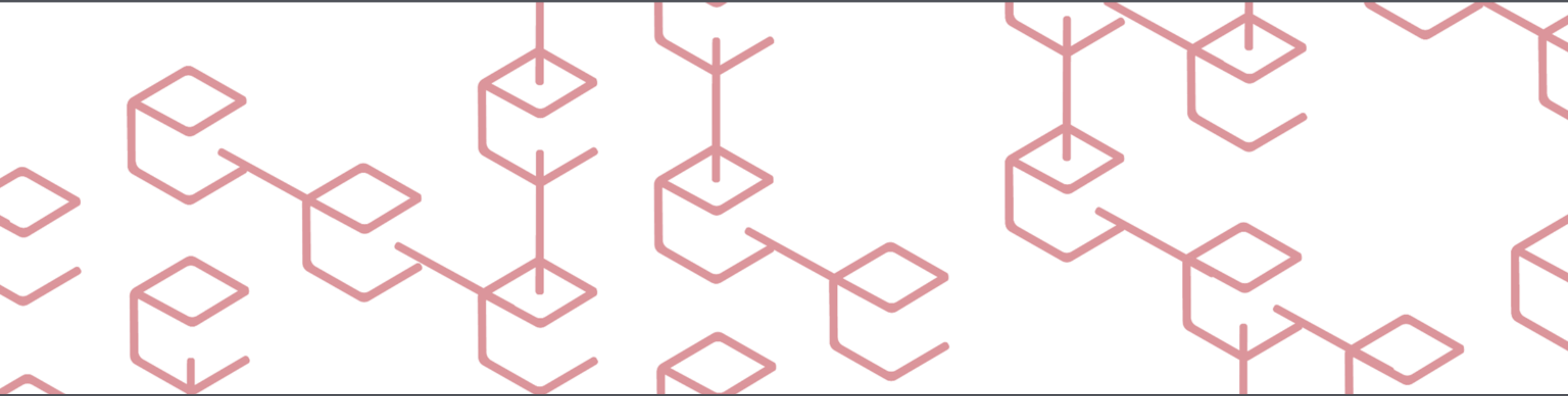
University of Reading

Blockchain: Course Overview



A quick overview of the course structure and content

Innovations in Distributed Architectures

Distributed Ledger – Block Chain

Lecture Breakdown (1)

- Lecture 1: Introduction to Blockchain
  - What is a blockchain? Attributes, use-cases and examples
- Lecture 2: Blockchain Concepts
  - Transaction processing, hashing and cryptographic signatures
- Lecture 3: Blockchain Concepts (2)
  - Blocks, nodes, networks and consensus algorithms
- Lecture 4: Consensus Protocol
  - Proof-of-Work vs. Proof-of-Stake
- Lecture 5: Smart Contracts
  - Smart vulnerabilities and threats in the past

Lecture Breakdown (2)

- Lecture 6: Blockchain Security and Vulnerabilities
  - Concepts concerning integrity
- Lecture 7: Byzantine Fault Tolerance
  - Fault tolerance amongst the peer-to-peer network of nodes
- Lecture 8: Hashing – Merkle Root Algorithm
  - Merkle root algorithm, fork resolution and interoperability
- Lecture 9: Public and Private Blockchains
  - Security and privacy in public and private blockchains
- Lecture 10: Legal Issues
  - Ethically and socially responsible designs of blockchains

WWW

Revolutionised Information

Information Data Highway

Web2

Programmable Web

Social Media & Sharing Economy

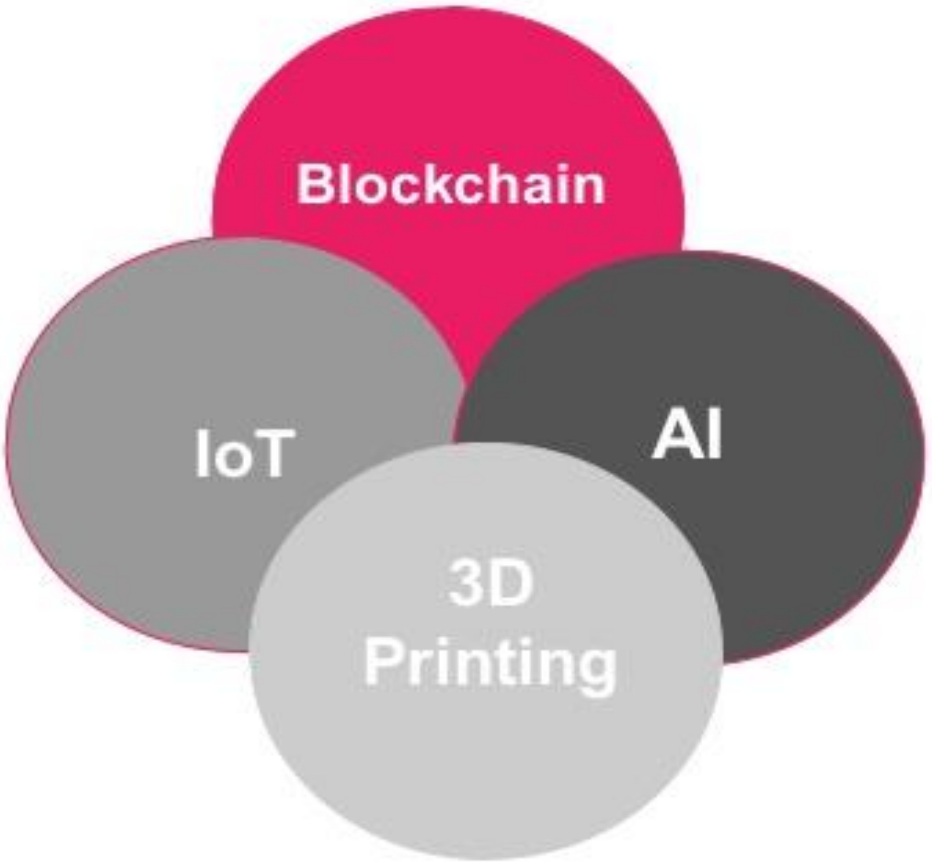
P2P solutions with middle man

Web3

Decentral Web

P2P transactions without middleman

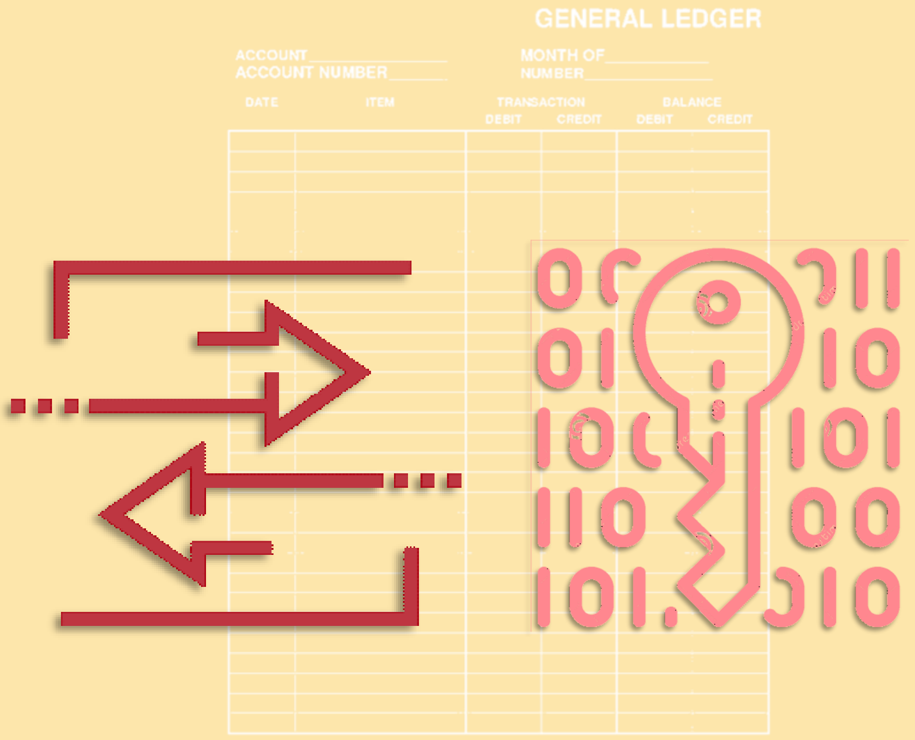
Future much more decentralised



The Future
Distributed, Decentralised,
Democratising Technologies

What is a Blockchain?

- Formally, Blockchain is a technology that:
  - Permits **transactions** to be gathered into **blocks** and recorded;
  - Cryptographically chains blocks in chronological order;
  - Allows the resulting **ledger** to be accessed by different servers.



What is a Blockchain? (2)

- A higher-level description is as follows:
  - A Blockchain is essentially chain of “**blocks”**
    - **Blocks** contain a number of **transactions**
  - New **blocks** are **appended** to the end of the **chain**

  - An identical **ledger** is distributed to all peers of a network for **validation**
    - A **ledger** is a record of all previous transactions.

  - Using a **consensus protocol “**rules” are defined
    - What is a **valid** transaction and block?
    - Who will add the **next** block to the chain?

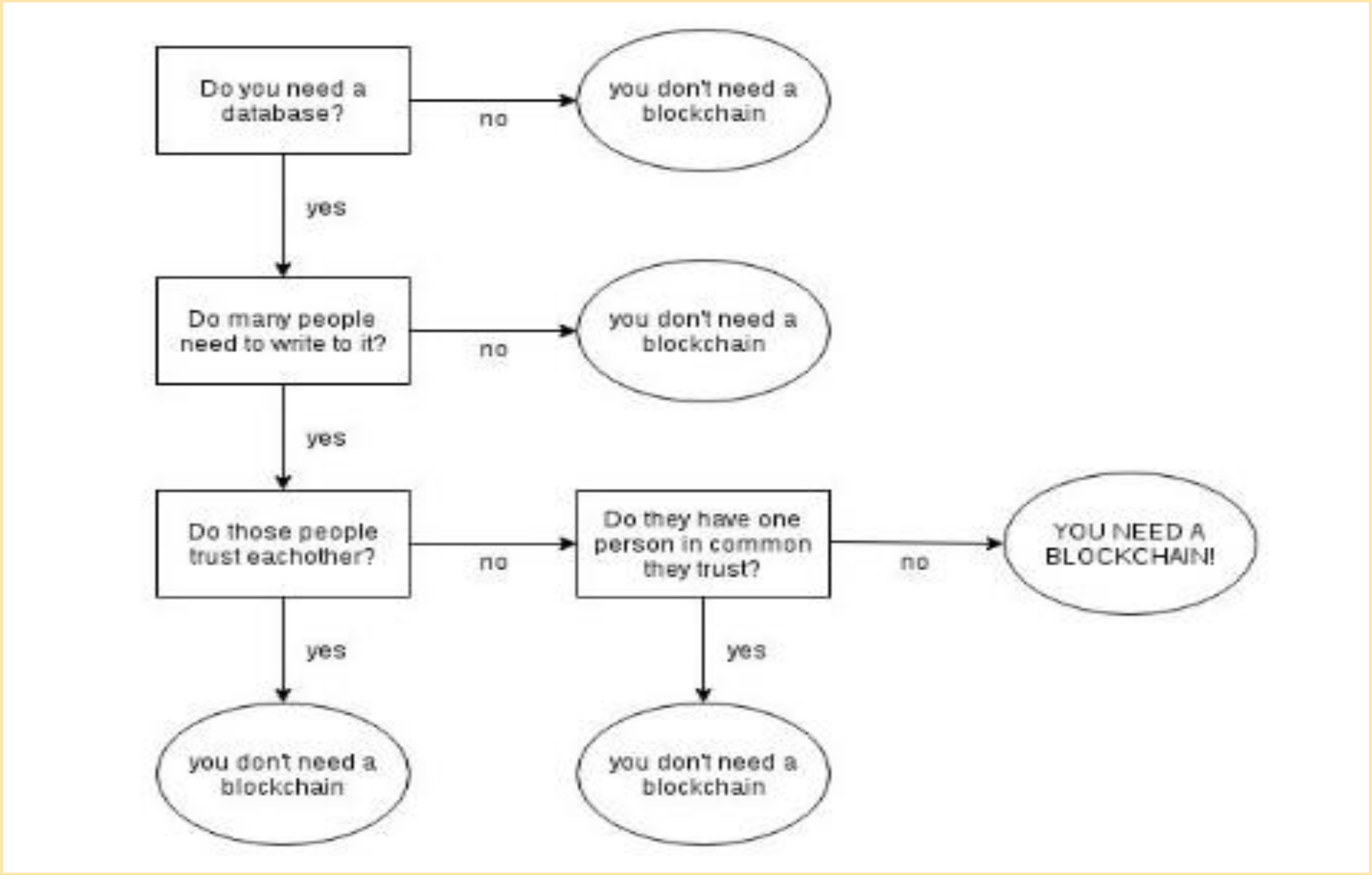
University of Reading

Why does Blockchain exist?

- By spreading its operations across a network of computers, blockchain technology allows transactions of digital assets to operate **without** the need for a central authority.
  - This is known as **decentralisation**

- Benefits of **decentralisation** include:
  - Reduces risks;
  - Eliminates or minimises processing and transactional fees.

When is Blockchain needed?


Do you need a database?
no
you don't need a blockchain
yes
Do many people need to write to it?
no
you don't need a blockchain
yes
Do those people trust eachother?
no
Do they have one person in common they trust?
no
YOU NEED A BLOCKCHAIN!
yes
you don't need a blockchain
yes
you don't need a blockchain

Blockchain**Hubs**

JS map by amCharts

Berlin based Hub in 2015

Now Network of Decentralised Hubs

Sharing the same vision and goals.
Sharing resources, but stay independent.

Internet of Value

Internet of Information

TCP/IP = **communication protocol**

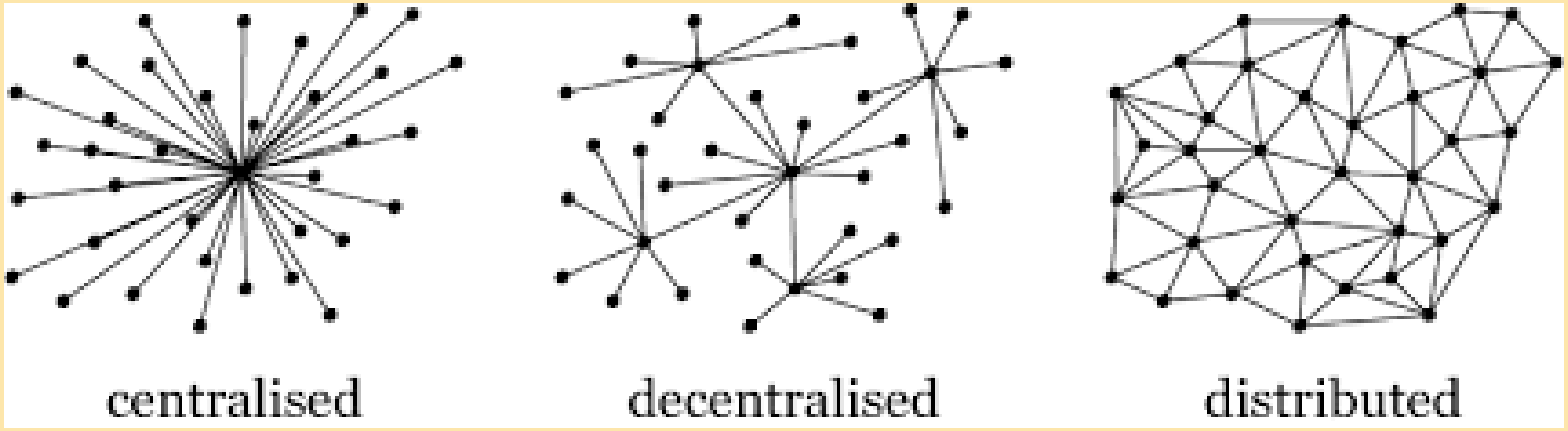
Distributed Ledger = **value exchange protocol**

Smart contracts for **peer to peer** transactions.

Crypto Currency distributes trust

No Unique point of failure

Points of system control & failure



Distribute Authority and Smarten up
Contracting & Compliance Assurance
Computers can verify or auto-enforce any type of business transaction or legal agreement
Decentralised Application dAPP
- long lived process on the block chain

Intermediation, certification, security

Democratising Trust

Through Parallel Computing Consensus

Nodes can collaborate/transact

No Need for Trusted 3rd party i.e. a central authority

---Block Chain, Distributed Ledger

Latest Evolution in Distributed Systems Architecture

Shared, Trusted, Public, Ledger of Transactions
Open to inspection by all
Controlled by all
No single entity controls

Shared: Every P2P node a stateful client & server

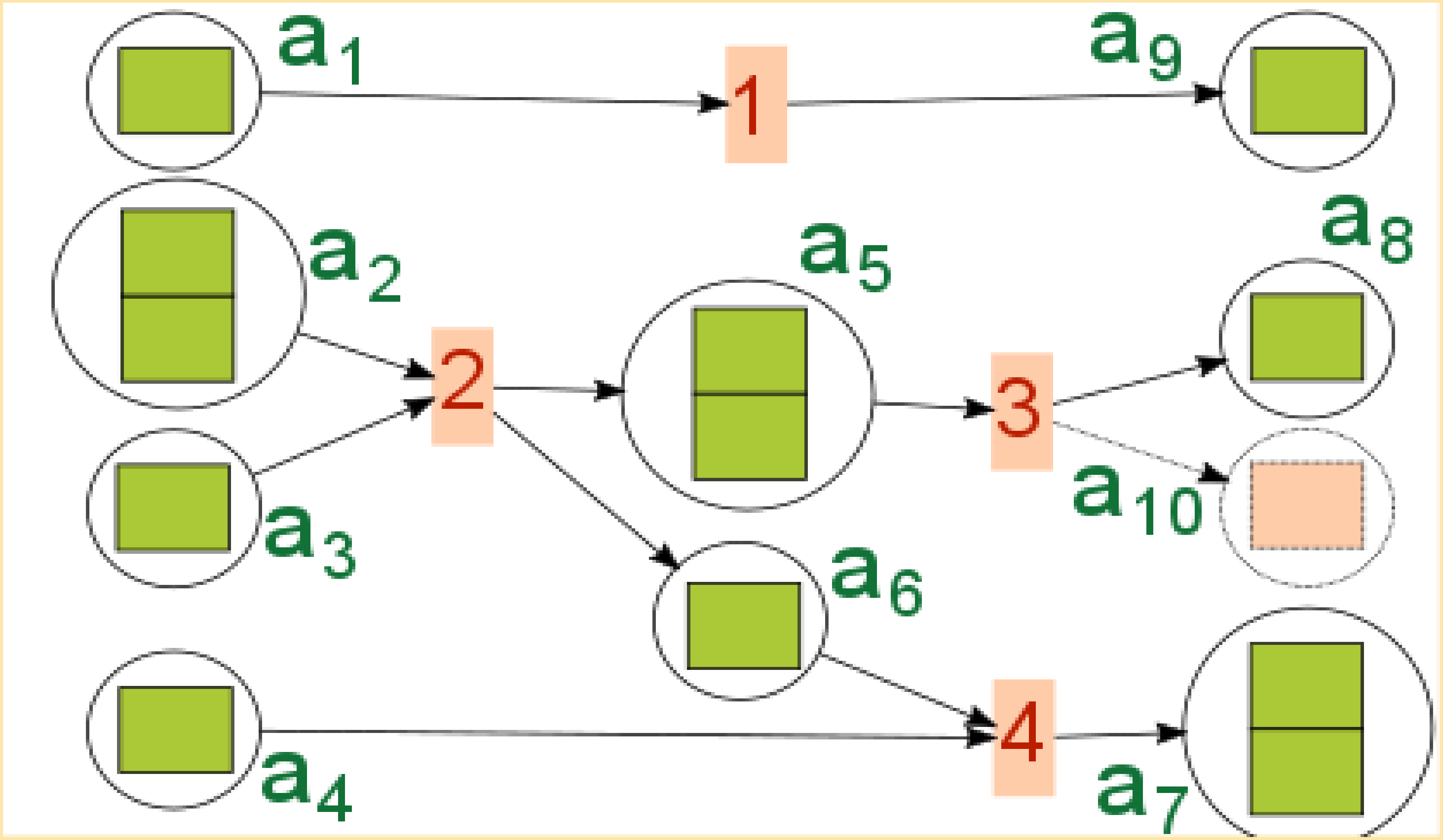
Trusted: Each node incentivised to verify the transactions

Public Ledge of Transactions Cryptography (Distributed Merkle Hash Trees) for security as well as privacy

Permission/Verification

By Consensus instead of Trusted 3rd Party
Each transaction on the blockchain is verified by all nodes in the network to be true or false?

If the majority of the nodes in the network do agree that a certain transaction is true, it is then validated and a new block is created and added on “top” of the chain.



Four basic type transactions among 10 addresses. Each green rectangle can be thought of as one Satoshi. Transaction 3 sends funds to a10 for transaction fees. Other transactions do not pay fees.

Decentralisation ...been trending long while...

Blockchain (Ethereum, Private BCs Bitcoin

Ipfs, swarm (decentralised file storage)

Whisper (decentralised messaging)

SoLiD (Social Linked Data) IPDB, BigchainDB

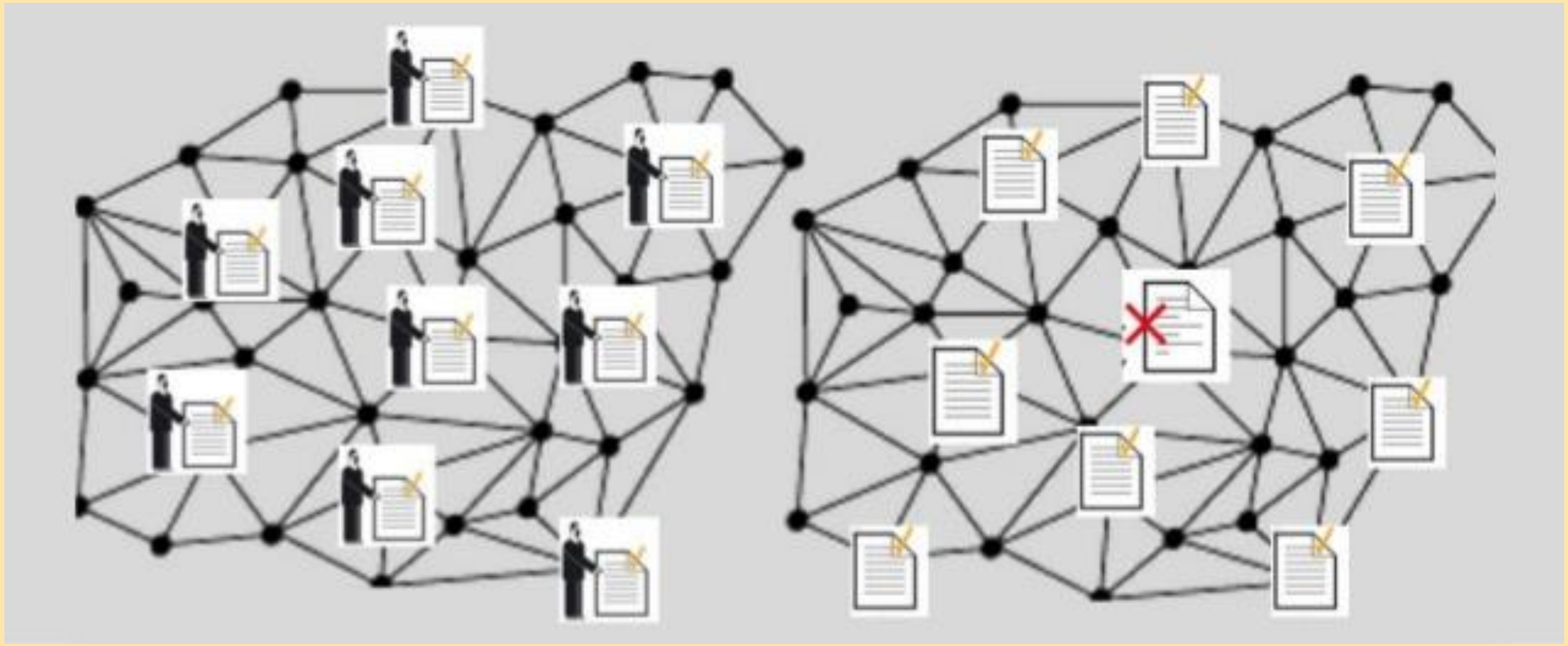
IPFS: The InterPlanetary File System is a peer-to-peer hypermedia protocol designed to make the web faster, safer, and more open

SoliD Derived from "**social** linked data") is a proposed set of conventions and tools for building decentralized **social** applications based on Linked Data principles ..

IPDB The Internet Pinball Machine Database

BigchainDB The blockchain database.

Distributed Encrypted Stateful Client Server Security



to tamper with the system need to manipulate the state (date, other transaction data) on the majority of blocks and the older the date, the more blocks have to be manipulated back in time – almost impossible to break the encryption.

Internet of Block Chains

Dedicated Applications:

Bitcoin (1 Smart Contract) –first mover

Altcoins (Experimental stage) Single purpose Forks of Bitcoin

Private/Consortium Blockchains Not Blockchains in the traditional sense, because not censorship resistant. Rather distributed databases

Framework Architecture for general purspose smart contracts

Ethereum (Smart Contract OS) Universal Operating System where you can build any type of smart contract

Ethereum: Distributed Crypto Economy Crypto Law

- **Not Just transferring value but also** verifying compliance with the contract.
- **Virtual machine** running any kind of **smart contract** or **dApp**
- **Ethereum contracts** –multilingually capable - compilation into bytecode for deployment to block chain through the Ethereum Virtual Machine

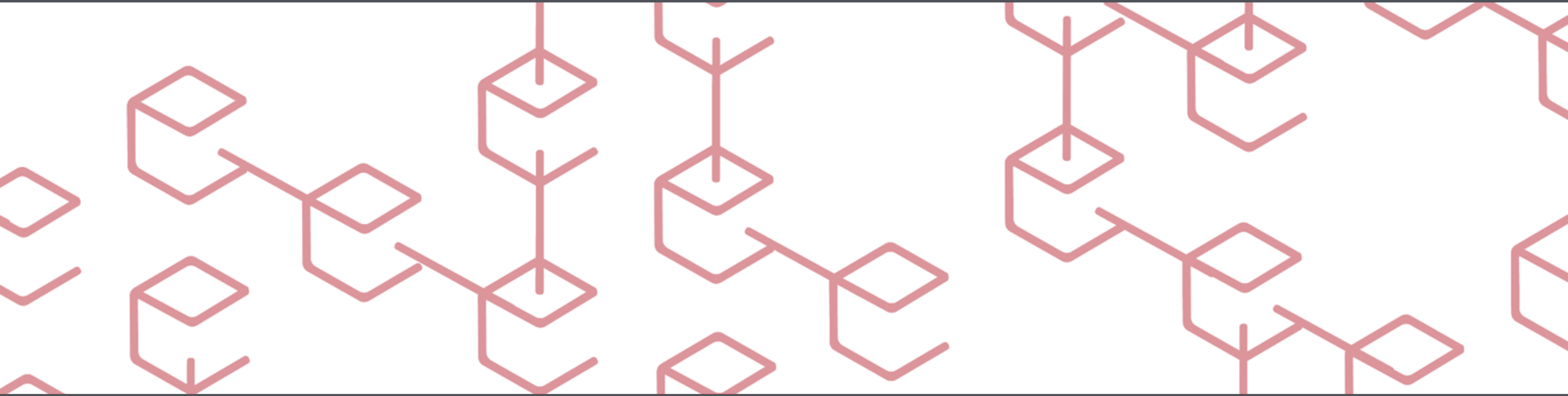
eGovernment

Self Sovereign Identity
Public Registries: Land registries, Marriage Certificates, Transparency
Participation, eVoting

Online Notaries
Registers of the ownership
Documents: Deeds etc
Property: Luxury goods, Works of art

University of Reading

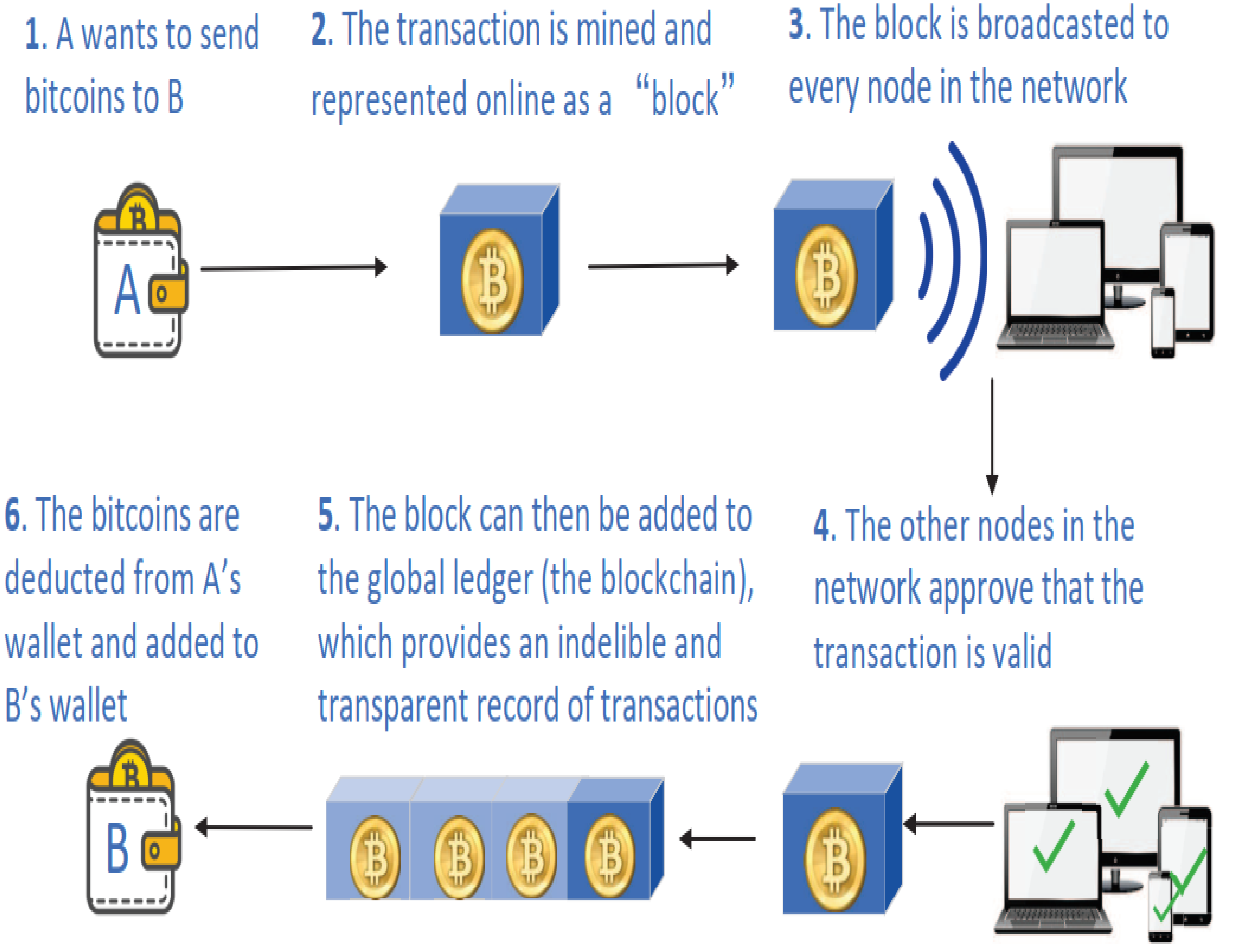
Introduction to Blockchain

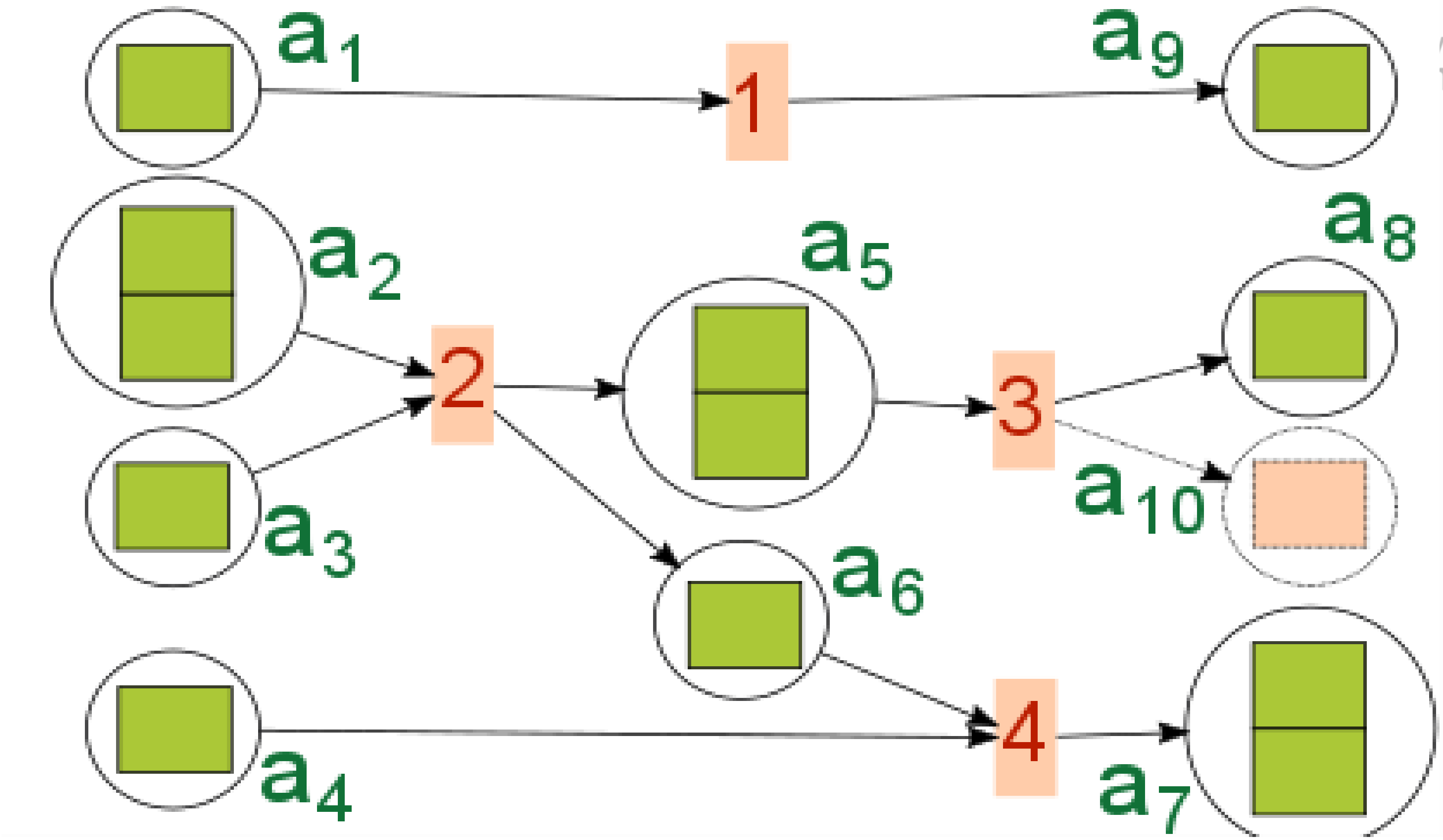


An introduction to the fundamental concepts of Blockchain – Why does it exist and what is it all about?

Ethereum Technology Stack Decouples the Smart Contract Layer from the Consensus Layer (unlike Bitcoin)

Relations	**Smart Contracts** **Application Layer**
Assets	Record of Transactions Blockchain Layer
Governance	Consensus Rules Blockchain Layer
Network	P2P Network Blockchain Layer
Infrastructure	TCP/IP Internet Layer


1. A wants to send bitcoins to B
2. The transaction is mined and represented online as a "block"
3. The block is broadcasted to every node in the network
4. The other nodes in the network approve that the transaction is valid
5. The block can then be added to the global ledger (the blockchain), which provides an indelible and transparent record of transactions
6. The bitcoins are deducted from A's wallet and added to B's wallet
A
B



Four basic type transactions among 10 addresses. Each green rectangle can be thought of as one Satoshi. Transaction 3 sends funds to a10 for transaction fees. Other transactions do not pay fees.

Emerging Developments

Banking

Blockchains as record of who owns what instead a series of internal ledgers

Bitcoin Blockchain

- Minimum Transaction Costs
- Faster
- Censorship resistant
- More secure

According to Santander Bank **Fintech** could save banks up to $20 billion by 2022

Energy

Energiewende in Deutschland

- P2P Energy Trading
- Smart Grids
- Mobile Charging

IoT

- User Access Control for machine to machine smart contracts

Accounting Revolution

In Auditing & Compliance

- On the fly, no after the fact
- **Cooking the Books?**
- **Less human errors**
- **Faster taxation cycles**

In Supply Chain Management

- Dis-intermediation
- Transparency
- Provenance

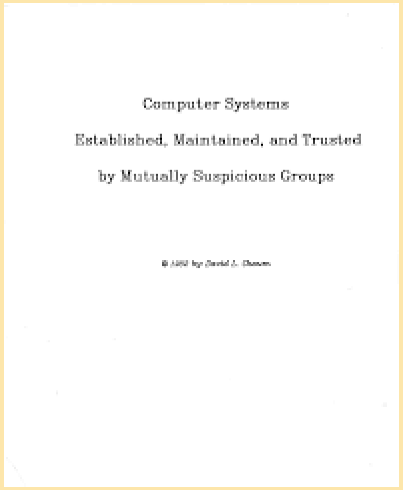
University of Reading

Distributed Technologies Current Challenges

- Scalability
- Identity & KYC
- Privacy
- Business models on Public Blockchains?
- Lack of Developers
- Regulatory uncertainties

History – Chaum

- Protocol first introduced in "***Computer Systems Established, Maintained, and Trusted by Mutually Suspicious Groups***." Chaum, D. (1982).



Computer Systems

Established, Maintained, and Trusted

by Mutually Suspicious Groups

© 1982 by David L. Chaum

" *A number of organisations who do not trust one another can build and maintain a highly-secured computer system that they can all trust (if they can agree on a workable design)… Cryptographic techniques make such systems practical."*

chaum.com/publications/chaum_dissertation.pdf

- **Vault system**
  - Each secured vault **signs**, **records**, and **broadcasts** every transaction on the network.

History – Chaum

- “Publicly permissioned”
    - Nodes need public authorisation
- Roles assigned to vault participants for **checks** and **balances**
    - **Passive watchers**
        - Explorers and analysts
    - **Doers**
        - Full nodes
    - **Executives**
        - Trustees responsible for signing blocks
    - **Czars**
        - Trustees empowered to change executives and policies

Philosophically Close to the governance of Bitcoin now

History – Haber and Stornetta

- Haber, S. and Stornetta, W. (1991) continued work on the protocol for the purpose of **tamper-proofing document timestamps**
- Later Haber, Stornetta and Bayer, D. (1992) incorporated **Merkle trees** to improve efficiency by allowing several document certificates to be collected into one block

History – Nakamoto

- Researchers have had difficulties in the process of identifying the permissionless character of the vault system.
  - Blockchain allows anyone to join and participate without requiring political hierarchies or reliance on a greater authority.

- The first **blockchain** was conceptualised by Nakamoto, S. (2008).
  - Implemented a year later as a core component of the cryptocurrency **bitcoin**.

Innovation in Distributed Architectures

- **World Wide Web** – Information Data Highway
  - Revolutionised Information
- **Web2**
  - Programmable Web
  - Social Media and Sharing Economy
  - P2P solutions with a Middle Man
- **Web3**
  - Decentral Web
  - P2P transactions without Middle Man

Overtime increasingly decentralised

Internet of- Value and Information

- **TCP/IP** = *Communication* Protocol
- **Blockchain** = *Value exchange* Protocol
    - Smart contracts for peer-to-peer transactions

Blockchain Use Cases

- **Cryptocurrencies**
  - A **digital asset** designed to work as a medium of exchange wherein individual coin ownership records are stored in a **ledger** existing in a form of computerised database using strong cryptography to secure transaction records, and to verify the transfer of coin ownership.
  - E.g. Bitcoin, Ether, Tether

  + Minimal Transaction Costs, Fast, Censorship Resistant, Secure

- **Smart Contracts**
  - A **transaction protocol** which is intended to automatically execute, control or document legally relevant events and actions according to the terms of a **contract** or an **agreement**.
    - E.g. Ethereum, Codius

Blockchain Use Cases (2)

- **Health Care**
  - Electronic medical records, insurance, etc.

- **Supply-Chain**
  - Blockchain can enable more transparency and accurate end-to-end tracking and tracing in supply chains.

- **Energy Trading**
  - Blockchain helps to distribute energy resources and efficiently balance supply and demand in smart grids.

- **E-Voting**
  - Electronic digital voting

Blockchain Use Cases (3)

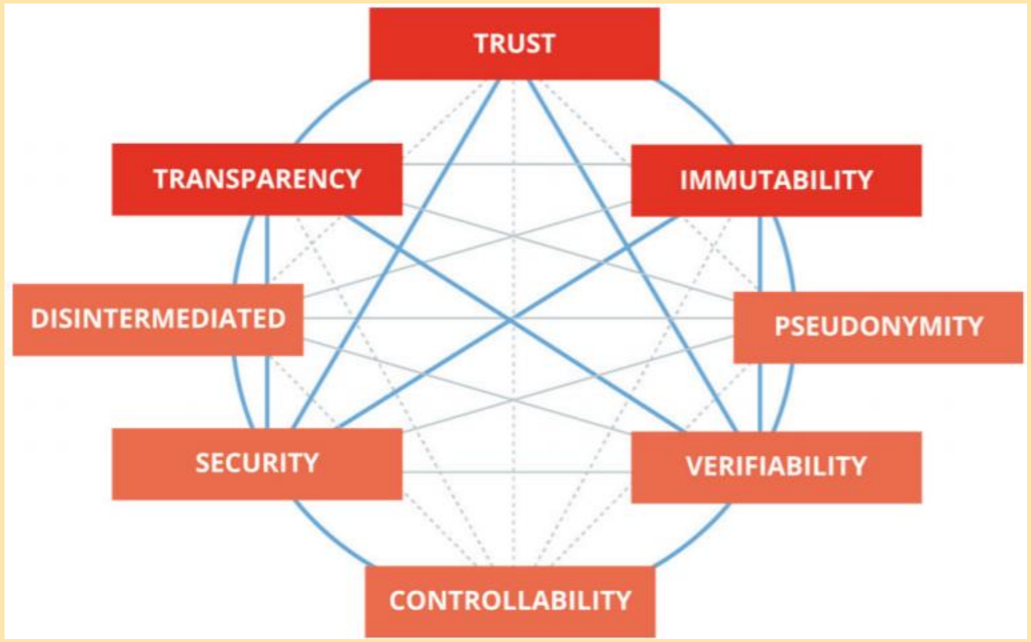
- Other **eGovernment** activities:
  - Self Sovereign Identity
  - Public Registries: Land Registries, Marriage Certificates

- **Online Notaries**
  - Registers of ownership
    - Documents: deeds etc.
    - Property: assets

- **Internet-of-Things (IoT)**
  - User access control for machine-to-machine Smart Contracts

Key attributes of blockchain

- *Unique attributes*
    - **Transparency**
    - **Trust**
    - **Immutability**
- *Other attributes*
    - Pseudonymity
    - Verifiability
    - Controllability
    - Security
    - Disintermediation



Key attributes of blockchain (2)

- **Transparency** → "Distributed Ledger"
  - Identical copies of all **valid** transactions (*the ledger*) is available to all participants at all times
- **Trust**
  - Strict governance rules, cryptography, and immutable transactions work together to provide strong security without the need for a centralised trust authority
- **Immutability**
  - Transactions recorded on a blockchain cannot be changed or removed:
    - New transactions are required to reverse effects of others.
    - There is no way to *expunge* the record of transactions

Key attributes of blockchain (3)

- **Pseudonymity**
  - Using **public** and **private** key systems, participants have a public-facing digital “**address**”
    - Not publicly associated with identities
    - Exercises uniqueness
    - Providing anonymity

- **Verifiability**
  - Transactions on a blockchain are immediately **auditable** in real-time
  - Complete record of transactions directly **verified**

Key attributes of blockchain (4)

- **Controllability**
  - The tracking of individual assets uniquely on a blockchain allows an individual to exercise **effective** and **exclusive** control over data and digital assets.
    - Transactions allow the secure transfer of control between individuals over the network.

- **Security**
  - The use of encryption algorithms combined with the disaggregation of data across a distributed network of nodes provides **security** against attempts to destroy or change the record of transactions.

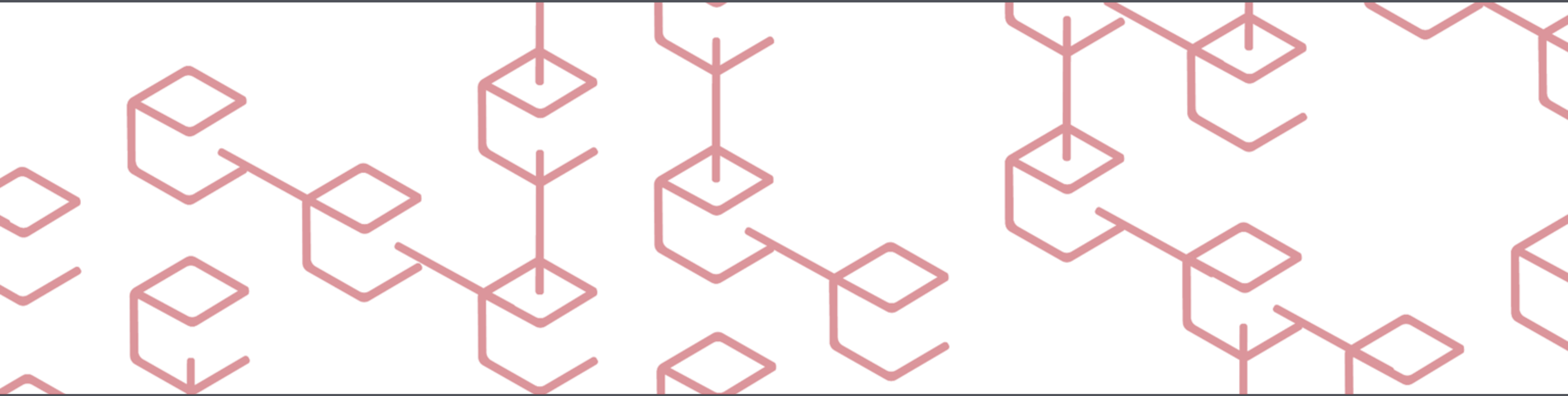
Key attributes of blockchain (5)

- **Disintermediation**
  - Using direct transactions, blockchain technology can streamline processes by:
    - Cutting out unnecessary intermediaries and process steps,
    - Reduce the risk of errors caused by additional transactions

Recap

- Topics Covered:
  - What is a Blockchain?
  - Why does Blockchain exist?
  - When is Blockchain needed?
  - A brief history of Blockchain and its evolution
  - Use-cases and examples
  - Key attributes of Blockchain

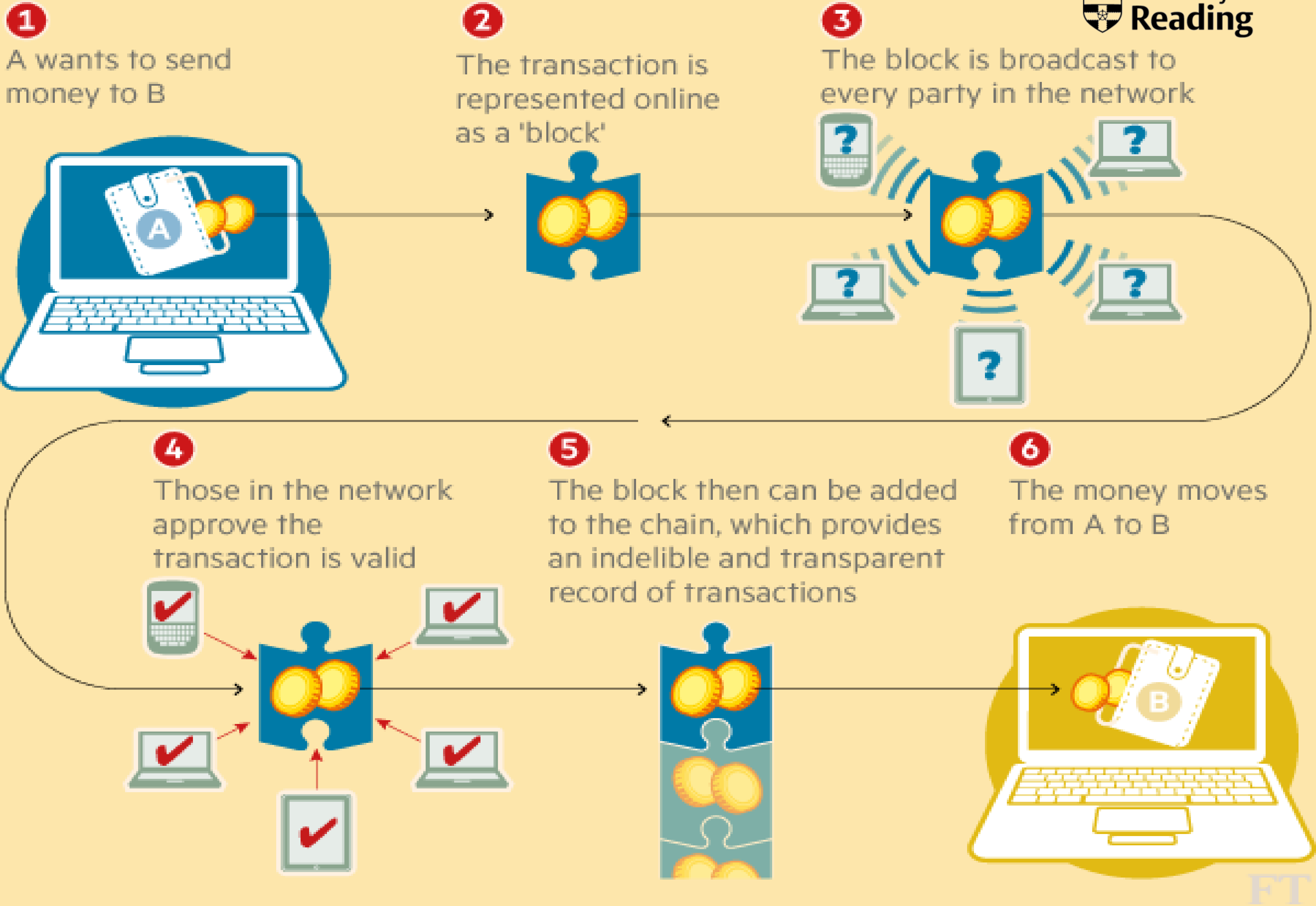
Fundamental Concepts of Blockchain



Transaction processing, hashing, asymmetric encryption and cryptographic digital signatures

How a blockchain works

University of Reading


1
A wants to send money to B
A
2
The transaction is represented online as a 'block'
3
The block is broadcast to every party in the network
?
4
Those in the network approve the transaction is valid
5
The block then can be added to the chain, which provides an indelible and transparent record of transactions
6
The money moves from A to B
B
FT

Transactions

- **Blocks** within a Blockchain can be used to store data
- Most commonly used as **ledgers**
  - I.e. **Transactions** are stored within blocks

- The Blockchain as a whole includes **all** transactions ever made
  - By reviewing all transactions, it is possible to derive the balance of any individual

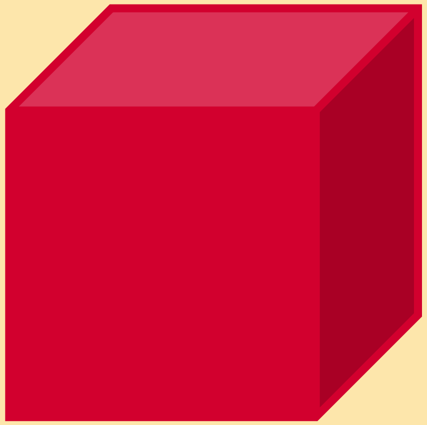
Transactions

- Blockchains are distributed on a **peer-to-peer** network
  - Requires **secure** transactions:
    - Ensure that:
      - > Senders are who they claim to be
      - > No one can fraudulently pose as someone else

  - *Solution*: **Cryptography**
    - Hashing
    - Digital Signatures

Simple Block Structure

- Varies slightly between implementations
- Common Features
  - **Header**
    - Version No.
    - Hashes
    - Timestamp
    - Difficulty
    - Nonce
  - **Body**
    - Transactions



Hashing

- **Anti-tampering** mechanism

- **All** data within a transaction block is hashed
  - If maliciously modified → hash altered

- “A hash function is any function that can be used to map data of arbitrary size to fixed size values.”

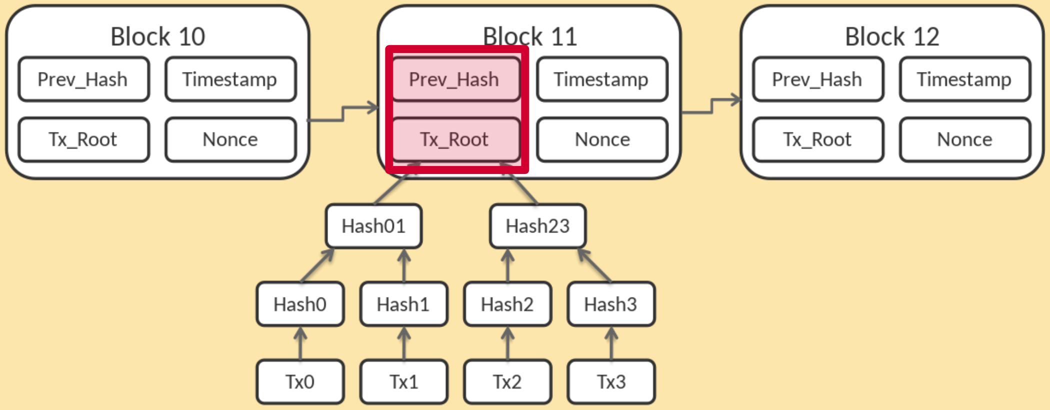
  - E.g. SHA-256 (Secure Hashing Algorithm)

Hashing Algorithm Properties

- Hashing algorithms **always** compute the **same** *hash* given the **same** *input*
  - **Pre-image resistance**
    - "For essentially all pre-specified outputs, it is computationally infeasible to find any input that hashes to that output; i.e., given y it is difficult to find an x such that $h(x) = y$"
  - **Second-preimage resistance**
    - "It is computationally infeasible to find any second input which has the same output as that of a specified input; i.e., given x, it is difficult to find a second preimage $x' \neq x$ such that $h(x) = h(x')$"

Hashing – Backwards Tampering

- In blockchains each block contains **two** hashes
  - The hash of the **current** block.
  - The hash of the **previous** block.



- Tx_Root = Current Block Hash
  - Merkle Root (Covered later)

Hashing – Backwards Tampering

- If an adversary chooses to change the hashes of blocks, they will fail due to the collision resistance property and the hash chain structure
    - Even if they can find an input with the same hash
        - Need to continue to change all previous hashes

Digital Signatures – Introduction

- A mathematical scheme for presenting authenticity of digital messages or documents.

- A **valid** *digital signature* gives recipients reason to believe:
  - **Authenticity**
    - Message created by claimed sender
  - **Non-repudiation**
    - Sender cannot deny having sent the message
  - **Integrity**
    - Message unaltered during transit

Asymmetric Encryption

- Asymmetric encryption uses two different yet related key pairs:
  - **Public** Key
  - **Private** Key
- Keys are mathematically related in a way that a computer can confirm but cannot feasibly guess the private key belonging to a public key

- Typically, **public** keys are used for **encryption** and **private** keys for **decryption**

Digital Signatures

- Using the digital key pairs generated:
  - Data is signed → "**Digital Signature**"
- Prevents malicious actors from signing transactions as someone else
  - Unable to create a valid transaction **without** private key

Digital Signatures (2)

- Key pair owned by X where pub_x and $priv_x$ are public and private keys respectively
  - X generates a transaction T
  - X then digitally signs T with $priv_x$
- Node N picks up T
  - Verifies the signature using pub_x

$$Digital\ Signature = sign(hash(T), priv_x)$$

$$hash(T) = verify(Digital\ Signature, pub_x)$$

Elliptic Curve Digital Signature Algorithm

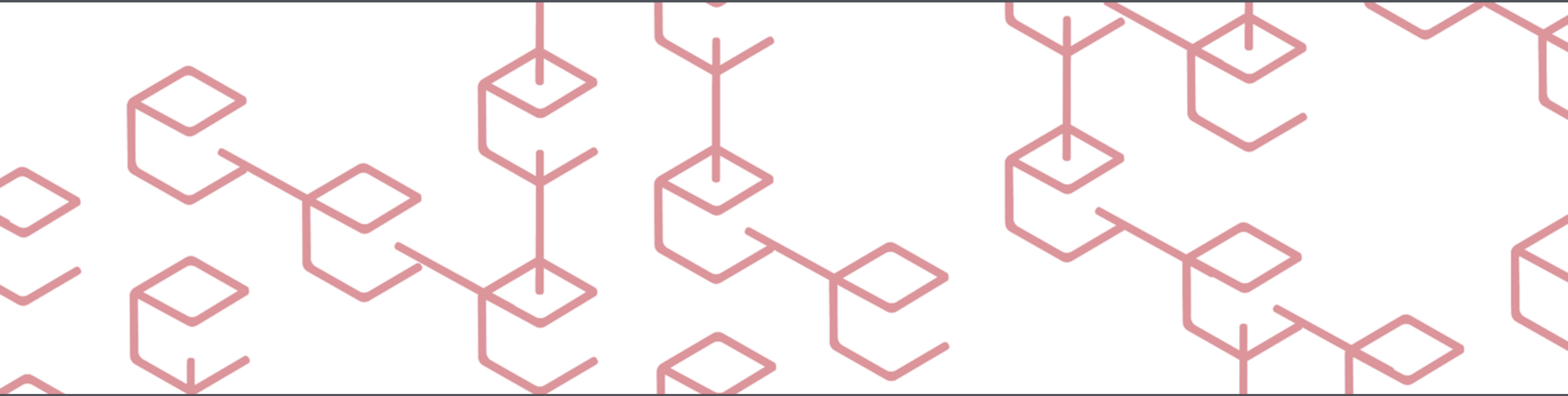
- In Bitcoin and other blockchains, **Elliptic Curve Digital Signature Algorithm** (ECDSA) is used
- Elliptic Curve Cryptography (ECC) is the next generation of public key cryptography
  - Provides a significantly more secure foundation than standard systems such as RSA
  - Complicated Mathematical Foundations
    (Could be an entire course on its own)
    - Elliptic Curves
      > A set of points satisfying a specific mathematic equation

      > Better trapdoor functions

Recap

- Topics Covered
  - Transaction and Block Structure
  - Security
    - Hashing
    - Digital Signatures
      - >Asymmetric cryptography
        - -Elliptic Curve Digital Signature Algorithm

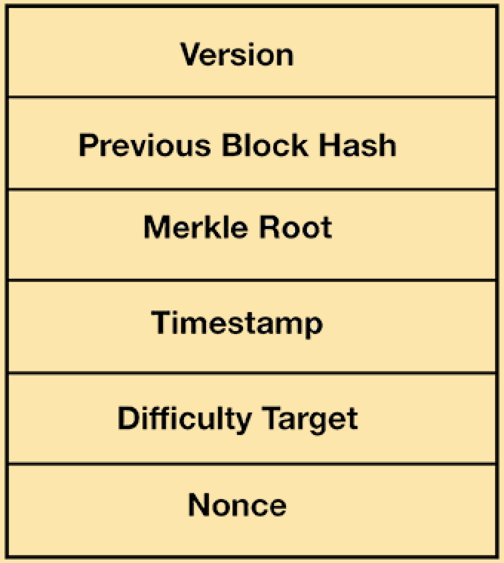
Blockchain Concepts (Continued)



Blocks, nodes, networks and consensus algorithms

Blocks (Recap)

- Blockchain **ledger** consist of linear structure of connected **blocks**
- Bitcoin block structure is **large**
  - Large amount of *metadata*



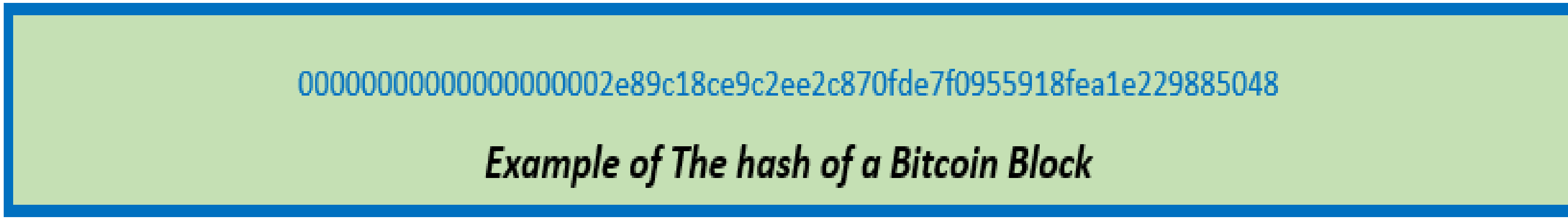
Block Structure

- The Ethereum White Paper details a block structure that can be simplified into the following elements:
  - **Index** (Height of the block)
    - The position of the block within the blockchain
  - **Timestamp**
    - Reference point for when the block was created
  - **Hash**
    - A hash of all the information in the block
  - **Hash of previous block**
    - Connects the latest block to the last block on the chain
  - **Nonce**
    - Number only used once – for Proof-of-work (PoW)
  - **Extra Nonce**
    - An extra nonce to prevent duplicated work

Block Structure (Continued)

- **Difficulty**
  - The difficulty of the PoW; corresponds to the resulting hash of the block
- **Reward**
  - Number of coins rewarded for mining the block
- **Merkle Root**
  - Generalisation of the hash list of transactions in the block

- **List of x transactions**
  - Blocks include a number of transaction



00000000000000000002e89c18ce9c2ee2c870fde7f0955918fea1e229885048

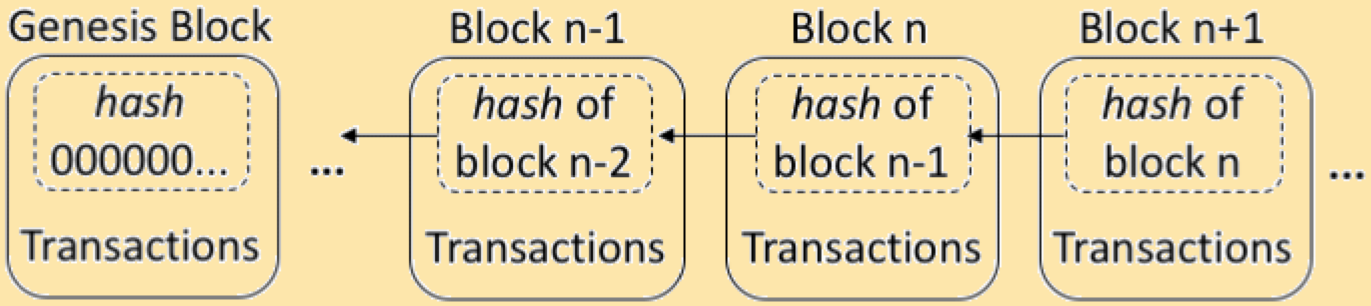
Example of The hash of a Bitcoin Block

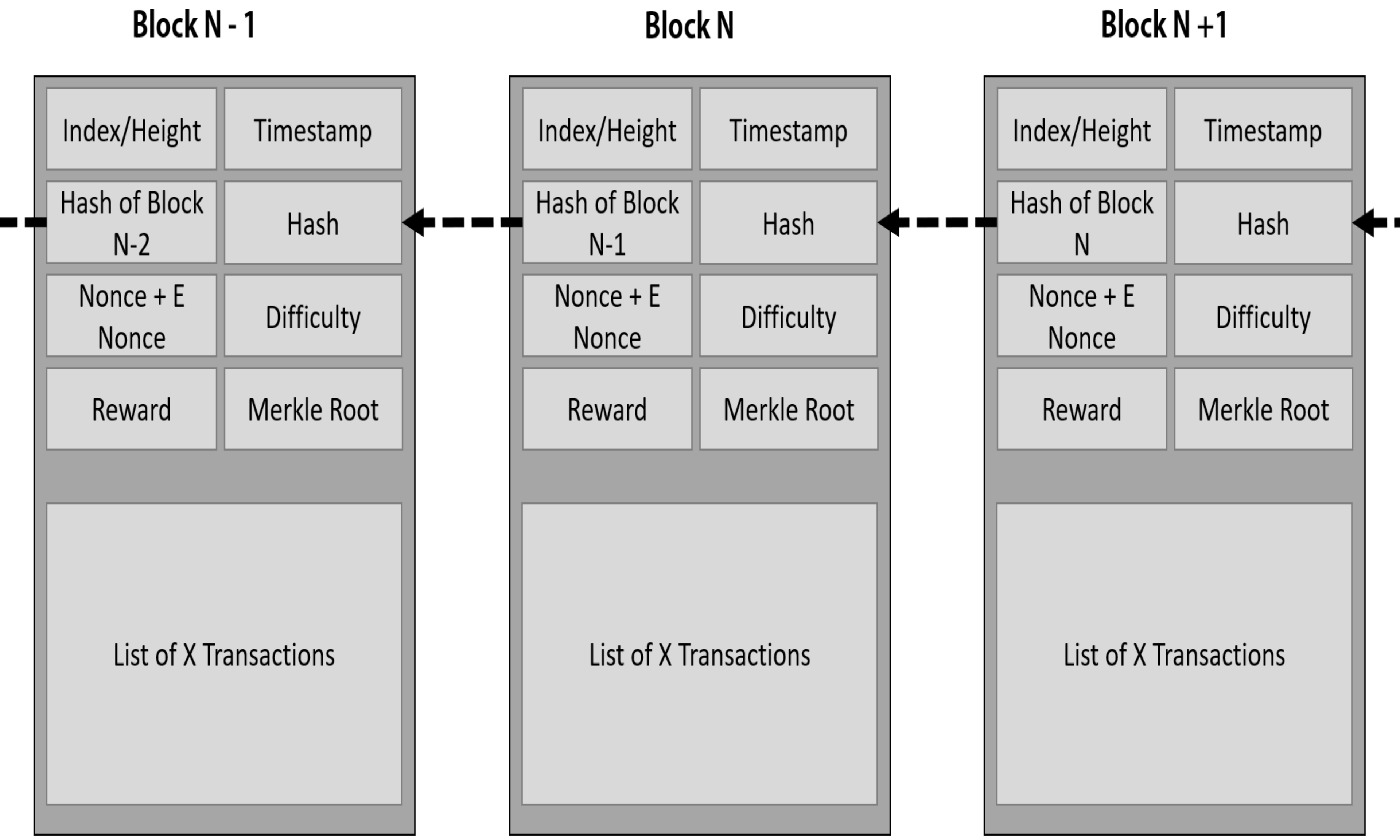
The Genesis Block

- The first block in the blockchain is known as the **Genesis Block**.
  - Its values are normally fixed
  - The rest of the chain then builds upon this block

- *Any chain that starts with a block that is not identical to the genesis block is invalid*

Blockchain – The Chain




Block N - 1
Block N
Block N +1
Index/Height
Timestamp
Hash of Block N-2
Hash
Nonce + E Nonce
Difficulty
Reward
Merkle Root
List of X Transactions
Index/Height
Timestamp
Hash of Block N-1
Hash
Nonce + E Nonce
Difficulty
Reward
Merkle Root
List of X Transactions
Index/Height
Timestamp
Hash of Block N
Hash
Nonce + E Nonce
Difficulty
Reward
Merkle Root
List of X Transactions

Hashing (Revisited)

- *“A hash function is a function that can map any data of an arbitrary size to a hash of a fixed size”*
  - Unfeasible to derive inputs of a function from a hash
  - Slight change in input results in drastically different hash
  - Collision resistance
    - Impossible (*or a least very difficult*) to find any two distinct input which compute the same output

- Hashing allows trustless design
  - Blocks and transactions can easily be verified using hashes along with consensus algorithms
    - E.g. Proof-of-Work, Proof-of-Stake etc.

Consensus Algorithms – Introduction

- **Consensus** = “A general agreement”
- **Process** used to achieve **agreement** on a single value among distributed processes or systems
- Designed to achieve **reliability** in a network involving multiple **unreliable** nodes
- Example: Proof-of-Work

Hashes in Proof-of-Work (PoW)

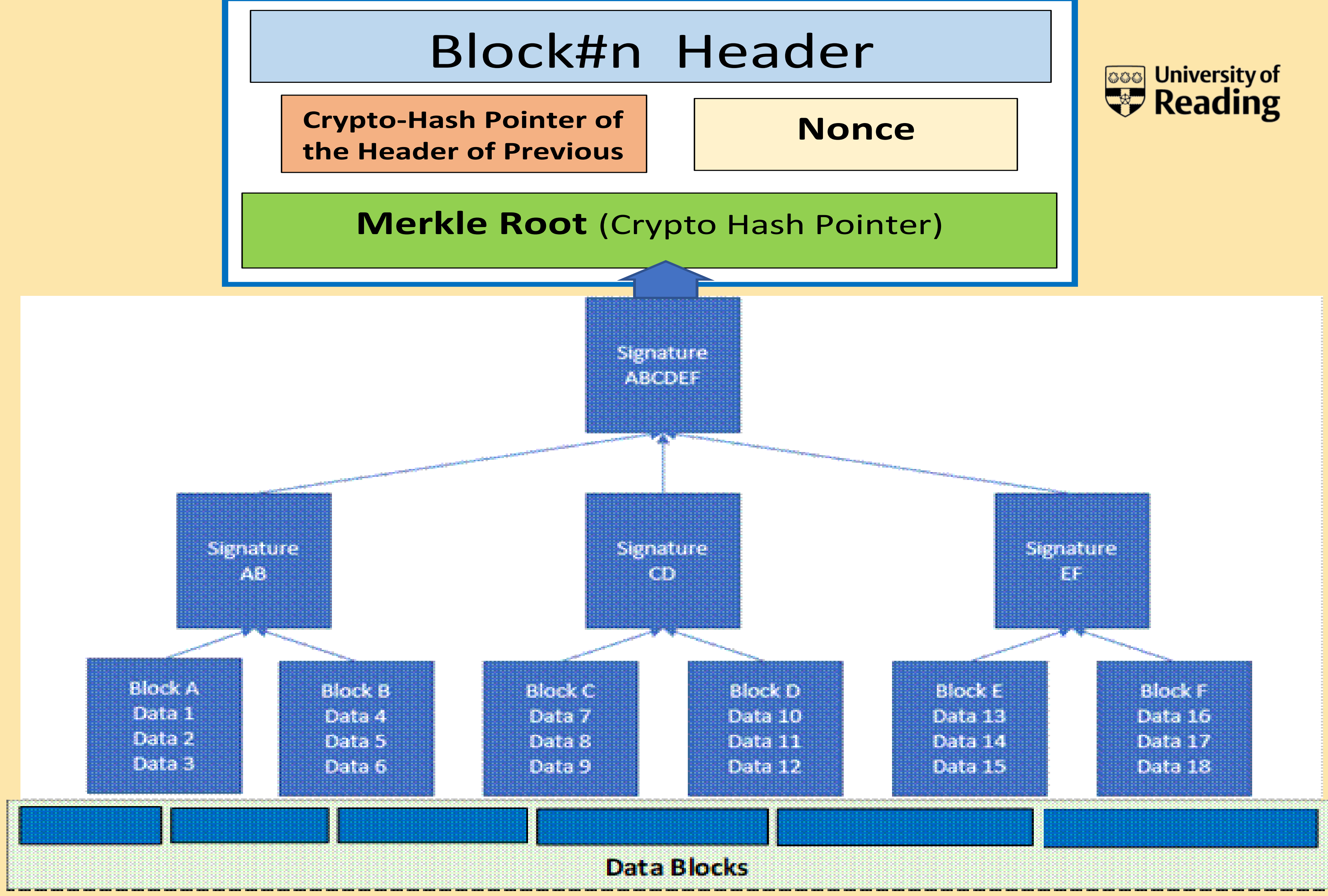
- Hashes computed by the PoW algorithm are an essential component of the block
- Entire block is hashed
  - If any information is changed the block is rejected
- Recomputing the hash is not an easy task
  - PoW done to create the initial hash must be repeated.
  - Not worthwhile
    - All information independently verified

Hashes in PoW (2)

- Each block includes the hash of previous block
  - Incorporated in all subsequent hashes

- Creates a chain of immutable blocks
  - Each block will refer to the previous block
  - Changing the previous hash in a block means all following block hashes must be recomputed

- This chaining plays a vital role in making Blockchains **immutable** and **irreversible**


Block#n Header
Crypto-Hash Pointer of the Header of Previous
Nonce
Merkle Root (Crypto Hash Pointer)
Signature ABCDEF
Signature AB
Signature CD
Signature EF
Block A Data 1 Data 2 Data 3
Block B Data 4 Data 5 Data 6
Block C Data 7 Data 8 Data 9
Block D Data 10 Data 11 Data 12
Block E Data 13 Data 14 Data 15
Block F Data 16 Data 17 Data 18
Data Blocks

Merkle Tree Linking Block Transactions (Digital Signatures) to Block Tree Merkle Root (Crypto-Hash Pointer)

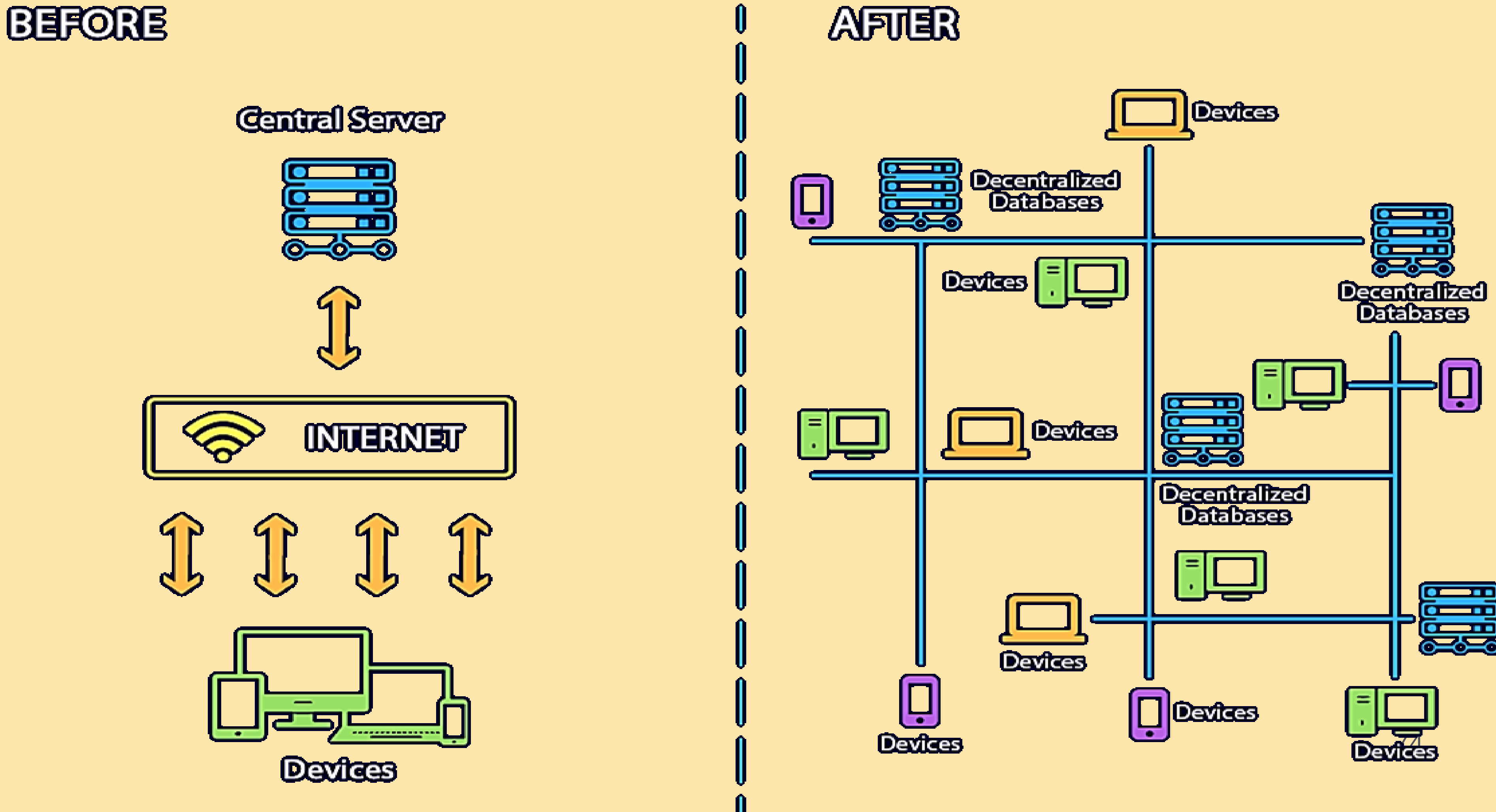
Network Structures (Recap)

- **World Wide Web** – Information Data Highway
  - Revolutionised Information
- **Web2**
  - Programmable Web
  - Social Media and Sharing Economy
  - P2P solutions with a Middle Man
- **Web3**
  - Decentral Web
  - P2P transactions without Middle Man

Overtime increasingly decentralised

Network Structures

Centralised vs. Decentralised Internet



Centralisation – Overview

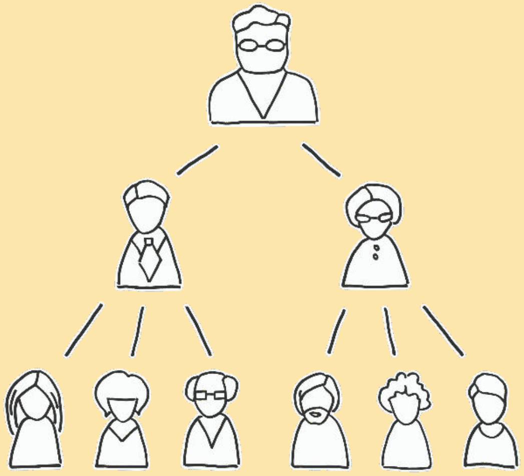
- In a **centralised** network:
  - **central authority** *governs* and *handles* network
  - Traditional Technologies e.g. social networks

- **Advantages**
  - Command Chain
  - Reduced Costs
  - Quick decision implementation

- **Disadvantages**
  - Trust
  - Single Point of Failure
  - Scalability Limitation

Centralisation – Advantages

- **Command Chain**
  - Clearly defined;
    - Every party knows their role and who to report to
    - Easy delegation
      - > If work is successfully completed it creates a level of trust among the workers and chain, improving upon the confidence required to make it work



Centralisation – Advantages

- **Reduced Costs**
  - Centralised networks and infrastructure requires less support and lower operational costs

Centralisation – Advantages

- **Quick Decision Implementation**
  - Fewer nodes require less communication among the different levels of authorisation.

Centralisation - Disadvantages

- **Trust**
  - Centralised organisations are *theoretically* secure and trustable.
    - *Trust is an agreement that is set by the service provider and the user*
      - > *Agreements can easily be broken*
      - > *Lapses in security can occur leading to vulnerabilities*

Centralisation - Disadvantages

- **Single point of failure**
    - Centralisation results in a single-point of failure

Centralisation - Disadvantages

- **Scalability Limitation**
  - As a single server is used in most cases, this leads to scalability limitations

Decentralisation

- In a decentralised network, there is **no** central authority that governs and handles the network. e.g. Blockchain network

- **Advantages**
  - User has Full Control
  - Data cannot be altered or deleted
  - Secure
  - No Censorship
  - Open Development
  - Transparency
  - Immutability etc

- **Disadvantages**
  - Conflict
  - Cost
  - Crime
  - Volatility
  - Accountability Problematic

Decentralisation – Advantages

- **Full Control**
  - Users in full control of their transactions
  - Verification process not dependent on third-parties
    - Instead utilises consensus methods to verify information

University of Reading

Decentralisation – Advantages

- **Data Immutability**
  - Strictly append-only.
  - No chance for anyone to modify or alter data once stored

Decentralisation – Advantages

- **Secure**
  - Highly secure due to how data and transactions are handled using cryptography

Decentralisation – Advantages

- **Censorship**
  - Less prone to censorship as peers interact directly

University of Reading

Decentralisation – Advantages

- **Open Development**
  - Collaborative Efforts
  - Enables advancements

Decentralisation – Disadvantages

- **Conflict**
  - Conflicts can occur if not well maintained

Decentralisation – Disadvantages

- **Cost**
  - Requires a complex communication network across many systems

Decentralisation – Disadvantages

- **Crime**
  - Transactions are essentially anonymous leading to misuse

Decentralisation – Disadvantages

- **Volatility**
  - Decentralised cryptocurrency shows volatile behaviour where prices fluctuate a lot

University of Reading

Centralisation vs. Decentralisation

	Centralisation	Decentralisation
Third-party Involvement	Yes	No
Control	Full control stays with the central authority	Control stays with the users
Hackable	More prone to hacks and data leaks	Less prone to hacks
Single Point of Failure	Yes	No
Ease of Use	Intuitive and easy-to-use	Not easy-to-use
Exchange Fees	High fees	Lower fees
Anonymous	Users are not anonymous	Offers potential for anonymity

Peer-to-peer (P2P) Networks

- Technology based on **decentralisation**
- P2P architecture allows transactions to be **worldwide** without requiring intermediaries (or central servers)
- With the distributed P2P network:
  - Anyone wishing to participate in the process of **verifying** and **validating** blocks can set-up nodes

P2P Networks and Blockchain

- Blockchain is a decentralised ledger tracking digital assets on a P2P network.
  - All nodes are connected:
    - Each maintains a complete copy of the ledger
    - Compares with others to ensure data is accurate

How does the network come to an agreement?

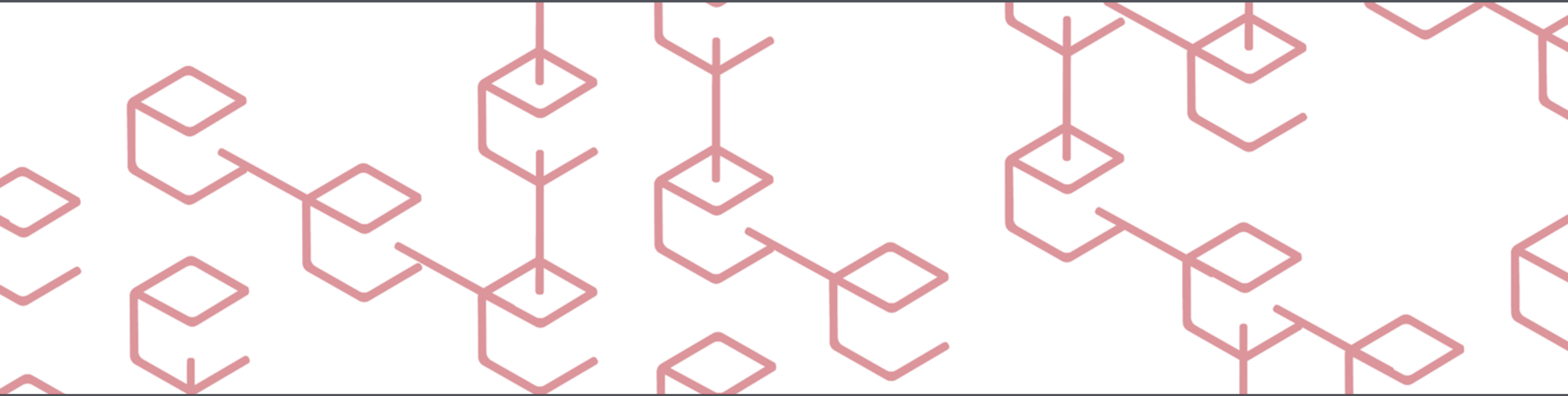
- Consensus protocols and algorithms
  - A set of computers wanting to reach an agreement concerning a single decision i.e. transaction validity

- Covered in detail in the next lecture

Recap

- Topics Covered
  - Blocks and Blockchain
  - Nodes and Networks
  - Brief Introduction of Consensus Protocol

Consensus Protocols



Consensus reaching using variants of Proof-of-Work and Proof-of-Stake algorithms amongst others

University of Reading

Consensus Algorithms

- In an online blockchain which is distributed across nodes in a peer-to-peer network, the nodes must reach a consensus on how the next block should be added to the chain.
  - In Bitcoin and other existing cryptocurrencies, Proof-of-Work is utilised
- Proof-of-Work dictates that for a new block to be added to the chain, this block must have a hash that satisfies a given difficulty threshold

In PoW nodes compete to create such a block

Proof-of-X

X= Work, Stake, Authority, Elapsed-Time, Sleepiness

Other Consensus Algorithms

- **Proof-of- Elapsed-time (POET)**
  - Each participating node is required to wait for a randomly chosen time period, and the first one to complete the designated waiting time is permitted to generate a new block.
  - Developed by Intel Corporation that enables permissioned blockchain networks to determine block winners and mining rights. ...
  - The node with the shortest *wait* **time** will win the block, thus being allowed to commit a new block to the blockchain. Adopted by Hyperledger (Sawtoolh Model Franework)

- **Proof-of-Authority (POA)**
  - Only validators have the right to approve transactions and new blocks. A participating node earns reputation to their identity and only when the reputation is accumulated, the node can become a validator.
  - Iincentive to retain the position of validator as gained. By attaching a reputation to identity, validators are incentivized to uphold the transaction process, as they do not wish to have their identities attached to a negative reputation.

Other Consensus Algorithms

- **Reputation**
  - An extension of PoA in which reputation is accumulated and calculated using pre-defined formulas.
  - In PoR, we let reputation serves as the incentive for both good behavior and block publication instead of digital coins, therefore no miners are needed. We also implement a prototype and our scalability experiments show that our protocol can scale to over a thousand participants in a peer-to-peer network with throughput of hundreds of transactions per second.

- **Sleepy Consensus**
  - Nodes have two modes (Awake and Asleep) and participants can freely change status during protocol execution. Effective when the majority of participants are honest. distributed protocols in a “sleepy” model of computation where players can be either *online* (awake) or *offline* (asleep), and their online status may change at any point during the protocol.Requires the majority to be trusted to be honest.

Proof-of-Work – Introduction

- When a transaction is initiated, the transaction data is fitted into a block with a maximum capacity of 1MB, and is then duplicated across multiple nodes on the network
- Nodes are the administrative body of the blockchain and verify the legitimacy of the transactions in each block.
- To carry out the verification step, nodes need to solve a computational puzzle, known as the Proof-of-Work problem.
- The first miner to decrypt each block transaction problem gets rewarded.
- Once a block of transactions has been verified, it is added to the blockchain.

Proof-of-Work: An example

- A node with a blockchain cannot add a new block until the hash begins with a number of zeros corresponding to the difficulty threshold.
  - For example with a difficulty threshold of 2 requires a hash $00x$ ...

- The node generates the hash: *57169c0619650ff122a8d74776e74a5e6b6e8e517aea48579b2be0af19440488*
- This does not satisfy the difficulty threshold; as a result it now must generate a new hash. *Note: hashing algorithms **always** generate the same hash given the same input.*
- To generate a different hash the block has an extra data property called a nonce (Number only used once) which is incremented after each attempt until the threshold is met.
- The node subsequently generates the hash:*008a78f40bb59bf4c9da8cfccf6a9c8e1202b01933d2363eabb277ad867ba738* satisfying the difficulty threshold and as a result shares the block to others on the network for acceptance.
- The block is appended to the blockchain across all nodes.

Proof-of-Work

- The difficulty in PoW corresponds to the number of zeros at the beginning of the hash that constitutes a valid block.
- The more zeros required the longer it will take to mine.
  - SHA-256 represents characters in hex format
    - 16 possible characters: $\{0-9,\ a-f\}$
    - Each extra-0 increases difficulty by a factor of 16

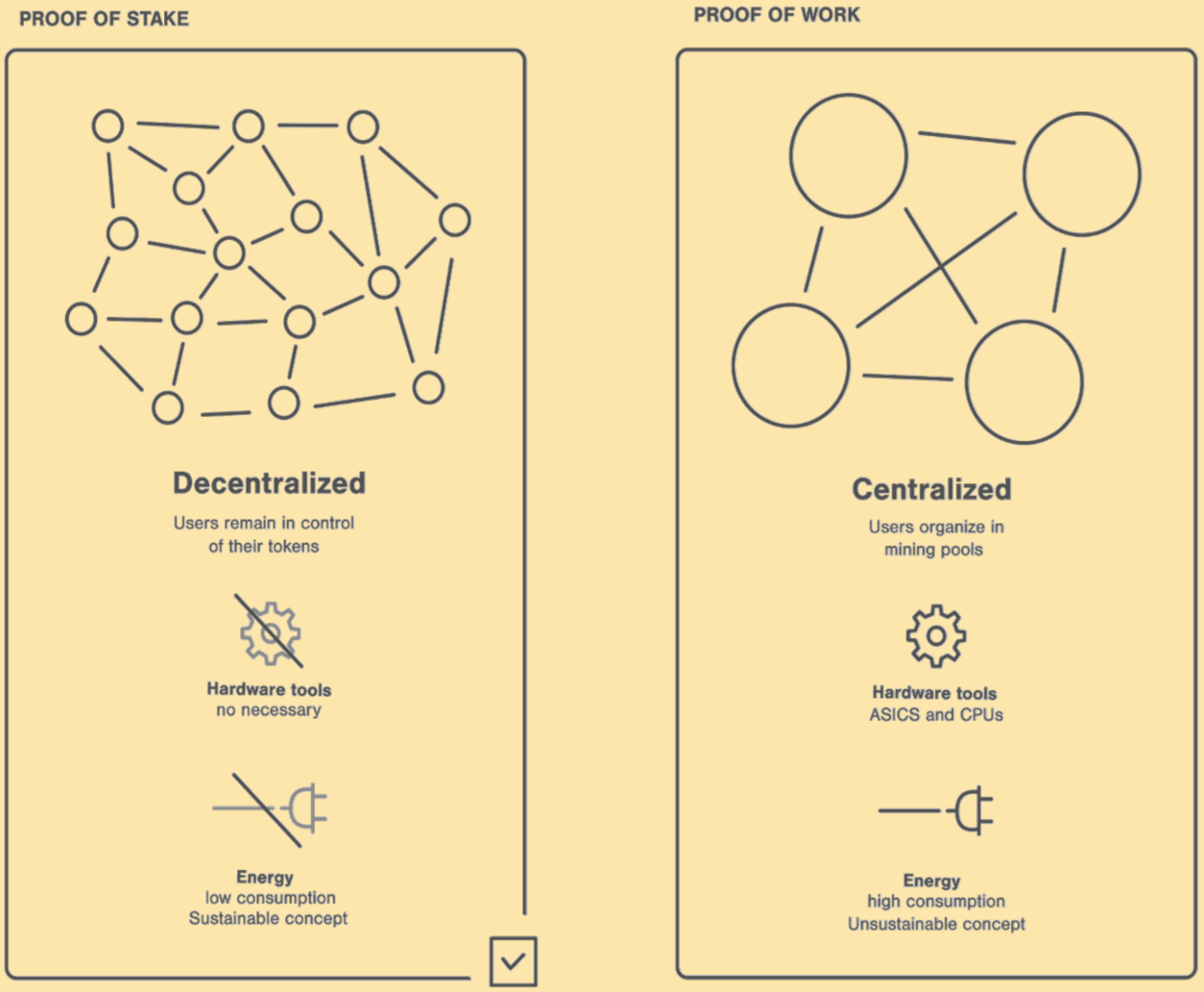
Proof-of-Work – Bitcoin Example

- Nodes across the world compete to be the next to generate a block in order to earn a reward
  - Rewards are gathered from transaction fees amongst other means
    - In other blockchains there is little-to-no incentives to mine
    - Fees are small sums added to each transaction to incentivise the mining node to choose the transaction in the next block.
      - > Higher fees increase the likelihood of mining of a block
- Bitcoin requires 19 0's at the beginning of the hash;
  - The chance of generating such a hash is $1/16^{19}$
  - As a result, this requires a huge amount of work to be done calling for specialised equipment and vast amounts of electricity

Proof-of-Stake

- Proof-of-Stake was created as an alternative to PoW to tackle inherent issues in the latter
- Mining using PoW requires a great deal of computing power in unlocking the computational challenges
  - It was estimated in 2015, that one Bitcoin transaction required the amount of electricity to power 1.57 American households per day
    - To foot the electricity bill, miners typically sell bitcoins for fiat money leading to the downward movement in the price of the cryptocurrency
- The Proof-of-Stake algorithm seeks to address this issue by attributing mining power to the proportion of coins held by a miner
  - Instead of utilising energy, a PoS miner is limited to mining a percentage of transactions that is reflective of their ownership stake

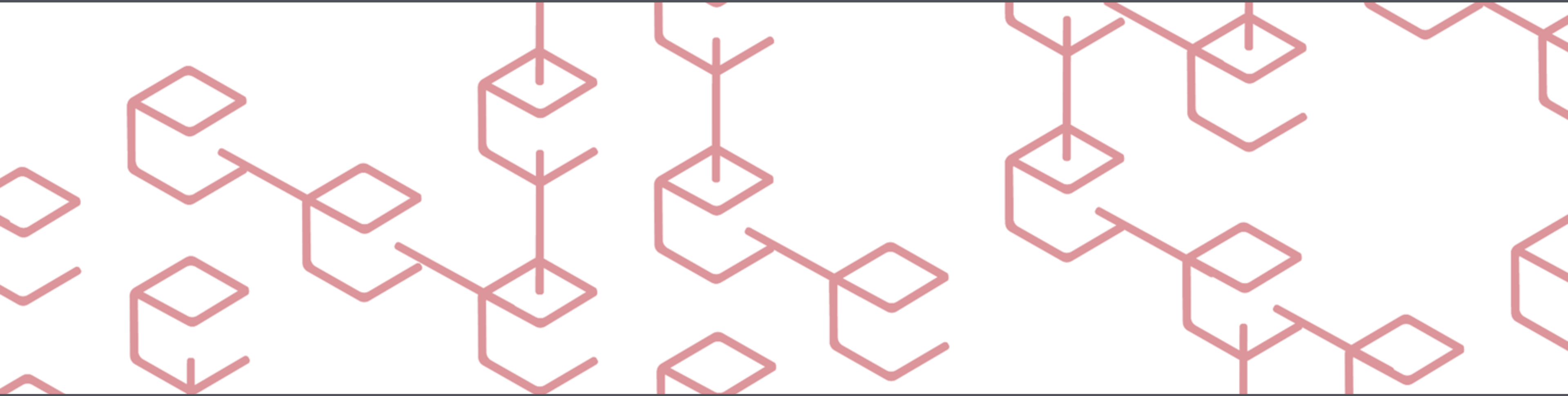
Proof-of- Stake vs. Work


PROOF OF STAKE
Decentralized
Users remain in control of their tokens
Hardware tools no necessary
Energy low consumption Sustainable concept
PROOF OF WORK
Centralized
Users organize in mining pools
Hardware tools ASICS and CPUs
Energy high consumption Unsustainable concept

Recap

- Topics Covered
    - Consensus Algorithms
        - Proof-of-Work
        - Proof-of-Stake
        - Alternatives

Smart Contracts



Smart Contracts, smart vulnerabilities and threats in the past

University of Reading

What are Smart Contracts?

- A **self-executing contract** that exists in code and is published to a blockchain.
  - An **agreement** between two parties, similar to their offline counterparts, they entail a set of services or requirements.
- When a criterion in the code is met, an action is triggered which pays-out the contract.
  - Smart Contracts are typically reactive agents that wait for an external instruction to trigger.
- Essentially by participating in a smart contract, you are ceding control over the aspect of performance as well as pay-out to a digital process.

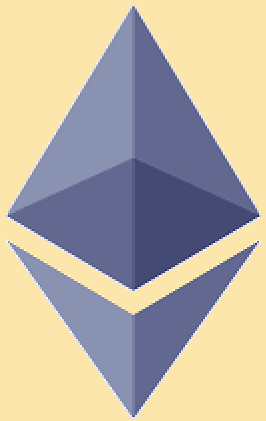
Comparison to Traditional Code and Transactions

- Unlike traditional code in a system or database, must be independently executed by **every** node
  - Due to decentralised nature of Blockchain
    - No other node in the network can be trusted
- Nodes keep their own state database and execute transactions and smart contracts themselves

- Like transactions, smart contracts pay fee
  - "*Gas*" in Ethereum
  - Value variable, based on demand
  - Transactions with more gas are likely to appended first.
    - Smart contracts said to be "*fuelled by gas*"

Brief History - Ethereum

- Smart contracts were proposed in 1994 before Bitcoin (2008) and other Blockchains however was not seen until Ethereum (2015)
  - Smart Contract can be written in a language that can be compiled into Ethereum Virtual Machine (EVM) bytecode.
  - For example:
    - Solidity
    - Vyper



Ethereum and Solidity

- Typically, Smart Contracts on Ethereum are written in Solidity, a high-level object-orientated language
  - Turing-complete,
    - Providing flexibility
    - Enables the development of contracts for voting, crowdfunding, auctions and multi-signature wallets etc.

  - Solidity targets EVM
    - Acts as a level of abstraction between the executing code and machine

  - Despite offering developers a lot of flexibility, it also empowers hackers

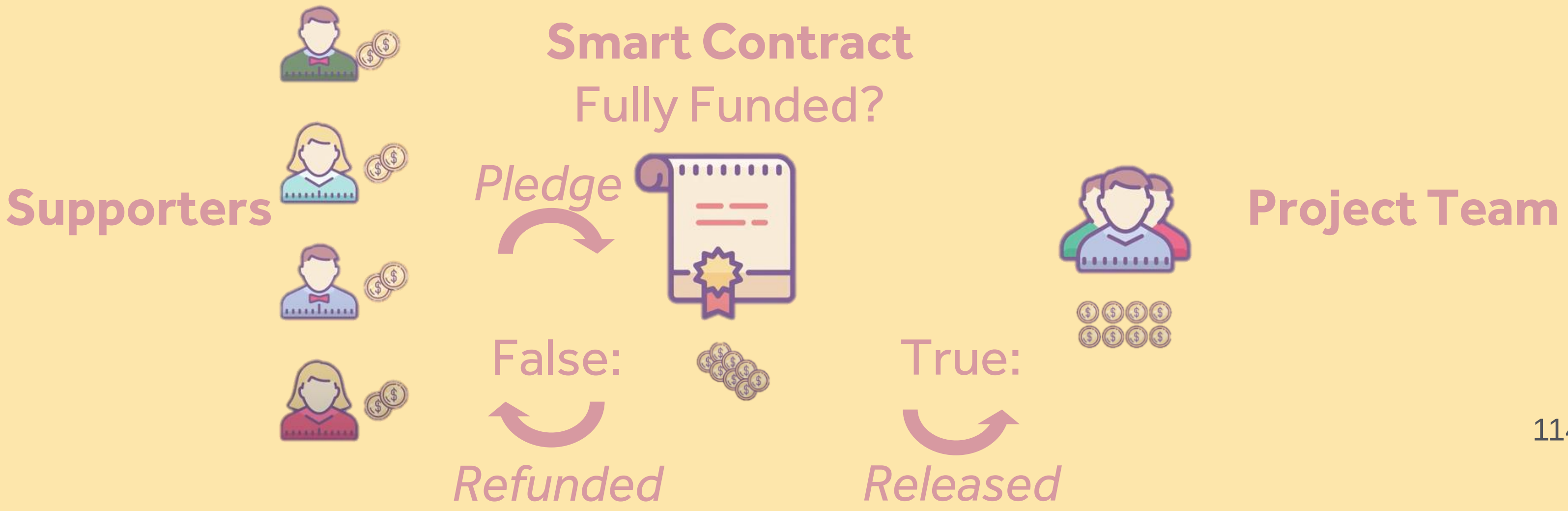
Ethereum Technology Stack

- Decouples the Smart Contract Layer from the Consensus Layer

Relations	**Smart Contracts Application Layer**
Assets	Record of Transactions Blockchain Layer
Governance	Consensus Rules Blockchain Layer
Network	P2P Network Blockchain Layer
Infrastructure	TCP/IP Internet Layer

Smart Contract – An Example

- Direct crowdfunding
  - Supporters of a project pledge funding
  - A smart contract holds the money raised from supporters and:
    - Should the fundraising goal be reached it could release the funds to the project team
    - Else the money is refunded back to the supporters



Further Potential Applications

- **Banking**
  - Issuing Loans
  - Automating payments

- **Insurance**
  - Processing Claims

- **Postal**
  - Payment on delivery

Smart Contacts - Limitations

- There exists deficits and shortfalls in the regulation of emerging technologies
  - This leaves companies and developers unsure of how to best approach the design of their service

University of Reading

Smart Contracts – Legal Concerns

- A smart contract is **not** legally binding, courts are yet to consider whether these contracts are **valid** and **enforceable** under contract law.

- Smart Contracts are **immutable**, so once the contract has been activated it cannot be changed.

Ethical and Provenance-based Concerns

- Many ethical and provenance-based concerns:
  - When entering in a smart contract with another party it is possible the **identity** of the other party is **unknown**.
  - Not only is this a **provenance** concern but also makes it difficult to judge the other parties' capacity to enter the contract
  - In typical contract factors such as duress, influence and force need to be considered before both parties can enter a legally binding contract.
    - In the case of a smart contract, the digital code within is not capable of judging these aspects.
      - >Smart contracts could be considered legally void

The Future

- Ultimately as smart contracts become more widespread, the law needs to adapt to cover them
- Additionally, transactions on the blockchain that represent transfer of ownership of real entities could also be under scrutiny.
  - Until legal groundwork is in place acknowledging cryptocurrency as a financial asset and acknowledging blockchain has non-repudiation and therefore transactions are a receipt, the law may not recognise transactions as genuine.

Technical Risk Analysis

- **Immutability**
  - Vulnerabilities and bugs cannot be patched once deployed

- Potential for hackers to execute arbitrary code within a smart contract presents a number of vulnerabilities
  - Ethereum Smart Contracts typically handle ether (the cryptocurrency of Ethereum), which can be sent from within the Smart Contract to external addresses.

University of Reading

Smart Contract – Potential Attacks

- Re-Entrancy
- Arithmetic Over- and Under-flow
- Unexpected Ether
- DELEGATECALL
- Default Visibility
- Entropy
- Uncheck call return values
- Race Conditions/Front Running
- Constructor Vulnerabilities
- Floating Points in Solidity
- Tx.origin Authentication

Smart Contract Threat Analysis

smart contract developed as lines of immutable Solidity (TC) code on a Blockchain.

Solidity targets Ethereum Virtual Machine (EVM) abstraction Ecode and EM

Opcodes; 1-byte long instructions -256 (16^2) opcodes

In Ethereum this fee is referred to as gas,

Ether limit gas limits

Smart Contract – Potential Attacks

- **Re-Entrancy**
  - Pre-requisite: Ethereum contracts have the capacity to call and use code from other smart contracts.
  - An attacker can hijack these external calls and force a smart contract to execute further code, achieved through a fallback function.
  - An attacker can force a smart contract to execute calls back to itself, essentially allowing the execution to “re-enter” the contract.
  - In practice this attack is applied to a vulnerable smart contract with code that transfers ether to the caller before logic checks are made that ensures the caller is behaving as expected.

Re-Entrancy

Ethereum SC can call and use code from other smart contracts.

Attacker Hijacks these external calls and force SC to execute further code, using fallback

to execute calls back into itself to execute 're-enter' the contract

code that transfers ether to the caller before logic checks

Code allows a user to deposit ether and also withdraw ether only once per week.

withdraw function. This fallback function would trigger before the rest of the logic in the withdraw function has been run

'DAO hack'over $60 million of DAO being drained from SC to DAO clone.

Prevent: gas-starvation: use of the 'transfer()' 2300 gas function which prevents the destination contract to call another contract

A mutex can be added, which is a state variable that locks the contract during execution.

Smart Contract – Potential Attacks

- **Arithmetic Over- and Under-flows**
  - Premise is based on the fact that integers in EVM are fixed-size data types.
    - Fixed size integers wrap around when exceeding the minimum or maximum limit
  - Using this knowledge an attacker could bypass a balance check function
  - Mathematical Libraries now exist that prevent this

Arithmetic Overflows and Underflows

Integers in EVM: fixed size data types, can only range from a certain set of numbers,

e.g. an 8-bit integer can range from 0 to 255

Fixed size integers wrap around when exceedingly the min/max limit,

255 +1 = 0 overflow

0-1 = 255 underflow

Attacker bypassing a balance check function that *require(balances[msg.sender] – amount >= 0*

Amount:8-bit unsigned integer

Replace with mathematical functions designed to be over/under-flow proof

Smart Contract – Potential Attacks

- **Unexpected Ether**
  - Takes advantage of contracts not expecting to receive ether, resulting in unintended effects.

Unexpected Ether

Attacker exploiting SC not expecting to receive any ether → Un-intended effects

Developer used invariant variables that should not change, as check variables

The variable *this.balance* returns the SC balance – not checked if SC not payable

Attacker sends ether to SC by

a *selfdestruct()* destroying the host SC and then sends all its ether to the address specified in the *selfdestruct(address)* parameter. Selfdestruct does not call any functions belonging to the target contract and as a result it can forcibly send ether.

Address are calculated using keccak256 [6] hash – not true random

Attackers can predict the address a prioir and therefore send ether.

Be aware of variables that appear to be invariant, but in fact are not.

Smart Contract – Potential Attacks

- **DELGATECALL**
  - In the EVM there are two opcodes used to modularise code; CALL and DELGATECALL
    - CALL opcode is run in the context of an external contract while DELEGATECALL is run in the context of the calling contract.
    - Due to the state-preserving property of DELEGATECALL a simple mistake in code can lead to an attacker hijacking an entire contract.
    - Issues with this often arise when used in conjunction with libraries.
      - >Libraries can change ownership and functionality.

Delegatecall

EVM opcodes: *CALL* (external SC)and *DELEGATECALL (calling SC) -state-preserving*

Misuse of DELEGATECALL when used with libraries.

Mistake in code --> Attacker hijacking SC

Second Parity Multisig Wallet hack self-destructed library, all Parity Multisig wallets frozen

Solidity provides the library keyword for development of libraries.

Smart Contract – Potential Attacks

- **Uncheck CALL Return Values**
  - Sending ether in Solidity is normally performed through methods such as transfer() or send().
  - It is also possible to use the CALL opcode to send ether.
  - In the case the transfer() function fails the code will revert, however the call() and send() functions have different behaviour and instead return true or false based off their result.
  - If a developer does not expect this behaviour, then their code may be vulnerable.

Default Visibility

DELEGATECALL attacks made stateless – library cannot self-destruct.

Solidarity Visibility specifiers are *public, internal, external and private*.

Default visibility = *public* which allows users to call them externally → Attack

internal functions can end up *public,* sensitive functions such as fund transfers being called by public users and funds being stolen.

Parity MultiSig Wallet hack: $31 million of ether from wallets

initWallet() and *initMultiowned()* public

Beware of visibility specification of variables – not public if unsafe so to do.

Smart Contract – Potential Attacks

- **Entropy**
  - Every node in a Blockchain network must be able to run a smart contract and all achieve an identical result and run in a calculable way with no uncertainty. As a result, there can be no entropy in the EVM and functions such as rand() do not exist in solidity.
  - Randomness is a highly desired function due to the demand for games of chance in smart contracts.
  - As a result some smart contracts attempt to simulate randomness through future block variables, such as hashes, timestamps, block number and gas limit.
    - These features are not truly random; miners that are responsible for adding blocks to the Blockchain have some influence over these features. If a miner wishes to influence a random function, they could withhold adding blocks to the Blockchain that have undesirable features.

Entropy

Every node to run an SC and all to achieve identical result with no uncertainty.

No entropy in EVM, No rand() in solidarity.

Randomness in demand for games of chance in SCs on EVM.

Some SC simulating randomness through future block variables,

such as hashes, timestamps, block number and gas limit.

miners can influence a random function say by withhold adding blocks to the Blockchain

43 SCs with pseudo-random number generation that can be exploited.

Prevent: Use peer-defined randomising methods can be used.

Smart Contract – Potential Attacks

- **Default Visibility**
  - Similarly, to other object orientated languages Solidity has visibility specifiers for functions.
  - This determines whether a function can be called externally by other contracts or users, only internally or only externally.
  - public, internal, external and private.
    - By default, functions have a visibility of public which allows users to call them externally, which can lead to unintended consequences if not realised.
  - Functions which are intended to be internal can end up public by mistake of developers, resulting in sensitive functions such as fund transfers being called by public users and funds being stolen.

Uncheck CALL Return Values

Sending ether in Solidity done with *transfer()* or *send()*. CALL opcode

If the *transfer()* function fails the code will revert

If the *Call()* and *send()* functions fail returns true or false , unexpected → vulnerability

Use *transfer()* instead of *send()* or *call()*

If *send()* must be used, check ok? return values immed. after *send()*

Smart Contract – Potential Attacks

- **Race Conditions/Front Running**
  - Race conditions can cause unexpected states and code behaviour when they occur. As smart contracts have the capacity to make external calls to other contracts there are many avenues in which race conditions can occur.
  - Front running is form of race condition where a miner or observer takes advantage of the public state of pending transactions to discover pending time sensitive information and undercut it.
    - Schemes such as commit-reveal exist that can be used to hide sensitive information before it is published on the Blockchain.

Front-Running

A mafia of miners exclude the pending contract from the block

that they are mining, to have the stolen one appended first

schemes such as commit-reveal exist that can be used to hide sensitive information before it is published on the Blockchain.

Prevent: Commit-REVEAL Hash transaction contents, Commit to be appended to the BC and follow up with a 2nd transaction published to Reveals the contents of the first,

Race Conditions/ Front Running

External calls to other contracts so race conditions can occur.

Attacker exploits public state of pending transactions

to discover pending time sensitive information and undercut it.

A pending SC issuing a bid on e-bay

Pending transactions and contracts with the highest gas fees published to the Blockchain first.

Attacker submits undercutting bid with a higher gas fee and claiming the goods instead

Smart Contract – Potential Attacks

- **Constructor Vulnerabilities**
  - In object-orientated languages constructors are special subroutines used to create an object. In Solidity constructors are used to perform the critical tasks that occur initialising contracts. Constructors in Solidity must have the same name as the contract to behave like a constructor, otherwise they will behave like a normal function. If the names do not match and visibility is not specified, then any user can call the constructor code and potentially take ownership of the contract.
  - Solidity introduced the constructor keyword to help mitigate this mistake.

Constructor Vulnerabilities

Solidity Constructors used to initialise SC with same name as SC else behaving as normal function

If so and visibility unspecified, then attacker can call the constructor and potentially Hijack SC

Solidity *Constructor* keyword to help mitigate this mistake.

Smart Contract – Potential Attacks

- **Floating Points in Solidity**
  - Solidity lacks native support for floating point numbers, as a result floating points numbers must be represented with integer representations.
  - Developers are required to create their own methods to execute floating point calculations, which if done incorrectly can lead to vulnerable code.

Floating Points in Solidity

Solidity – not supporting floating point numbers so these are treated as integers

DIY methods to execute floating point calculations, can lead to vulnerable code.

Libraries such as ABDK Math Quad can be used

Must keep the right precision in smart contracts when dealing with ratios and decimals.

Smart Contract – Potential Attacks

- **Tx.origin Authentication**
  - Tx.origin is a global variable in Solidity.
    - It returns the address of the account that sent the call to the contract.
  - Using tx.origin as means for authentication is a pitfall and leaves the contract open for phishing attacks.

Tx.origin Authentication

Tx.origin Solidity global variable returns the address of the caller account

If using *tx.origin* for authentication -> phishing attacks.

If user can withdraw all the ether in the contract,

that uses *tx.origin* as its authentication method.

The attacker tricks the owner of the SC to send some ether to the phishing contract.

If so then the phishing contract triggers the fallback function to withdraw from vulnerable contract and it would pass the authentication as the *tx.origin* would belong to the owner.

Tx.origin should not be used for authentication or authorisation in smart contracts for this reason. However, if it must, then logic such as *require(tx.origin == msg.sender)* prevents intermediate contracts from being used to call the

Smart Contract – Potential Attacks

- **General Preventative Measures**
  - Vulnerabilities in Solidity and the EVM means developers must be diligent and stay aware when developing smart contracts
  - Preventative measures exist that help detect and stop these vulnerabilities before the smart contract is deployed

Upgrading Smart Contracts

- Smart contracts are immutable
    - This does not mean that they cannot be upgraded
    - It is possible to upgrade a smart contract given that a module to make it upgradable has been included in the original smart contract code
        - Bugs or vulnerabilities can be patched out

Vyper

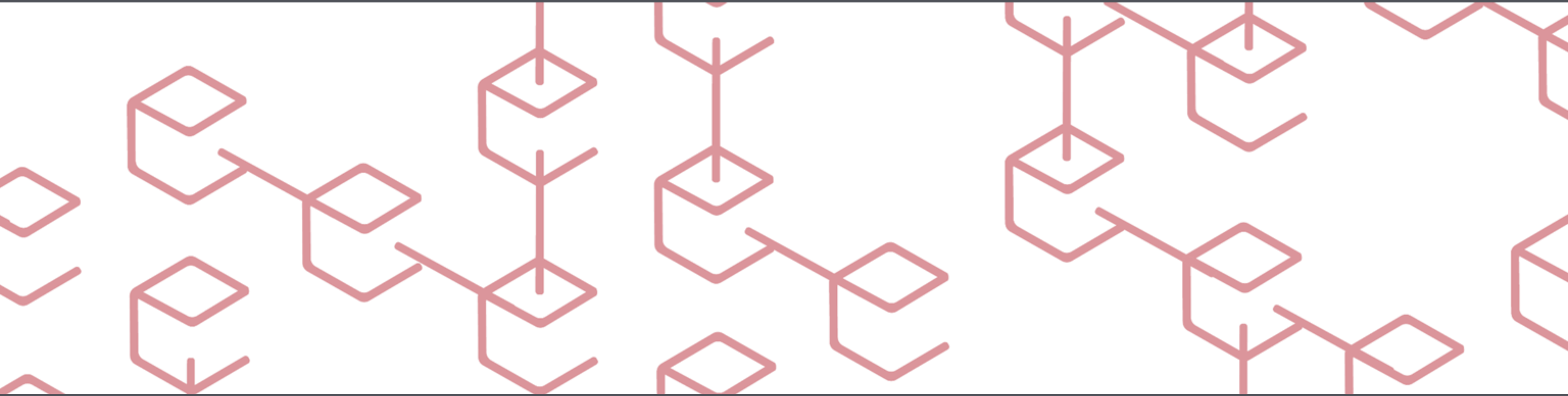
- Vyper is another programming language that compiles into EVM bytecode. It resolves the majority of issues with Solidity
  - It is designed to boost security and is not Turing-complete.
  - Goals
    - Language and design simplicity,
    - Security, and
    - auditability.
  - Vyper focuses on:
    - Making the compiler simple,
    - Language easy to read and understand, and
    - Preventing any loopholes or vulnerabilities occurring in smart contracts.

Recap

- Topics Covered
  - Smart Contracts
    - Example
    - Use-cases
    - Solidity
      - > Basics
      - > Vulnerabilities
      - > Upgrading
    - Vyper

University of Reading

Blockchain Issues & Vulnerabilities



Concepts concerning integrity and resolution of Blockchains

Attacks on Blockchain

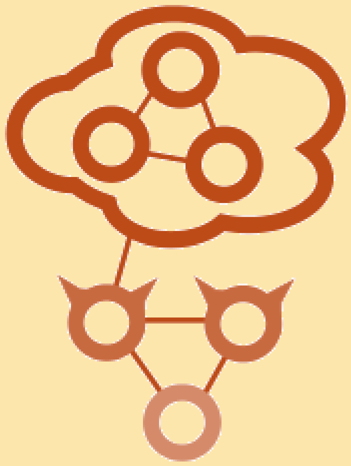
- Categories of attacks on blockchains include:
  - Peer-to-Peer network-
  - Consensus and Ledger-
  - Smart Contract-
  - Wallet-

  … -based attacks

Peer-to-Peer Network-based Attacks

- **Eclipse attack**
  - A node will depend on x nodes selected using a peer selection strategy to have its view of the distributed ledger.

If an attacker can manage to make the node *choose* all the x nodes from their malicious nodes alone, then they can eclipse the original ledger's view and present their own manipulated ledger to the node.



Peer-to-Peer Network-based Attacks

- **Sybil attack**
  - While the eclipse attack is about eclipsing a user’s view of the true ledger, the Sybil attack targets the whole network
  - In a Sybil attack, an attacker will flood the network with a large number of nodes with pseudonymous identities and try to influence the network
    - These nodes, though appearing like unrelated individuals, are operated by a single operator
  - In this case, the objective is not to target one user, but a number of nodes or the network as a whole, and generate a fork in the ledger if possible, allowing the attacker to make double spending amongst other attacks

Consensus Mechanism and Mining-based attacks

- **Selfish mining attack**
  - Many blockchains consider the longest chain to be the true latest version of the ledger.
    - A **selfish** miner can try to keep building blocks in stealth mode on top of the existing chain, and when they build a lead of greater than two or more blocks than the current chain in the network, they can publish their private fork, which will be accepted as a new truth as it is the longest chain.
    - They can do transactions in the public network just before publishing their longer stealth chain to reverse the transaction just applied.
    - This effectively provides a small window for the attacker to do double spending based on this ability to build a stealth chain by building sufficient block lead. (Finney attack).

Consensus Mechanism and Mining-based attacks

- **Mining malware**
  - Malware uses the computing power of unsuspecting victims’ computers to mine cryptocurrencies for hackers.

Consensus Mechanism and Mining-based attacks

- **51% attacks**
  - This attack is possible when a miner or a group of miners controls 51% or more of the mining power of the blockchain network.
  - Though it is very difficult to occur for large networks, the possibility of a 51% attack is higher in smaller networks.
  - Once a group has majority control over transactions on a blockchain network, it can prevent specific transactions or even reverse older transactions.

Consensus Mechanism and Mining-based attacks

- **Timejack attack**
  - Nodes in certain blockchain networks like Bitcoin depend on internal timing derived from median time reported by its peer nodes.
  - For example, you depend on your friends to know the time.
  - An attacker manages to put a lot of malicious people in your friends' list, then they can manipulate your time.
  - The first step to this attack can be an Eclipse attack on the target node. Once this attack is complete on a target node, then the node will not accept blocks from the actual network as the timestamp of the blocks will not be inline with its timestamp.
  - This provides an opportunity for the attacker to be double spending or do transactions with the targeted node as these transactions can't be submitted to the actual network

Consensus Mechanism and Mining-based attacks

- **Finney attack**
  - If you can mine a block with one of your transactions in it and keep it in stealth, there is an opportunity for you to double spend the money.
  - If a merchant accepts the unconfirmed transaction, you can transfer them this earlier transacted currency.
  - Next you publish the earlier mined block, which was kept in stealth, before your new transaction is confirmed on the network.

Consensus Mechanism and Mining-based attacks

- **Race attack** (A minor variant of the Finney attack).
  - Main difference is that the attacker need not pre-mine the block with their transaction, which they intend to double spend.
  - During the attack, the attacker submits an unconfirmed transaction to a merchant (the victim) and simultaneously does another transaction which they broadcast to the network.
  - It is easier for the attacker to launch the attack if they are directly connected to the merchant’s node. This would give the merchant an illusion that their transaction is the first, but that is never submitted to the blockchain network by the attacker.

Attacks Summary

- Mistakes in blockchain deployment can be very costly, especially in permission-less networks, as anyone can participate, their identity is anonymous and reverting back mistakes is impossible.
- The majority of attack vectors and vulnerabilities have solutions when found before deployment but considering the immutable nature of blockchain and hard- or soft-fork not being a practical option, through understanding of concepts, security audits and detailed testing are very important before deployment.

Scalability of Blockchain

- Within blockchains scalability can refer to:
  - Throughput
  - Latency
  - Bootstrap time
  - Cost per transaction

- Blockchains are notably difficult to scale across all of these.
- As a result there must be a trade-off between throughput and public verifiability.

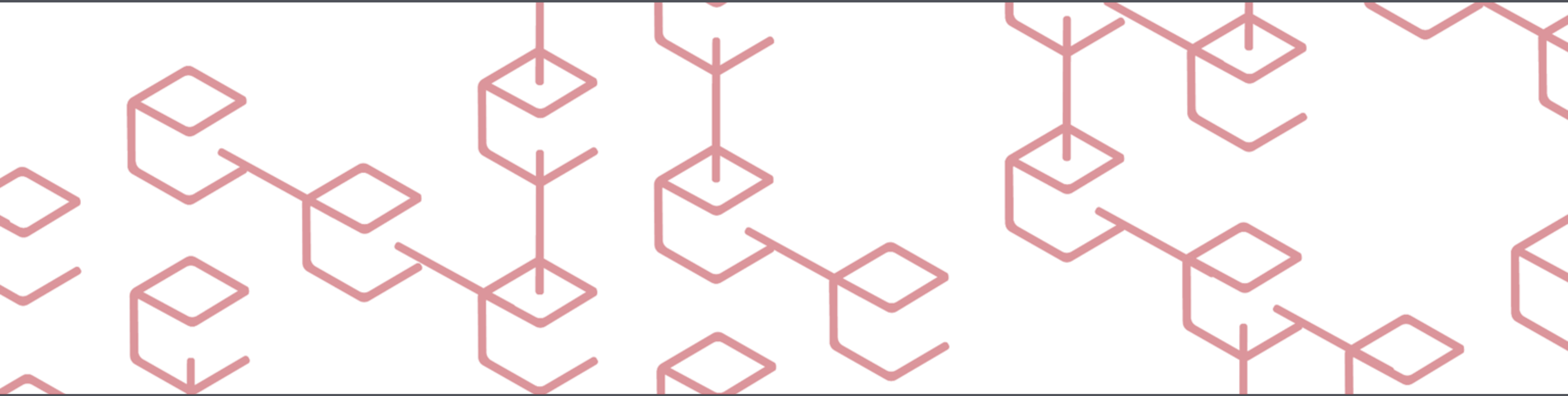
Scalability of Blockchain

- Solutions for scalable blockchains can be categorised into *four* types:
  - **Scaling Bitcoin**
    - Solutions to improving the throughput of Bitcoin via enlarging its block size or reducing block interval without changing Bitcoin's PoW consensus algorithm.
  - **Scaling PoW**
    - Solutions that still work in the Nakamoto consensus framework, but achieve a higher throughput than the Bitcoin's PoW algorithm through modifying the algorithm.
  - **Scaling Byzantine Fault Tolerance (BFT) Algorithms**
    - Solutions based on BFT algorithms, but having a reduced message complexity compared to PBFT.
  - **Scale-out Blockchains**
    - Solutions that relax the requirement of having mining nodes to know the whole transaction history, so that the throughput of the system could grow as the network size grows and therefore, achieve better scalability.

Recap

- Topics Covered
  - Blockchain Issues
    - Duplication of Work
    - Scalability
  - Vulnerabilities
    - P2P
    - Consensus Mechanism
    - Mining

Byzantine Fault Tolerance



Fault tolerance amongst the peer-to-peer network of nodes

Terminology

- **Byzantine Fault**
  - "A **condition** where components **may** fail and there is imperfect information on whether a component has failed"
- **Byzantine Generals Problem**
  - "A **situation** in which, in order to avoid catastrophic failure of the system, its actors must agree on a concerted strategy, but some are unreliable."
- **Byzantine Fault Tolerance**
  - "The dependability of a fault-tolerance system to such conditions."

Byzantine Faults

- Any **fault** presenting different **symptoms** to different observers

- Formal Definition:
  - Given a system of components, some of which are dishonest
  - A component A tries to broadcast a value: x
  - Other components discuss with each other and verify the consistency of A's broadcast
    - Settle on a common value: y
      > *y may or may not be equal to x*

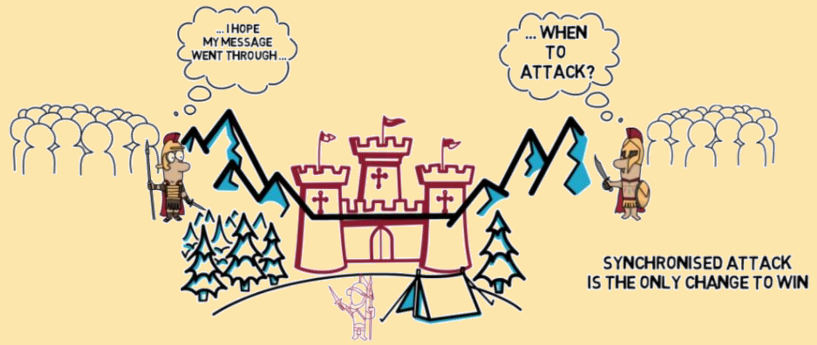
  - System is said to resist Byzantine Fault if A can broadcast a value x, and then:
    1. If A is honest, then all honest components agree on the value x
    2. In any case, all honest components agree on the same value y

Byzantine Fault Tolerance (BFT)

- **Objective**
  - Defend against system component failure which prevent others from reaching an agreement when required for correct operation
- Operationally correct components of a BFT system must be able to continue service as intended, assuming sufficient accurately-operating components remain

Two Generals Problem

- An example of a **consensus problem**
  - A consensus problem is where two nodes in a distributed system simply have to agree
- **Background**
  - In this problem there are two armies: army A and army B, each led by a general
    - They need to agree on one fact, which is:
      - >Attack army C or retreat


... I HOPE MY MESSAGE WENT THROUGH...
... WHEN TO ATTACK?
SYNCHRONISED ATTACK IS THE ONLY CHANGE TO WIN

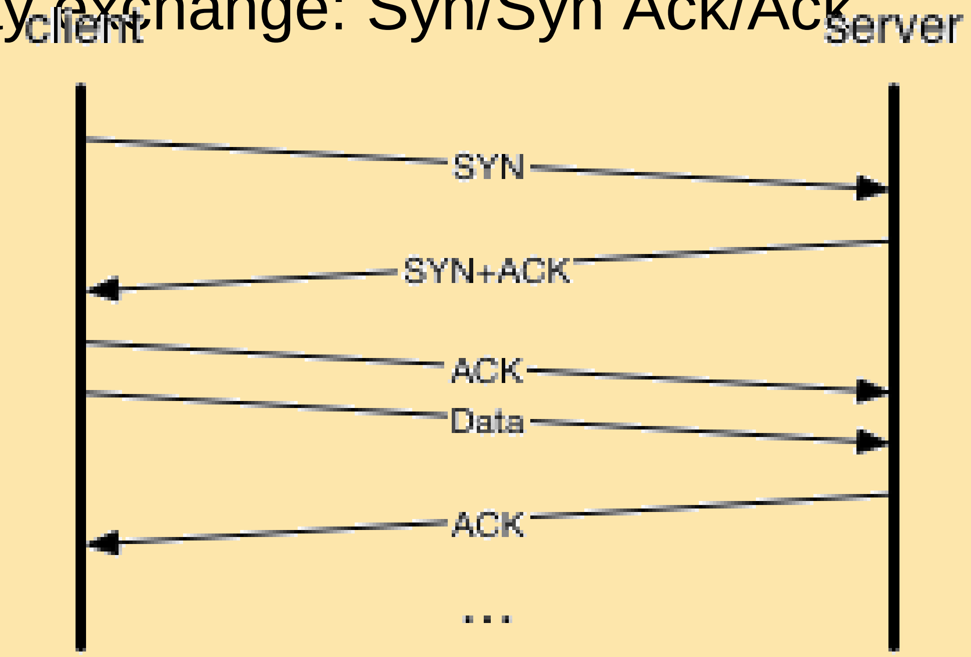
Two Generals Problem

- If both armies A and B attack they will win
- If neither army attacks they will survive to fight another day
- If only one attacks they will lose
  - Army C is bigger than each of the two individual armies
- **Challenge**
  - Generals can only communicate through couriers
    - Couriers ride through the territory of army C
      - > May or may not reach opposing general
- **Requirement**
  - Protocol for messages such that Generals are in consensus
    - They can agree to attack or not
    - Using minimum number of messages possible

Two Generals Problem

- General A decides to attack and informs General B

- Issue made clear through a TCP/IP-based approach
    - Three-way exchange: Syn/Syn Ack/Ack



Two Generals Problem

- **Protocol**
  - General A decides to **attack**
  - Sends message to General B:
    "*If you respond I'll attack*"
    - If this message gets lost there is no problem
      - > A not going to attack unless response received
  - B receives message and responds:
    "*Okay, I now know that A is going to attack if I respond. I'm going to respond*"
  - B also wants to attack so they send a message saying:
    "*If you respond I'll attack*".
    - If that message is lost, no problem once again
      - > Neither are committed

Two Generals Problem

- General A receives the second message
  - Their challenge got response so **commits** to attack

- General A responds with:
  "*Okay, We are going to attack. B you should attack too*."

- **Problem**
  - If this message is lost, general A is committed to attack but B is not yet committed

- Highlights challenges with consensus in Distributed Networks

Two Generals Problem

- **Searching for Solutions**
  - No matter how many messages are add to this protocol, there will always be this problem
    - There will always be an inconsistent state until the last message is received

- There is no way of solving the two generals problem for definite
  - Instead protocols offer levels of Fault Tolerance
  - Workarounds based on assumptions

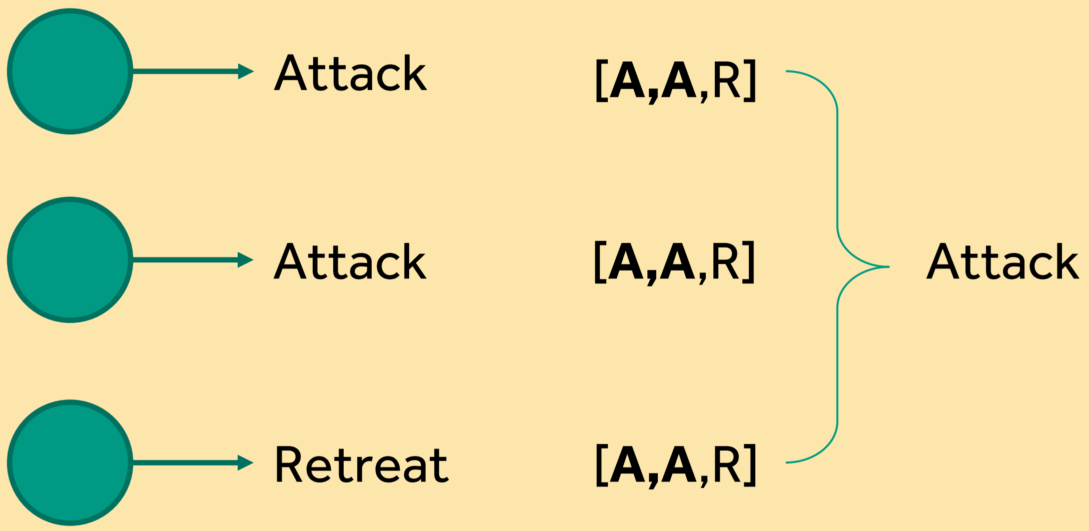
Byzantine Generals Problem

- **Two Generals Problem**
  - Demonstrates that if a Byzantine Failure results in the failure of an entire communication network there is no way to achieve consensus

- **Byzantine Generals Problem**
  - If nodes are corrupted opposed to communications
  - How many Byzantine node failures can a distributed system survive and still generate the correct answer?

Byzantine Generals Problem

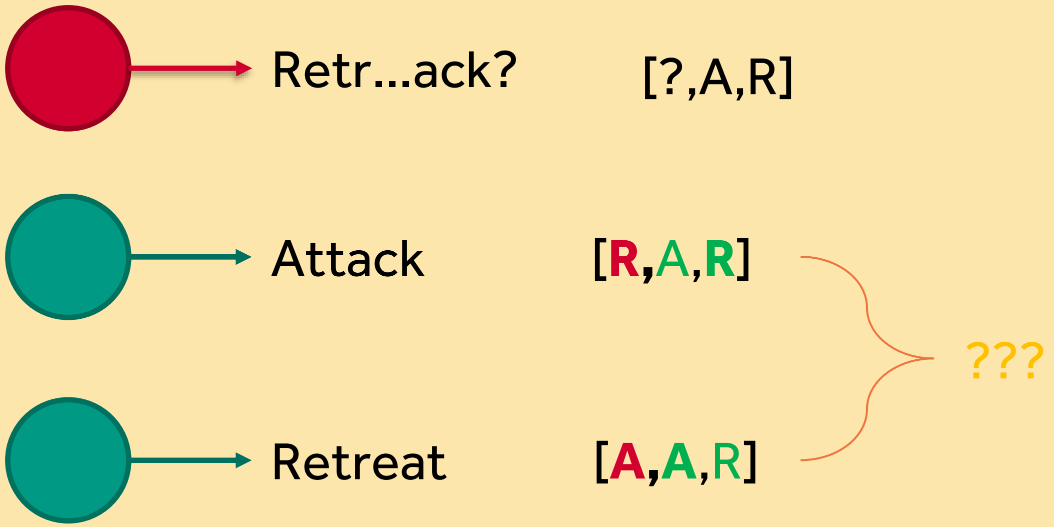
- A number of Generals are planning to attack a fort
  - Once again, need to decide whether to **attack** or **retreat**
  - Each decide on their approach and cast a vote accordingly



  - Everyone knows all votes
  - Majority wins therefore: Attack

Byzantine Generals Problem

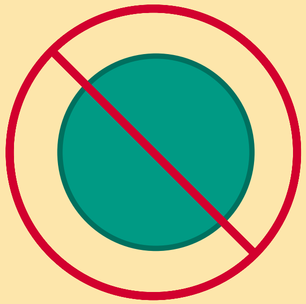
- If one general is a traitor:
  - Goal - Corrupt the consensus

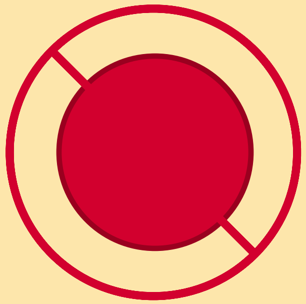


- BGP is the problem of communicating one decision from a general to all other generals
  - Goal: Have all loyal generals agree on a single fact (?)

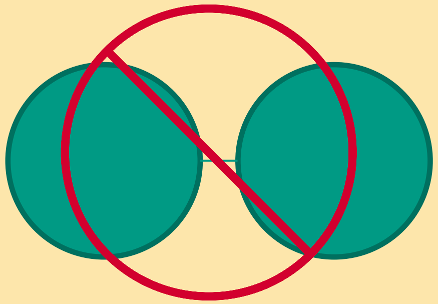
Byzantine Generals Problem

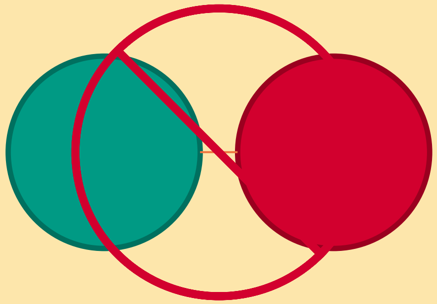
- Special cases: N=Number of Generals
    - N=1 or N=2





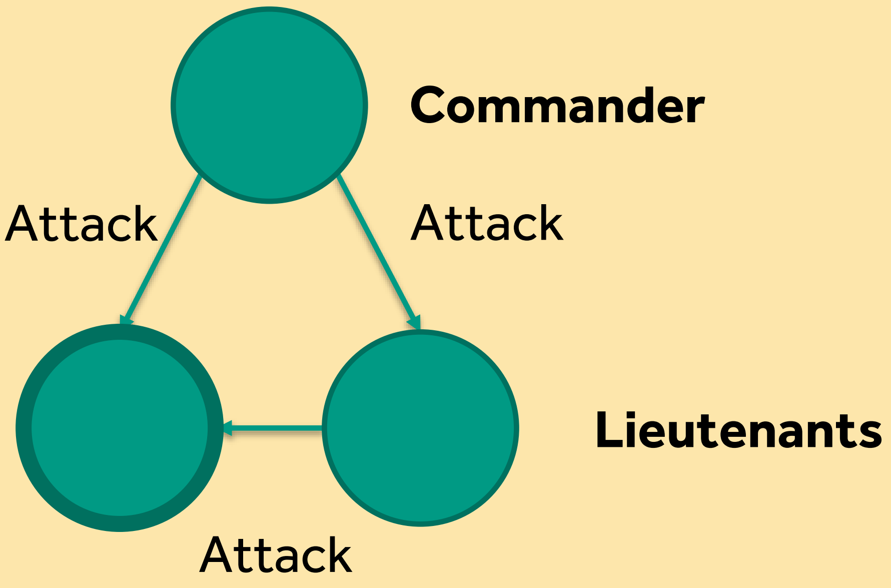
No Consensus to be had
in all of these cases

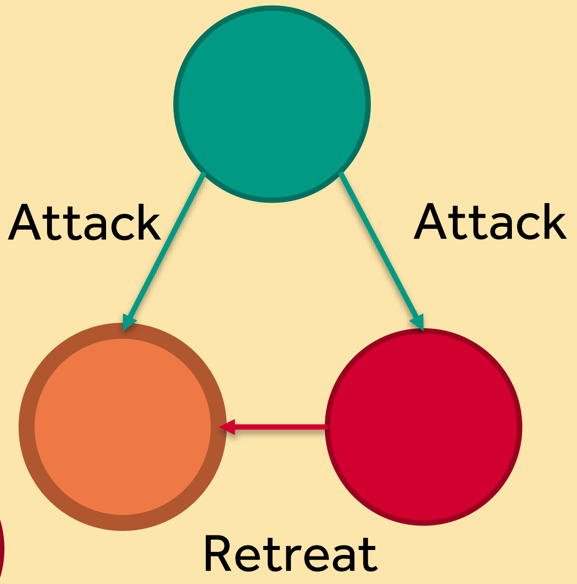


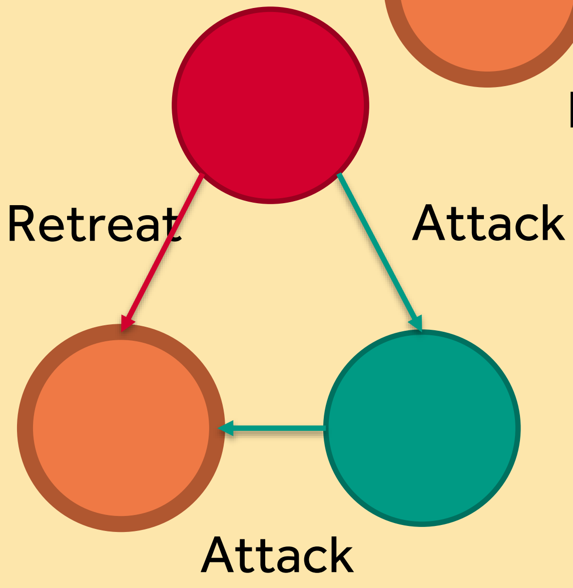


Byzantine Generals Problem

- N=3
  - Commander: general issuing order
  - Lieutenants: Listening for orders







- If one is a traitor there is no solution

Byzantine Generals Problem

- If $\frac{1}{3}$ of generals are traitors there is no solution
- For a solution at least 2/3 mjst be honest

Byzantine Fault Tolerance

- Byzantine Fault Tolerance (BFT) is defined as the failure tolerance capability of a system against the **Byzantine Generals' Problem** (BGP).
  - Consider an agreement scenario among a set of players:
    - Each player holds a possibly different initial value, and all players need to agree on a single value by obeying a consensus protocol.
    - In a system where such agreement is reached if a majority of the players are honest players who rigorously follow the protocol, even when a minority of the players are malicious and may diverge from the protocol arbitrarily.
    - This system is known to be Byzantine fault tolerant.

Byzantine Fault Tolerance (2)

- Most traditional distributed computing systems have central authorities that coordinate and determine what to do next when Byzantine failures occur, however, in a decentralised blockchain system, there is no such authority.
- The blockchain is maintained as a distributed global ledger by the network such that each node has a replica of the chain.
- The initial values are the candidate block to be validated and then inserted into the blockchain.
- For each candidate block, the verification is done by having the network reaching an agreement through the digital signatures of a sufficient number of nodes.
- Only those candidate blocks that are verified by the network can be added to the blockchain.
- In order to prevent the occurrence of the Byzantine faults, the blockchain systems rely on the consensus algorithms, such as PoW and PoS, to endorse transactions.

BFT Implementations – Example

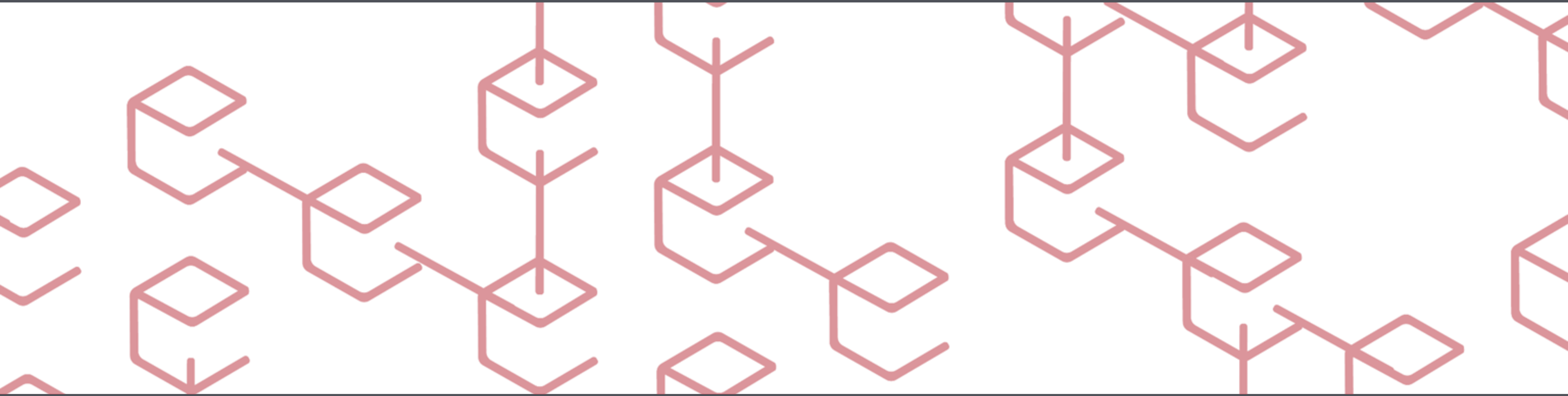
- Proof-of-Work consensus algorithm in the Bitcoin Blockchain allows the system to overcome Byzantine failures and reach a coherent global view of the system’s state

Recap

- Topics Covered:
  - Byzantine Generals Problem
  - Byzantine Faults
  - Byzantine Fault Tolerance

University of Reading

Hashing – Merkle Root Algorithm



Merkle root algorithm, fork resolution and interoperability

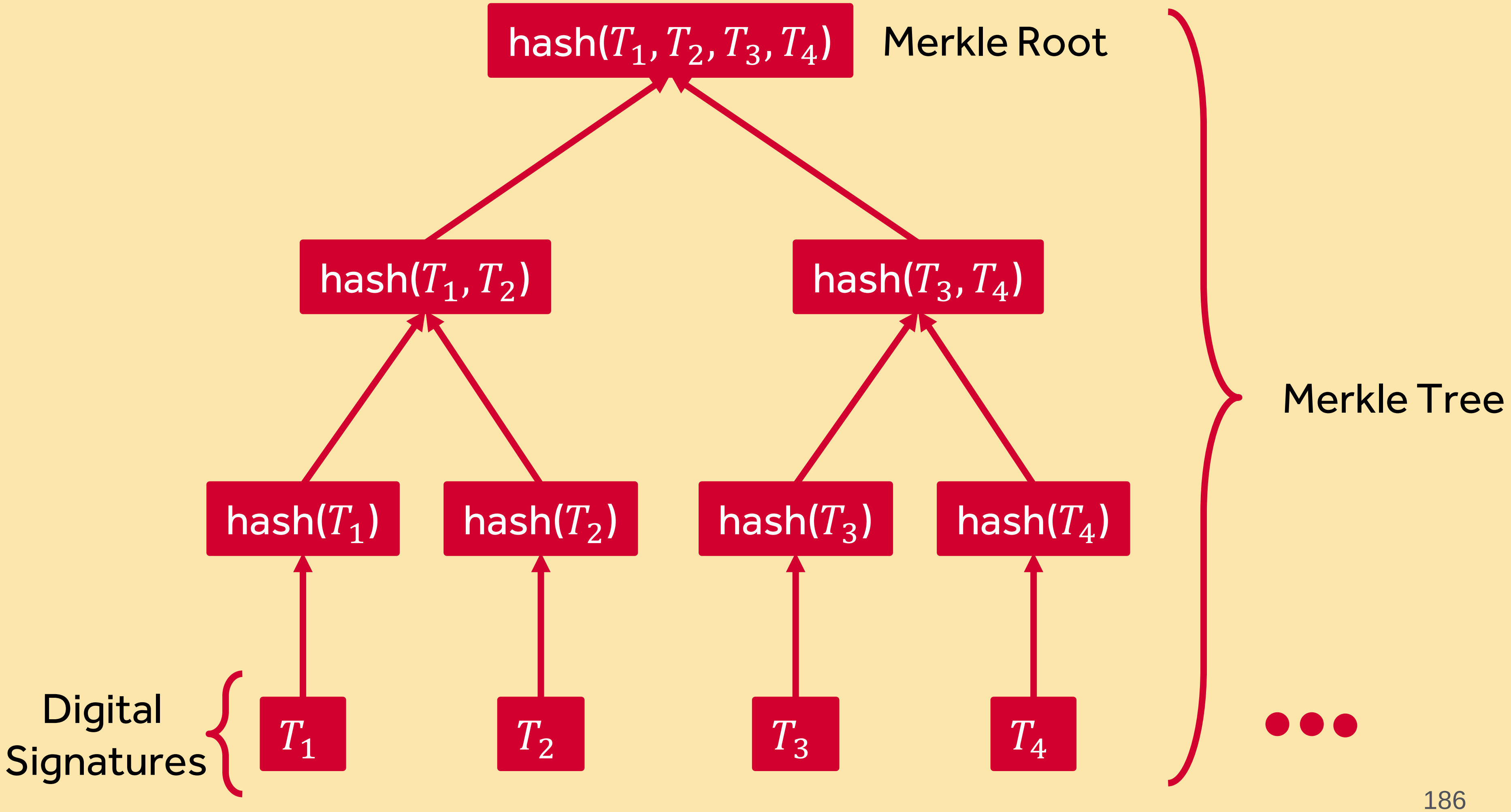
Merkle Trees - Introduction

- Vital component of Blockchain
  - Verifies the **integrity** of transactions within a block
  - Ensures data is:
    - **Complete**
    - **Unaltered**

- Blocks contains numerous transactions → Must be combined

- The Merkle Root is the result of a Merkle Tree function
  - Hashes of each transaction are hashed together to make one overall hash for a block

Merkle Trees - Introduction



Merkle Root Overview

- A block contains n transactions: $T_1, T_2, T_3, T_4 \ldots, T_n$

- The hash of T_1 and T_2 are combined to produce $H_{1,2}$
- The hash of T_3 and T_4 are combine to produce $H_{3,4}$
    - For blocks containing an odd number of transactions $T_n \rightarrow H_n$

- Next hashes $H_{1,2}$ and $H_{3,4}$ are combined to produce $H_{1,2,3,4}$
- And hashes $H_{1,2,3,4}$ and H_n are combined to produce the Merkle root: $H_{1,2,3,4,\ldots,n}$

Merkle Root Pseudocode

```
MerkleRoot(hashes) {
        List<String> tempHashes
        String tempHash

        if (hashes.size == 1)
                return CombineHash(hashes[0],hashes[0])

        while (hashes.size != 1) {
                for (I = 0, I < hashes.size, I += 2) {
                        if (I == (hashes.size - 1))
                                tempHash = CombineHash(hashes[i], hashes[i])

                        else
                                tempHash = CombineHash(hashes[i], hashes[i+1])

                        tempHashes.add(tempHash)
                }
                hashes = tempHashes
        }
        return hashes[0]
}
```

Combining Hashes

+ Concatenate and Re-Hash

$$hash(concat(hash(A), hash(B)))$$

- Commutative Operations
  - Addition: Order independent

$$hash(A) + hash(B) = hash(B) + hash(A)$$

  - XOR: Security reduction

$$hash(A) \oplus hash(B)$$

Vulnerable to "Generalised Birthday Attacks"

Merkle Roots Functionality

- In blockchain, every transaction in a block is paired with another;
  - the hashes of the transaction pairs are then hashed together to make a single hash for each pair
- This process is then repeated, each hash is paired with another and they are hashed together until only one remains called the **Merkle Root**

- Merkle Roots confirm that identical information exists across two different locations, as the roots should both be the same.

- Useful for a node receiving a block to confirm that all transactions were included and unchanged

- This allows blockchains around the world to quickly and efficiently coordinate records across nodes

Merkle Roots Properties

- Blockchain validity, immutability and irreversibility originates from the many validation and verification methods built into it
- Blocks can be validated by re-calculating the provided hash and checking the previous hash.
- The transactions in blocks can be validated by confirming the signature and checking if the sending address has the rights to the digital asset such as having enough currency for the transaction.
- Merkle Root ensures all transactions are included.
- Other checks include validation that the difficulty, reward and hash match the expected values.

University of Reading

Forks in Blockchain

"A **collectively** agreed upon software update"

- A condition where the state of the Blockchain diverges into different chains, where one chain operates by different rules than another.

Soft Forks

- Backwards **compatible** changes
    - Miners can still interact without updating

- I.e. Maintains compatibility with older versions
    - E.g. Bitcoin’s SegWit Update to new class of address

Hard Forks

- Change to protocol that renders older versions **invalid**

- This normally involves changing parameters that would be rejected by the old protocols, such as block size or difficulty algorithm.

- When disagreements in the way to run a Blockchain implementation occur hard forks can follow leading to networks running different versions.

Hard Forks

- **Diverges** into different chains operating different rules
  - Requires user to choose which chain
  - Examples:
    - Pending Casper update to Ethereum (PoW → PoS)
    - Bitcoin (1MB) and Bitcoin Cash (8MB) concerning Block-size

$$>\uparrow Block\ Size = \uparrow \frac{Transactions}{Second}$$

$$>\uparrow Rewards\ \ but\ \downarrow \frac{Reward}{Transaction}$$

- Less opportunity for replace-by-fee

Reasons for Forking

- Technical Changes
- Cultural Differences
- Differing values of Developers

Forking Benefits

- A way of allowing the community to change certain aspects of the rules while maintaining the security that comes with chains
  - *New chain is a duplicate of the old chain*

- Soft forking allows a network to remain completely intact while continually introducing new features
  - Improve performance/security
  - Solve issues

- Hard Forks create new digital assets
  - Provide “free” coins to balance issuance

Forking Issues

- In the case of a major difference Soft Forking is not always possible.
- Hard Forks:
  - Create splits in communities
  - Impact volatility

Recap

- Topics Covered
  - Merkle Roots
  - Forks
    - Soft vs. Hard

Security and Privacy

Public/Permissioned vs. Private/Permission-less blockchains

University of Reading

What is a Public Blockchain?

- A public blockchain is permission-less.
- Anyone can join the network and read, write, or participate within the blockchain.
- A public blockchain is decentralised and does not have a single entity which controls the network.
- Data on a public blockchain is secure as it is not possible to modify or alter once it has been validated on the blockchain

- Bitcoin and Ethereum are well-known examples of public blockchains

What is a Private Blockchain?

- A private blockchain is permissioned blockchain.
- Private blockchains work based on access controls which restrict the people who can participate in the network.
- There are one or more entities which control the network and this leads to reliance on third-parties to transact.
- In a private blockchain, only the entities participating in a transaction will have knowledge about it, whereas the others will not be able to access it.

- Hyperledger Fabric of Linux Foundation is an example of a private blockchain

Similarities of Public and Private Blockchains

- Both function as an append-only ledger where the records can be added but cannot be altered or deleted. Hence, these are called immutable records.
- Each network node in both have a complete replica of the ledger. Both are decentralised and distributed over a peer-to-peer network of computers.
- In both, the validity of a record is verified, thus providing a considerable level of immutability, until the majority of the participants agree that it is a valid record and reach consensus. This helps prevent tampering with the records.
- Both Blockchains rely on numerous users to authenticate edits to the distributed ledger thus helping in the creation of a new master copy which can be accessed by everyone at all times.

Differences between Public and Private Blockchains

- The order of magnitude of a public blockchain is lesser than that of a private blockchain as it is lighter and provides transactional throughput.
- Level of access granted to participants – in a public blockchain, anyone can take part by verifying and adding data to the blockchain. In private blockchains, only authorised entities can participate and control the network.
- A public blockchain is decentralised, whereas a private blockchain is more centralised.
- Consensus algorithms such as Proof of Elapsed Time, Raft, and Istanbul BFT can be used only in the case of private blockchain.

Differences between Public and Private Blockchains (2)

- Transactions per second are lower in a public blockchain when compared to private blockchains. As the number of authorised participants is fewer in a private blockchain, it can process hundreds or even thousands of transactions per second.
- A public blockchain cannot compete with a private blockchain in terms of scalability issues as it is slow and hence can process transactions only at a slow pace. In a private blockchain, as only a few nodes need to manage data transactions can be supported and processed at a much higher pace.
- Public blockchains are trust-less, and in a private set-up, participants must trust one another.

Differences between Public and Private Blockchains (3)

- In a private blockchain, the validity of records cannot be independently verified as the integrity of a private network relies on the credibility of the authorised nodes.
- A public network is more secure due to decentralisation and active participation. Due to the higher number of nodes in the network, it is nearly impossible for “bad actors” to attack the system and gain control over the consensus network. A private blockchain is more prone to hacks, risks, and data breaches. It is easy for bad actors to endanger the entire network.
- A public blockchain consumes more energy than a private blockchain as it requires a significant amount of computational power to function and achieve network consensus. Private blockchains consume a lot less energy

Differences between Public and Private Blockchains (4)

- In a public blockchain, it is necessary to grant access to a centralisation authority to oversee the entire network, thus making it a private blockchain at this point. In a private blockchain, anyone who is overseeing the network can alter or modify any transactions according to their needs.
- In a private blockchain, there is no chance of minor collision. Each validator is known and they have the suitable credentials to be part of the network. But in a public blockchain, no one knows who each validator is and this increases the risk of potential collusion.

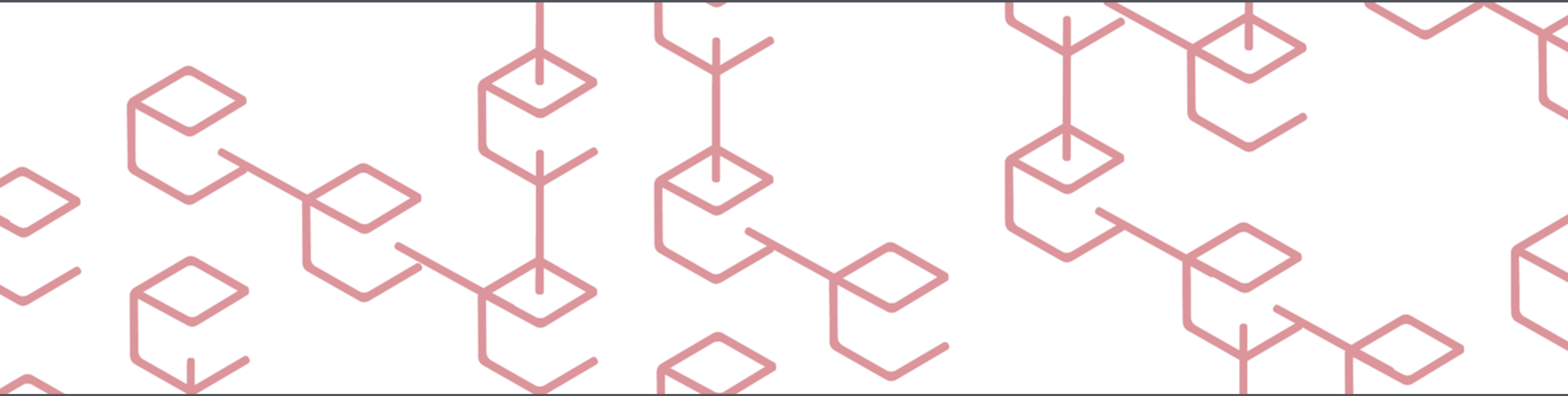
Public vs. Private Blockchains

- Blockchain technology is a highly complex and profound field which consists of numerous concepts and different forms of network.
- Which form is best depends on the use-case proposed and even a hybrid solution may be optimal.
- One such hybrid solution is Directed Acyclic Graphs (DAG) aka **Tangle**.

Recap

- Topics Covered
    - Public vs. Private Blockchains

Legal Issues of Blockchain



Ethics and Socially Responsible blockchain designs

Socio-Economic Impacts of Blockchain

- Black Markets
- White Collar Employment
- Variance in Blockchains
- Risk of recentralisation
- Unequal access
- Trusting developers
- “Looming threat of China”

Black Markets

- Given the relative pseudonymity that blockchain offers; online black markets can exist that use Blockchain currencies as tender.
- On such markets, goods and services which would normally be illegal in the users native country are traded anonymously.
- Without an explicit identity attached to each wallet it is difficult to track who uses these markets.

White Collar Employment

- Contract lawyers may find some of their work replaced by smart contracts as they are cheaper and immutable.
- It may become important for contract lawyers to learn how smart contracts work such that they can offer their services in validating them.
- Other areas in White Collar Employment may be affected too – administrators that deal with wire transfers may find that their jobs won’t exist in the future due to Blockchain technology.

Variance in Blockchains

- Not all blockchains share the same values.
- Some are much more decentralised than others, some are permissioned, some are not.
- For consumers of different blockchains; the differences of each Blockchain should be made abundantly clear, therefore the trust of the users is not violated.

Risk of Recentralisation

- Ideally Blockchain should be decentralised, such that corporations have no power or control over them.
- Instead of relying on a bank as an intermediary, you can instead trade directly with another user.
- However, cryptocurrency exchanges have become the norm where cryptocurrencies are traded, meaning users must trust these exchanges with their funds.
- There have been many cases where exchanges have been hacked and funds stolen.

Risk of Recentralisation (2)

- Miners or nodes that take part in adding new blocks and transaction to the blockchain, often group together into groups known as “mining pools”. These pools perform the Proof-of-Work for each block together, so they do not need to compete with each other.
- The rewards of the next mined bock are then split between every node in the pool.
- They still need to compete with other nodes and mining pools on the network, but as a large group they have a better chance of being the first to mine a block.
- This results in very large mining pools, as users appear to be more inclined o join the larger pools; as it results in more consistent rewards over time.
- One large flaw in this idea is the risk of 51% attacks.
- If this is the case, the blockchain is essential centralised, as the large mining pool can subsequent launch double spend attacks on the chain and succeed thus the Blockchain immutability is gone and all are at risk.

Equal Access

- Fast and reliable internet access is often required to fully utilise blockchain technology. In areas of the world where this is not the case interacting with a blockchain or running a node may be a cumbersome task.
- As a result, some Blockchain implementations may favour certain cultures and countries due to optimisation requiring high bandwidth, storage and processing capacities and electricity cost is relatively low…

Trusting Developers

- Ultimately users must trust that core developers of blockchains operate ethically and honestly.
- A blockchain implementation should be trust-less however a rogue developer could change the implementation to steal funds.
- Even if a developer does operate honestly it is required to trust that they are competent.
- Trust doesn’t just extend to developers, if you are not confident in understanding smart contract syntax you must trust the author of a smart contract.
- Unless you have an expert who can validate it for you, you cannot be confident it does what the author promises.

Looming Threat of China

- China has expressed adversarial position regarding cryptocurrency and especially Bitcoin.
- Under the current Chinese leadership, they are unlikely to want cryptocurrency to succeed as it takes financial power from the state.
- 78% of Bitcoin miners are located in China, due to the relatively low cost of power.
- In the case the government decide to seize a significant portion of these miners, they could attack the network with 51% attacks.
- Permissionless cryptocurrency developers and investors should be aware of the threat China potentially holds.

Ethical Analysis

- Time-Vital Data
- Environmental Sustainability
- Public nature of the chain
- Immutability of the chain
- Encryption Lifespan
- Lack of Regulation
- Single point of failure for users
- Volatility in value
- Trusting the Blockchain

Time-Vital data

- Traditional blockchain implementations do not priorities time-vital transactions or data while they are waiting to be added to the blockchain.
- Nodes normally have no way of discriminating one type of transaction from another.
- When nodes select transactions to add to the next block, they normally choose the transactions with the highest reward attached.
- Banking and insurance sectors will likely require time-vital data to be transferred as soon as possible.
- Without a mechanism to prioritise this important data, transactions will have to compete with each over via fees to be processed quickly.

Time-Vital data (2)

- Transactions on the blockchain are not normally considered “confirmed” or ready until x blocks have been added to the chain after the transaction.
  - In Bitcoin it is normal to expect an hour wait before a transaction is confirmed.

Environmental Sustainability

- Many existing blockchain currencies that use the PoW consensus algorithm have huge issues with power consumption.
- PoW was originally a system designed to prevent spam and double spend attacks in blockchain.
  - In Bitcoin, the total computational power is estimated to be a minimum of 22 TWh, almost the same as the electricity consumption rate of a country the size of Ireland.
  - However this remain profitable for miners. If Bitcoin prices remain high then huge amounts of energy will continue to be consumed in mining efforts.

Public Nature of the Chain

- The public nature of the blockchain can be used for good and has been use to track down distributers of illegal dark web sites via their wallet addresses. However, users may feel violated by this, which could discourage adoption in the future.
- Unless the blockchain is private or permissioned the public will be able to view the contents of the blockchain.
- Sensitive or personal data can be seen by anyone.
- Pseudonymised data helps relieve this risk but algorithms have been developed that can revers engineer anonymous databases with 99.98% accuracy.

Immutability of the Chain

- Under GDPR users have the right to be forgotten.
- This is infeasible with Blockchain, as once the data has been added to the chain it cannot be removed or changed.
- The only exception to this rule is through the power of a central organisation that runs the blockchain, however, it will undermine consumer trust as the blockchain is not truly immutable.
- In the case of fraudulent transactions, they cannot be undone. Typically, a bank can resolve this issue if you inform them. However as there is no authority in Blockchain, once the funds are transferred, they are gone.
- A central organisation could undo this, but once again it will degrade the trust of users.

Immutability of the Chain (2)

- In niche cases where someone may wish to change their name, they will always be attached to their old identity. If this person is being persecuted they may desperately want to change their details on the chain. The best way to mitigate this issue is to use a minimal set of identifiers on the chain. Ideally this should increase the difficulty of reverse engineering the details to a single person.

Encryption Lifespan

- As time passes, encryption algorithms that are currently considered strong will begin to be phased out, as computational power increases to a point where these algorithms are no longer recommended. As blockchain heavily relies on encryption; there is a danger that in the future encrypted information may become increasingly vulnerable to exposure.

Lack of Regulation

- In comparison to banks, stock exchanges and insurance brokers cryptocurrencies are considered to be incredibly under-regulated. This means highly unethical trading tactics such as wash trading are open to exploitation.
- These tactics are illegal in stock trading but not in cryptocurrencies as they are still in an inception period.
- In the meantime, trading should be approached with caution due to the risk of manipulation.

Single Point of Failure for Users

- If a user forgets their private/public key pair, they will no longer have access to their wallet. This means any funds in the wallet of the user is lost and cannot be retrieved.
- The more functionality a blockchain has, the larger the impact of loosing the keys, as the user will no longer be able to interact with these functionalities.
- There are methods to recover pairs using backup “phrases” or codes, however, if they are also lost then the keys cannot be recovered.
- Some systems store key pairs centrally; such that if a user forgets their keys, they can log into their account using 2 Factor authentication.
- A centralised system that can remind users of their keys is however also a risk – if a users key pair is stolen their funds and identity can easily be stolen as there are no additional security measures.

Volatility in Value

- Given the volatile value of cryptocurrencies for day-to-day, trading is highly speculative and somewhat a “gamble”.

Trusting the Blockchain

- The PoW consensus algorithm has been replaced with PoS in some implementations.
- PoS uses exponentially less power than PoW, but ultimately requires users to trust rich individuals to control consensus.
- In PoS nodes which own a certain amount of the currency are chosen randomly to add the next block to the chain.
- These nodes will not want to fool the system, as they stand to loose out because the currency they hold will devalue if the blockchains integrity is harmed.
- Ultimately users are burdened with the fact they must trust the richest nodes to be honest and not cheat the system, which enables new attack vectors which do not exist with PoW, such as the Bribe attack.
- PoW can be considered a trustless algorthim as the competition between nodes keeps the blockchain honest excluding cases such as 51% attacks.

University of Reading

Recap

- Topics Covered
  - Ethics
  - Law