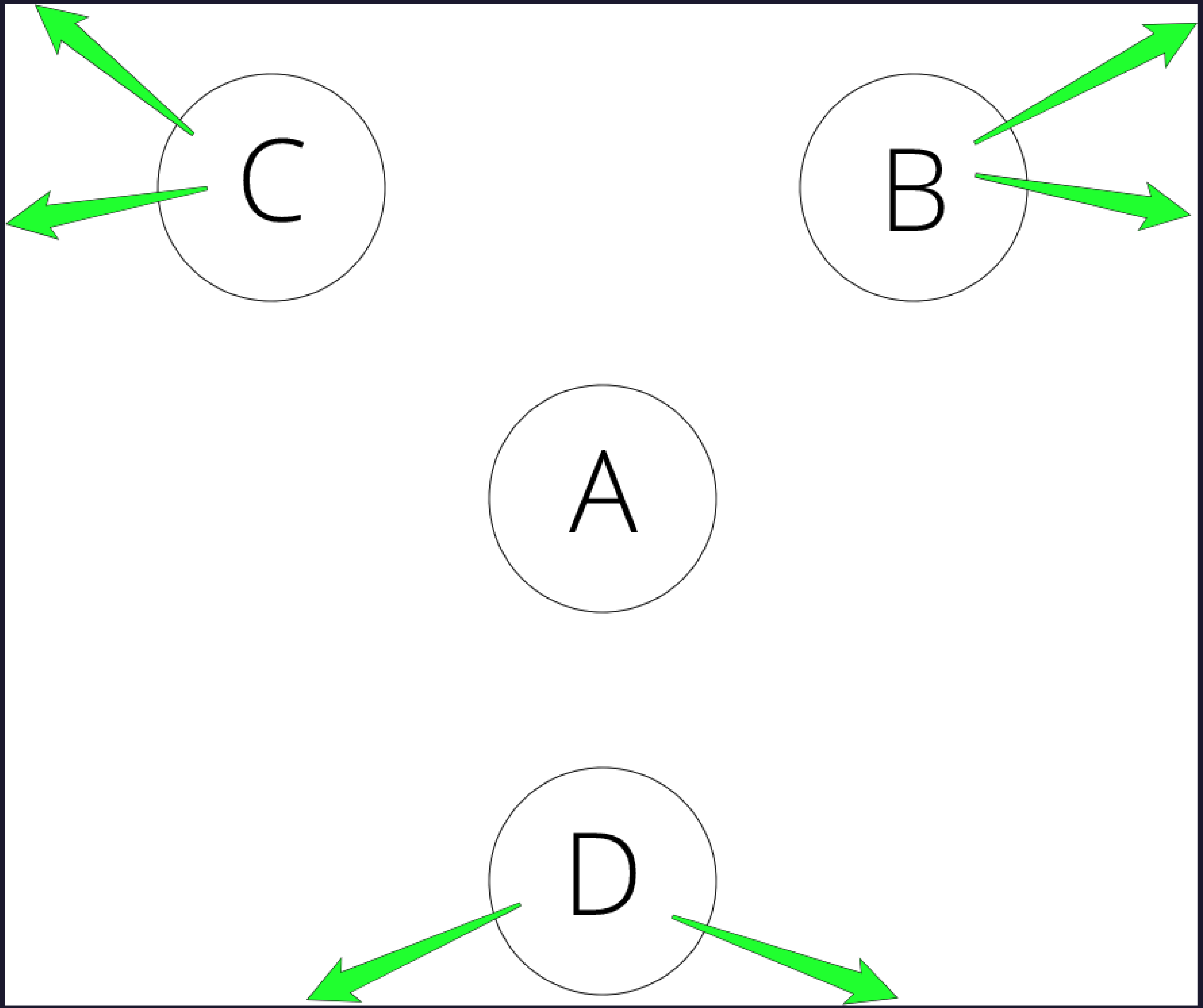
An Example



An Example



An Example



If Something Goes Wrong

As all blocks are hashed, we know that communications are clear.

Any disparity is cleared by first re-requesting, and then by rejecting.

A node will notice this when they can’t find their hash in the current blockchain.

They will then have to re-sync.

Re-Syncing a Blockchain

What if there is a clash?

- Half the servers have a different blockchain to the other half

Eventually, one side will win out as being larger

The others will have to synchronise to the new chain

If possible, a transaction will be made that rectifies the changes

Otherwise, the entire losing blockchain is rejected.

Isn’t that a bit extreme?

- Rejecting an entire blockchain might seem a little extreme, but remember:
  - Most of the blockchain will be the same, as they started at the same place
  - If there is some unreconcilable transaction, then something illegal has already occurred (someone has created two accounts with the same username, etc). This is simply resolving that issue.
  - Reconcilable transactions will still be carried over! Legitimate changes to the database are not lost!

When should you use Blockchain?

- Blockchain technologies are extremely computationally expensive.
  - Imagine how expensive *non* blockchain BFT tech is!
- Even when you're using an insecure method of hashing, you still need to perform many network operations.
- You should *only* use Blockchain when absolutely necessary.
  - Or when you're trying to get investors for your new app.

The Blockchain Hype

Unfortunately, Blockchain is the backbone of Cryptocurrencies.

Which are extremely hype rich, and fact poor.

This means that (in general) the general public don’t know what Blockchain is

They just know it’s really cool and apparently the future

So be *very* careful with the concept out in industry!

When to use Blockchain

The only time Blockchain is a good idea is when:

- You have a *very large* (thousands of servers) peer-to-peer system
- They are sharing data of some kind
- The state must be kept in sync
- No one organisation can be the owner of the data
- Byzantine Fault Tolerance is critical

If these are *not* all true, a traditional multi-tiered system is better.

But isn't Blockchain Tamper Resistant?

Yes, but that's BFT.

We have a method of doing this with client-server communications, Public Key Cryptography

- It's what's used for HTTPS/TLS certificates!
- It protects your internet purchases, and your VPN!
- It is more efficient, faster, easier to implement, and most Blockchain implementations *already use it* as part of their system!

Distributed (but not P2P) Data Tier?

What you want is caching.

Have another Data tier that is the master database.

- Slice the database up, and make fast hashes of that data.

Other data tier servers check to see if the data currently being accessed has a different hash to the one in the Master DB.

This is a widely available and fast technology.

The Thing About Blocks And Chains

You should only use Blockchain if *no other solution can work!*

- Truly gargantuan scale.
- Peer to Peer networks controlled by untrustworthy users.
- Doing illegal stuff.
- No, really. One of the largest userbases are botnets.

Blockchain is an extremely complicated technology to implement properly.

- Don’t do it if you don’t need to.

Aww man, but Blockchain is Cool!

- Yes, it is.
- But it's not practical in any sense.
- Nearly everything that can be implemented in Blockchain can be done better without it.
- Cool but impractical is fine for personal projects…
- But bitcoin mining is now the largest consumer of compute power on Earth.
- Please be responsible with your choice of tech!

Thank You

Kavinga Yapa Abeywardena

Dept. CSE