Blockchain Technology and Applications

CS 731

Distributed Consensus

Dr. Ir. Angshuman Karmakar

IIT Kanpur

Teaching assistants

- **Sumit Lahiri (sumitl@cse.iitk.ac.in)**
- **Chavan Sujeet** (sujeetc@cse.iitk.ac.in)
- **Indranil Thakur** (indra@cse.iitk.ac.in)

A simple coin

- Use the cryptographic primitives learned previously

- Also the Bitcoin environment is anonymous
  - There is no real world identity
- Remember that each public-key is unique
  - Everybody in the network is addressed by its public-key
  - Or hash of the public-key

A simple coin

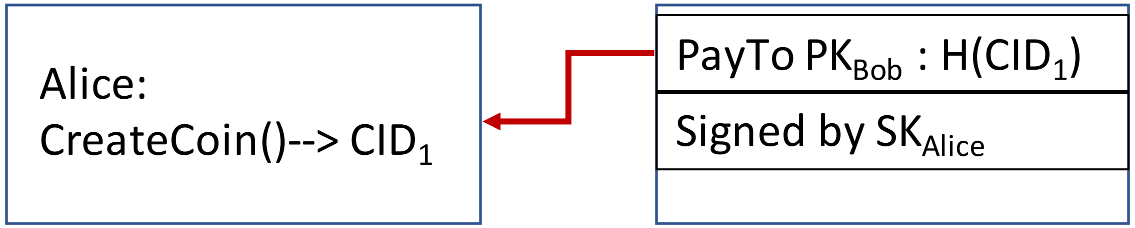
- Use the cryptographic primitives learned previously
  - Alice creates first coin of this cryptocurrency
  - Genesis block

Alice:
CreateCoin()--> CID_1

[1]From Arvind Narayanan Et al's Britcoin and cryptocurrency technologies

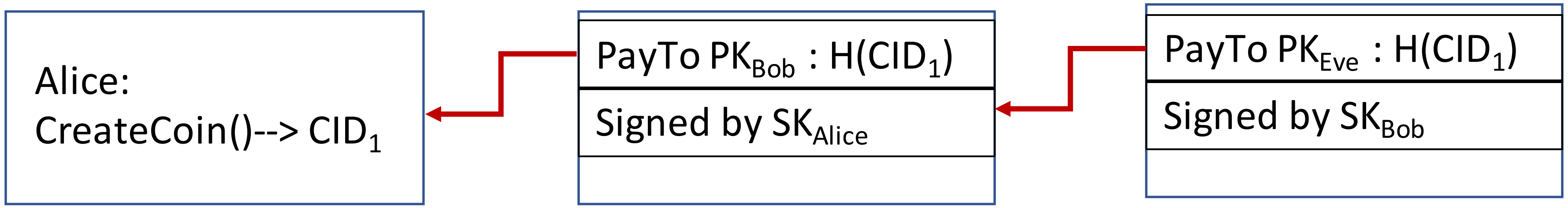
A simple coin

- Use the cryptographic primitives learned previously
  - Alice creates first coin of this cryptocurrency
  - Genesis block
  - Alice wants to pay Bob



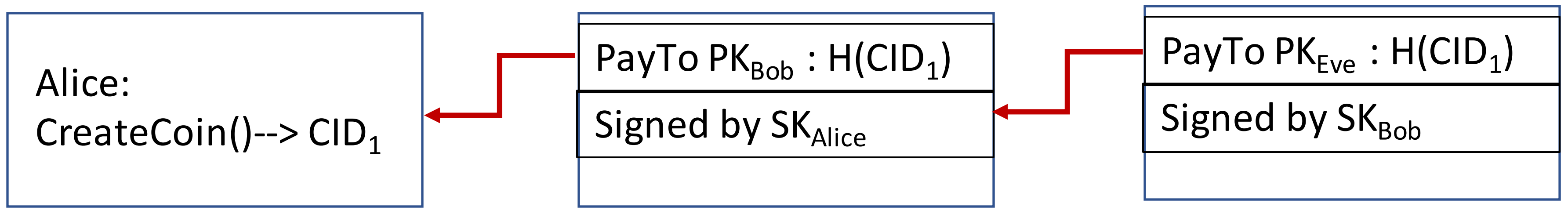
A simple coin

- Use the cryptographic primitives learned previously
  - Alice creates first coin of this cryptocurrency
  - Genesis block
  - Alice wants to pay Bob
  - Bob may further send it to Eve



A simple coin

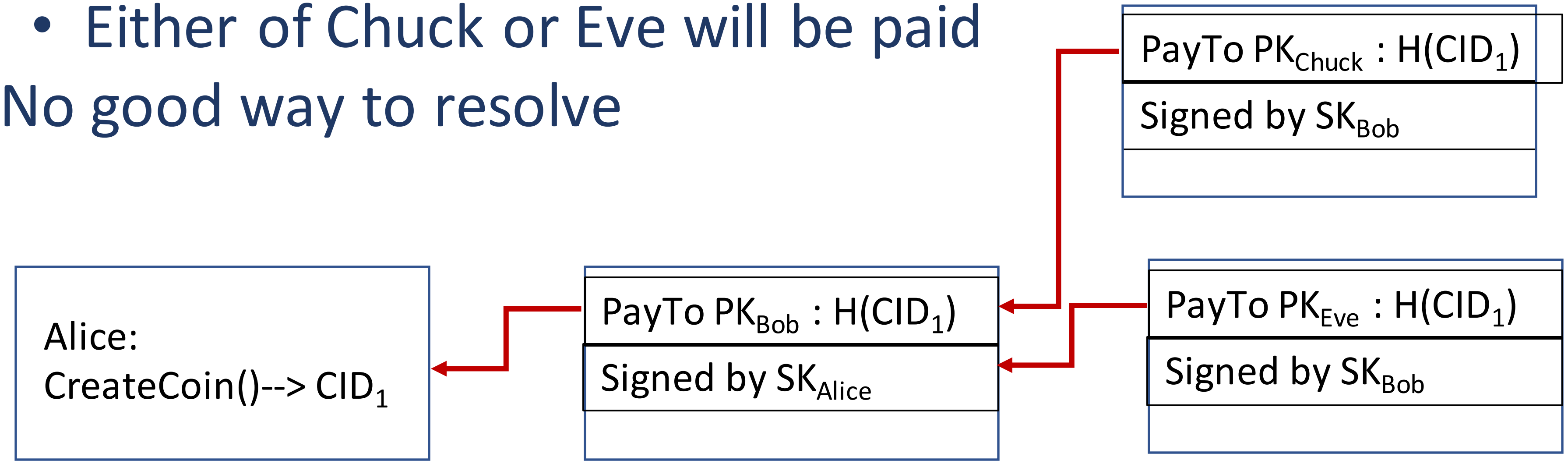
- Rules of Simple coin
  - Alice can create a coin (subject to some restrictions)
  - Owner of a coin can pass it to someone else
    - Statement/transaction signed by the owner
  - To verify a transaction, follow the transaction to its origin
    - Verify the signatures along the way



A simple coin

Problem

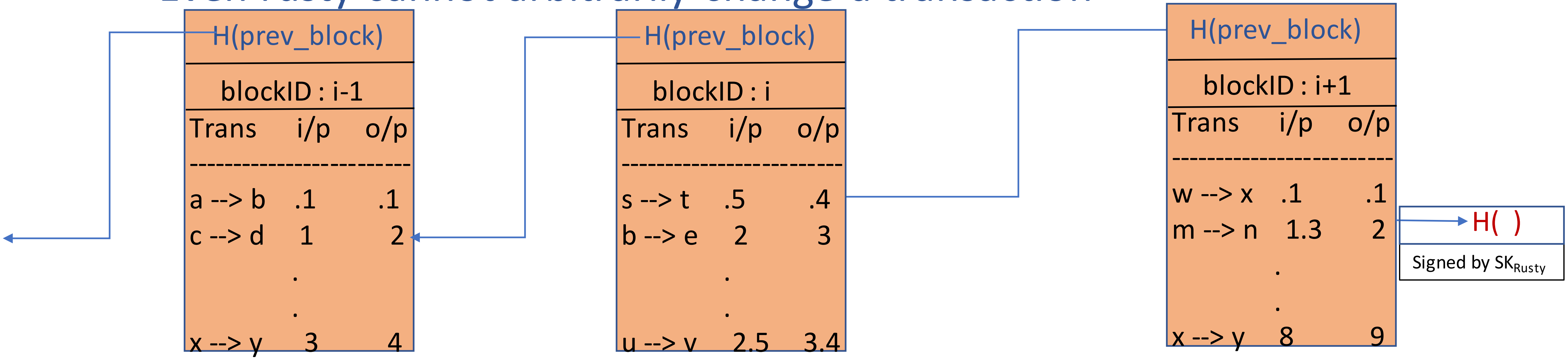
- Double spending
  - Bob after paying to Eve
  - Can sign another statement "PayTo" Chuck with same coin ID
  - Issue in e-cash transactions
- To a verifier both transactions are valid
- Leads to the same genesis block
  - Either of Chuck or Eve will be paid
- No good way to resolve



A simple coin

One solution

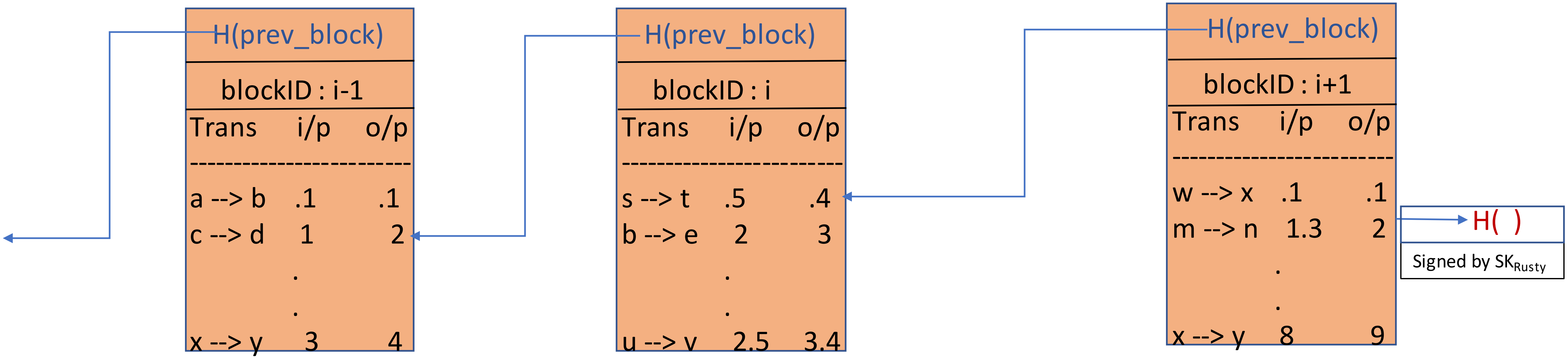
- Use a blockchain
- Trust a designated user "Rusty"
- Rusty adds the transactions to the blocks
- Rusty signs the final "hash pointer"
- Ensures append-only property
- Even rusty cannot arbitrarily change a transaction



A simple coin

One solution

- Rules
  - Consumed coins are valid--> created in some previous transaction
  - Not consumed before (No double spend)
  - Total value of coins in a transaction = total value of coins out a transaction
  - The transactions have valid signatures by the owners of consumed coins



A simple coin

One solution

- A transaction is only accepted if it is present in a block
    - Recipient is assured of the payment
- Can a transaction be faked?
- Can a previous transaction be changed?
- Double spend?
- Centralized
    - Rusty can blacklist entities
        - Deny service, make coins unspendable
    - Can stop updating the blockchain
    - Can demand more money to endorse a transaction
    - Etc..
- How does bitcoin achieve decentralization?

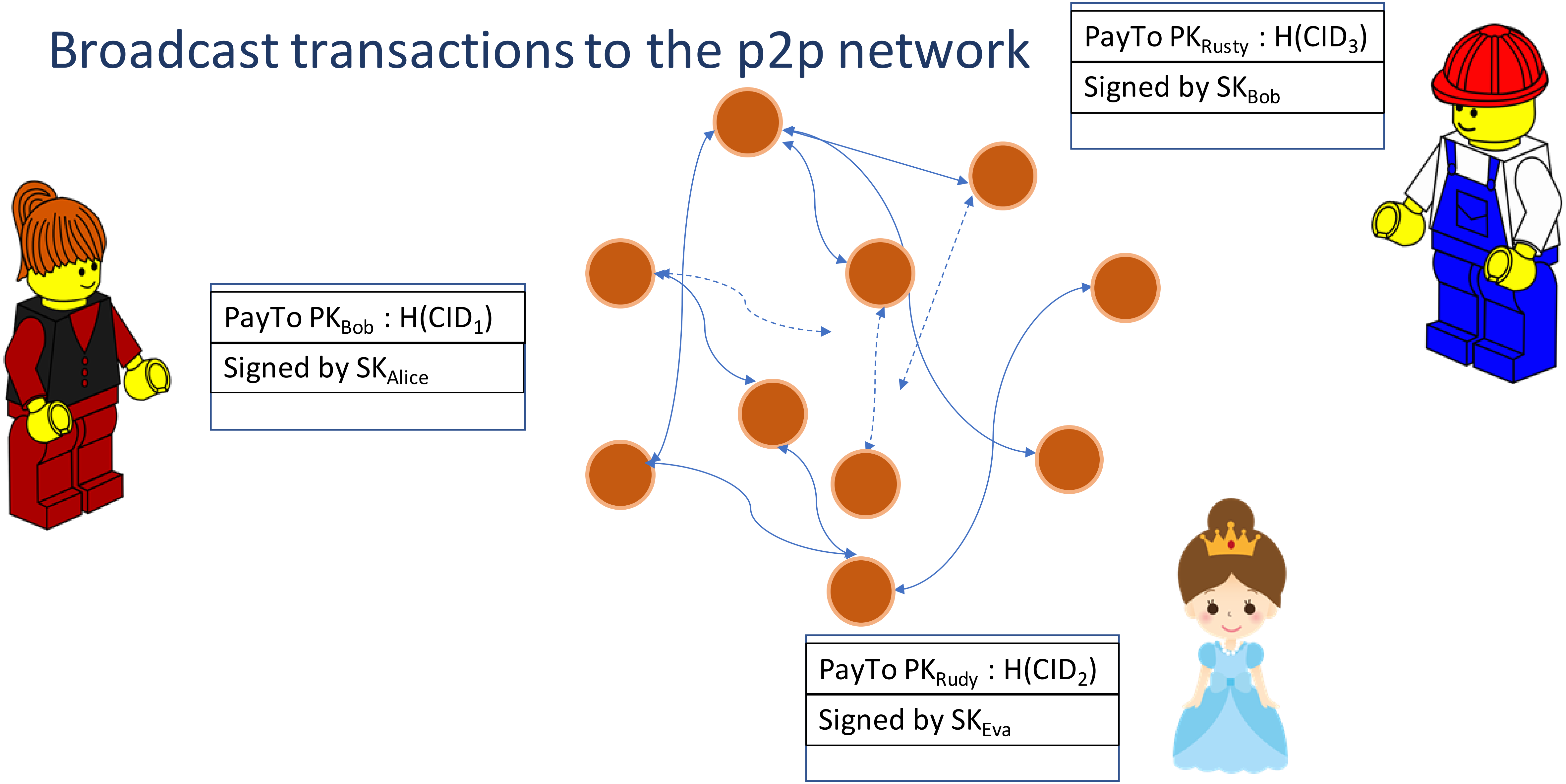
Decentralization

1. Maintenance of the ledger of transactions?
2. Who adds the transactions to the blocks?
3. Who validates transactions?
4. Who creates new Bitcoins?
5. Who determines or approves updates?
6. Price of Bitcoins?

Distributed consensus

Consensus

- Broadcast transactions to the p2p network



Consensus

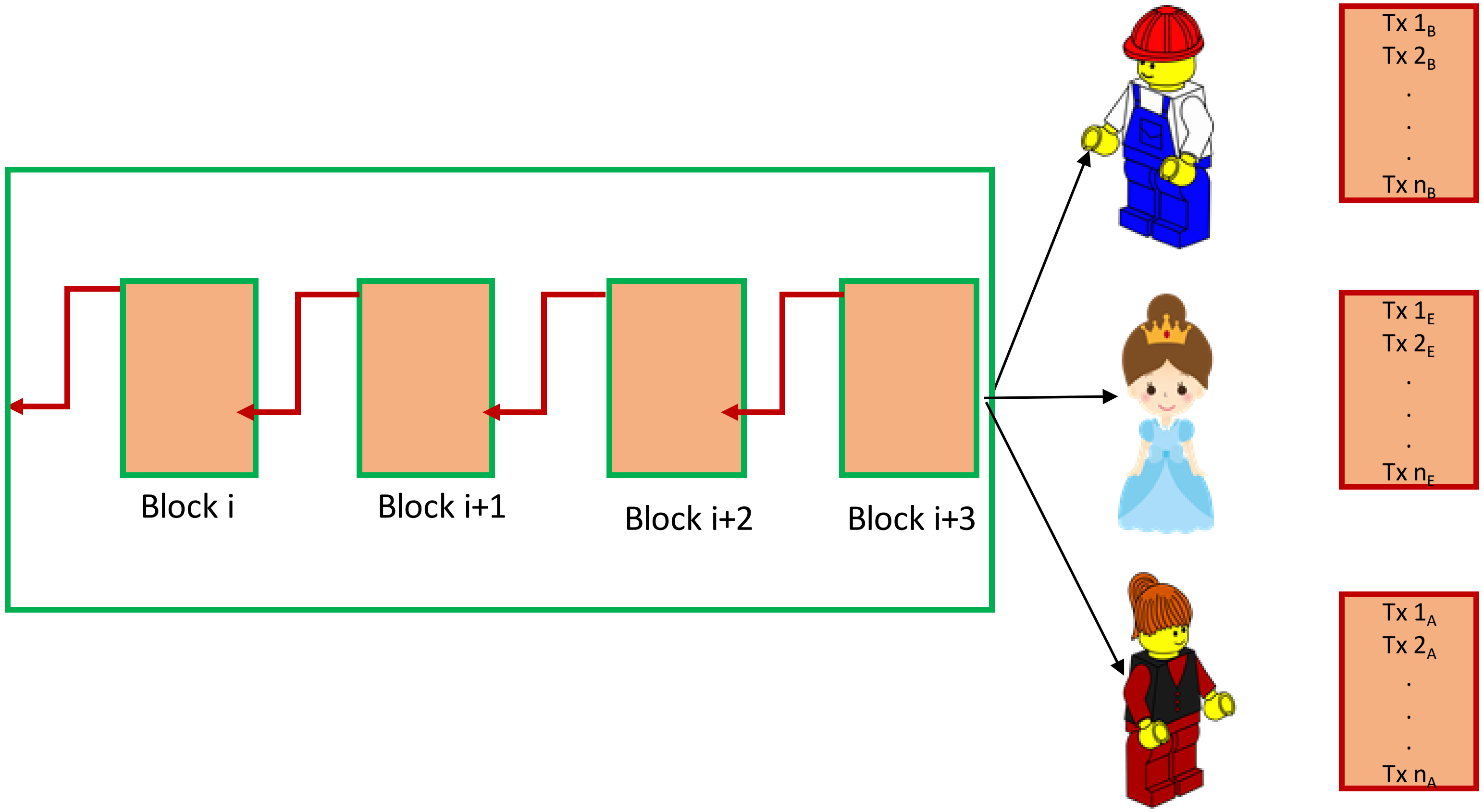
- We want to build a single global ledger
- We need consensus on

1. Which transactions were broadcast?
2. What is the order of this broadcast?

- Straightforward solution
- Global timestamps
  - Very difficult

Consensus

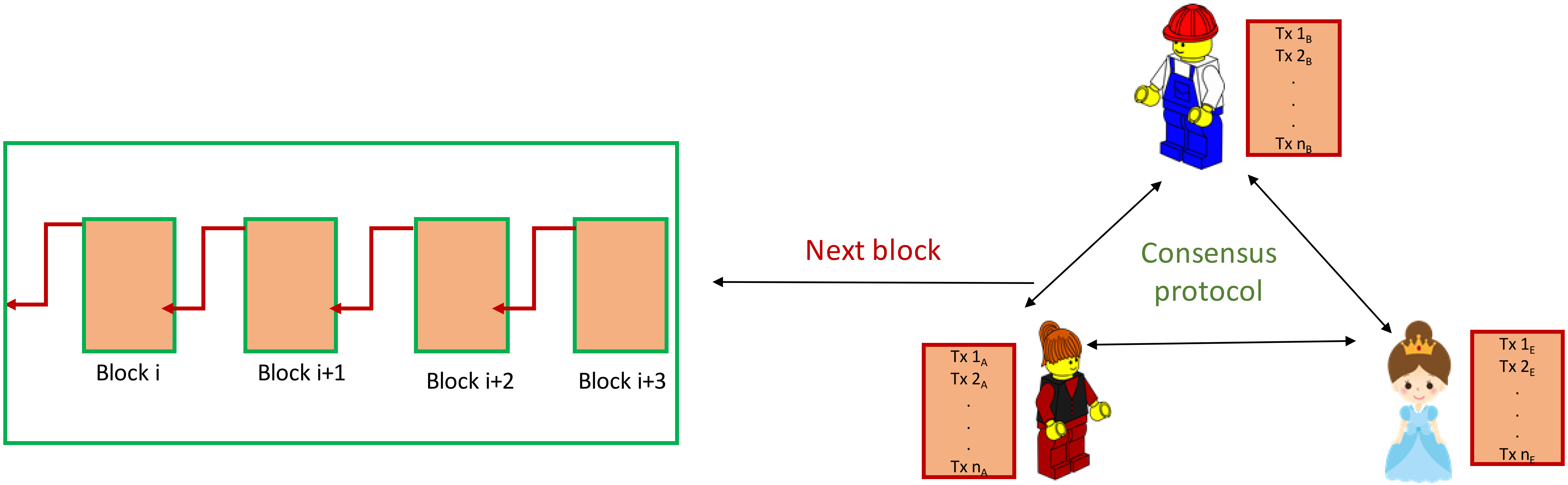
- At any time, any node has
  - A blocks of transactions they have agreed
  - A series of transactions where they are yet to reach consensus



Imperfections in the p2p network due to network latency, crash, etc.

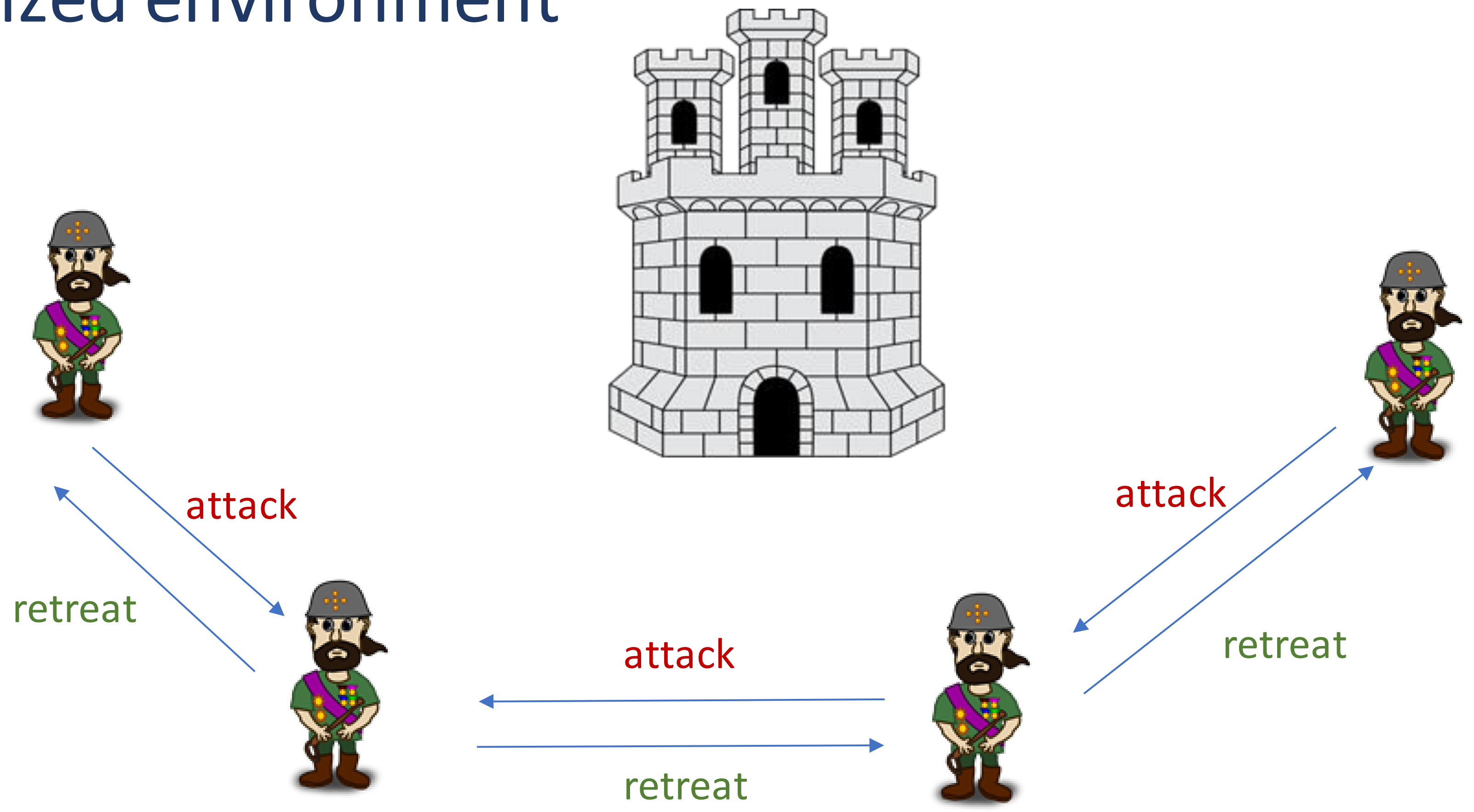
Consensus

- At regular intervals after let's say n minutes,
  - Nodes participate in a consensus protocol
  - Everybody agree on a block one of the node proposes



Distributed Consensus

- A procedure to converge to decision in distributed or decentralized environment



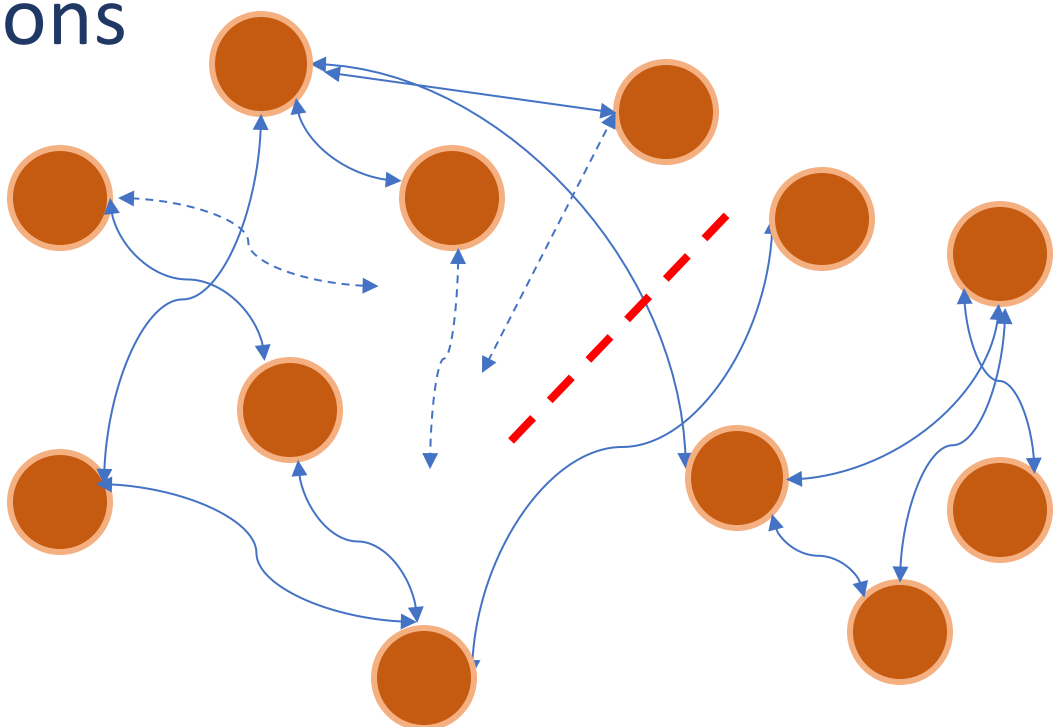
Distributed Consensus

- Faulty nodes might be present in the system
  - Reliability
  - Fault tolerance
- Ensure correct operation even in the presence of faulty nodes

- Example
  - Commit transaction in a database
  - State machine replication
  - Clock synchronization
  - etc.

Distributed Consensus
Faults

- There can be different types of fault in a distributed system
- Crash Fault: A node crashes or becomes unresponsive during the execution of the protocol
- Network fault: A network issue arises and divides the two nodes into two or more partitions

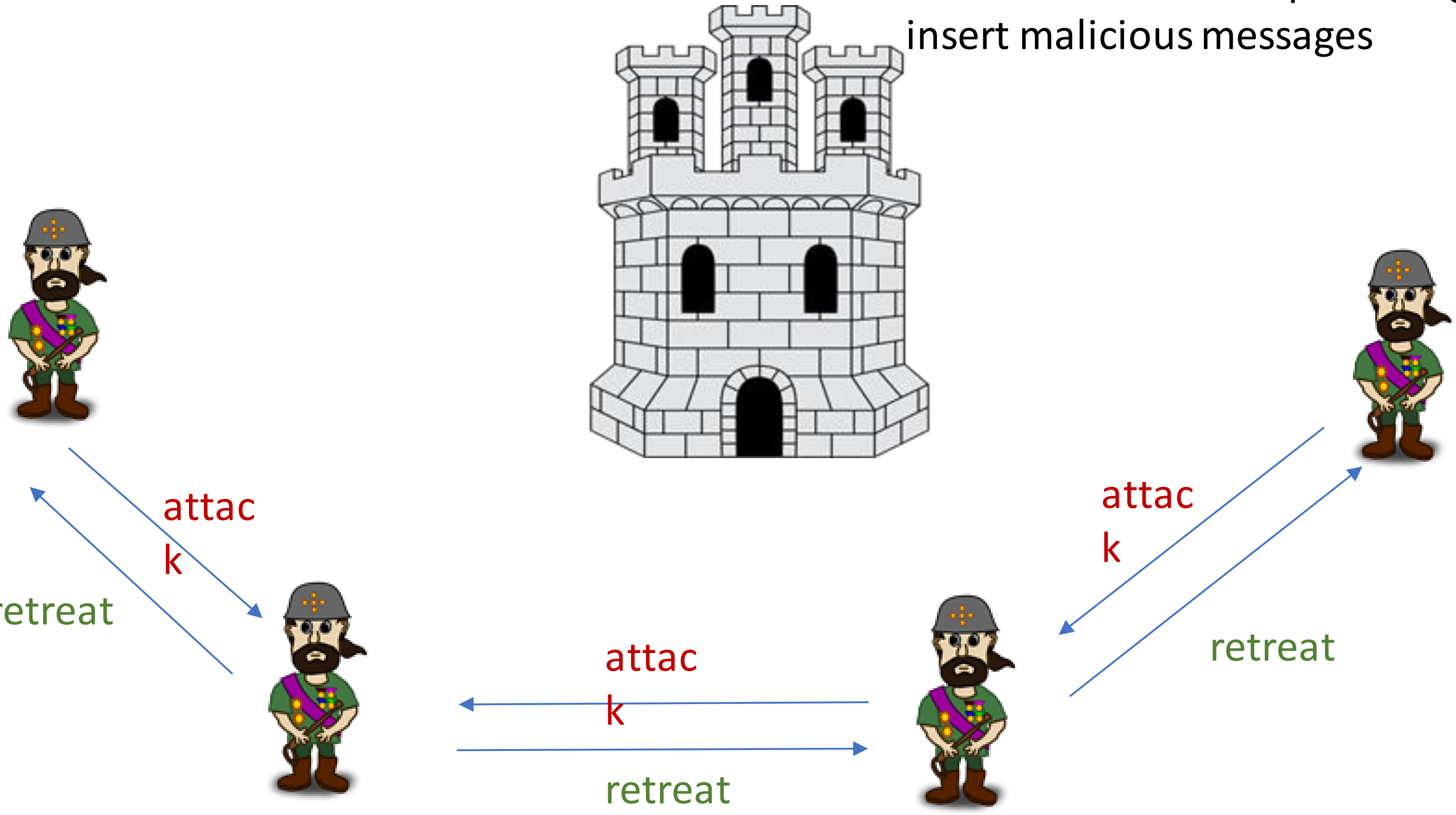


Distributed Consensus
Faults

- Byzantine fault: The node starts behaving maliciously

Defender can intercept messages and insert malicious messages

- Can only win if they all attack at the same time
- No secure communication



Distributed Consensus
Properties

- Termination: The consensus protocol terminates and every honest node decides on some value at the end of the protocol
- Validity: If all the nodes propose the same value than all the honest nodes agree on that value
- Integrity: Every honest node decide on at most one value, and the value must be proposed by some other honest node
- Agreement: Every honest node must agree on the same value

Distributed Consensus

Properties

- In Summary, a distributed consensus protocols should satisfy the following two conditions,
  - It must terminate with all honest nodes agreeing to a value
  - The value is proposed by an honest node

Distributed Consensus

Synchronous vs. Asynchronous systems

- Synchronous Message Passing System
  - The message must be received within a predefined time interval
  - Strong guarantee on message transmission delay
- Asynchronous Message Passing System
  - No upper bound on message transmission delay or message reception time
  - Message can be arbitrarily delayed for arbitrary period of time

Distributed consensus

Impossibility results

- Byzantine generals' problem
  - Synchronous messages
  - Impossible to reach a consensus if number of dishonest generals ≥ n/3
- Fischer-Lynch-Paterson[1]
  - Nodes act in a deterministic manner
  - Purely asynchronous system
  - Impossible to reach a consensus even with a single dishonest node
  - Most influential paper in ACM PODC 2001

[1] Fischer *et al.* Impossibility of distributed consensus with one faulty process. https://dl.acm.org/doi/10.1145/3149.214121

Distributed consensus
Impossibility results

- Paxos protocol[1]
  - Makes certain compromises, activity level of participants, numbe
    of messages received and sent, delays, etc.
  - Never produces an inconsistent result
  - May get stuck without any progress

[1]The Part-Time Parliament Leslie Lamport .http://lamport.azurewebsites.net/pubs/lamport-paxos.pdf

Distributed consensus

Correctness Properties

- Safety
  - Honest nodes must not agree on an incorrect value
    - Nothing bad happened

- Liveness
  - Every correct value must be accepted eventually
    - Something good eventually happens

Distributed consensus

Consensus in an Open System

- The traditional distributed consensus protocols are based on
  - Message passing
    - Nodes are connected over the internet
  - Shared memory
    - When a common memory is available to read and write the shared variables which everyone can access
- Message passing needs a closed environment
  - Identity of each node is known

Distributed consensus

Consensus in an Open System

- There is no shared memory in the internet
    - Where is the shared memory located?

- Bitcoin is an open environment
    - Anyone can join or leave Bitcoin anytime
    - How do we overcome impossibility results and achieve consensus in such an system?

The End !!