Blockchain Replication:

The Whys and The Hows

Replicating Smart Contracts for Dependability

Miguel Pupo Correia

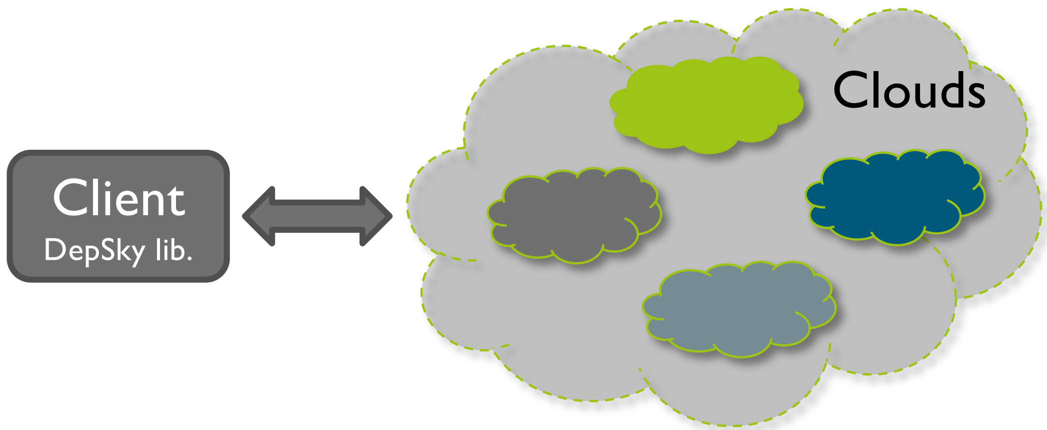
IFIP WG 10.4 83rd meeting – Melbourne
January 2023

inesc id lisboa
20 YEARS
DEFINING TECHNOLOGY

TÉCNICO LISBOA

Motivation: Cloud storage replication – DepSky

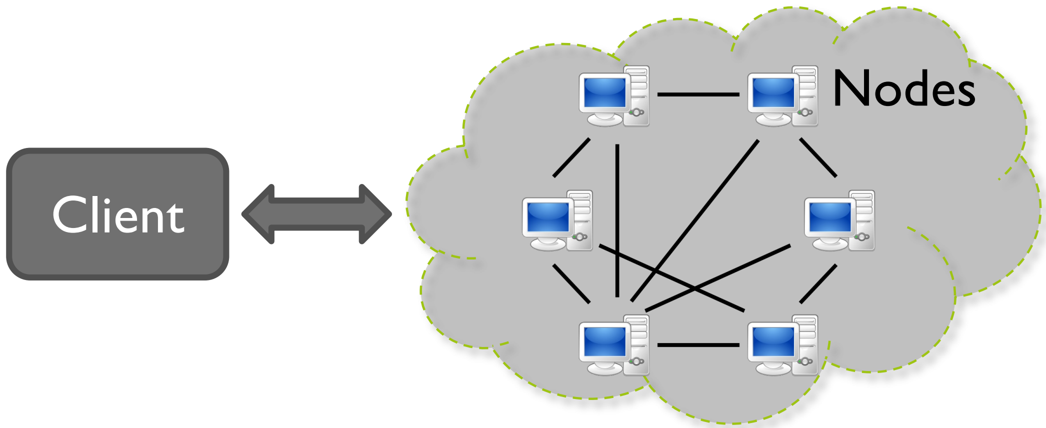
- Multi-cloud storage: client-side library that accesses clouds using a BFT quorum protocol
  - Benefit 1: dependability even if *f* clouds fail
  - Benefit 2: enhance the dependability provided by individual clouds



few clouds as clouds are trustworthy

Replication in a Blockchain

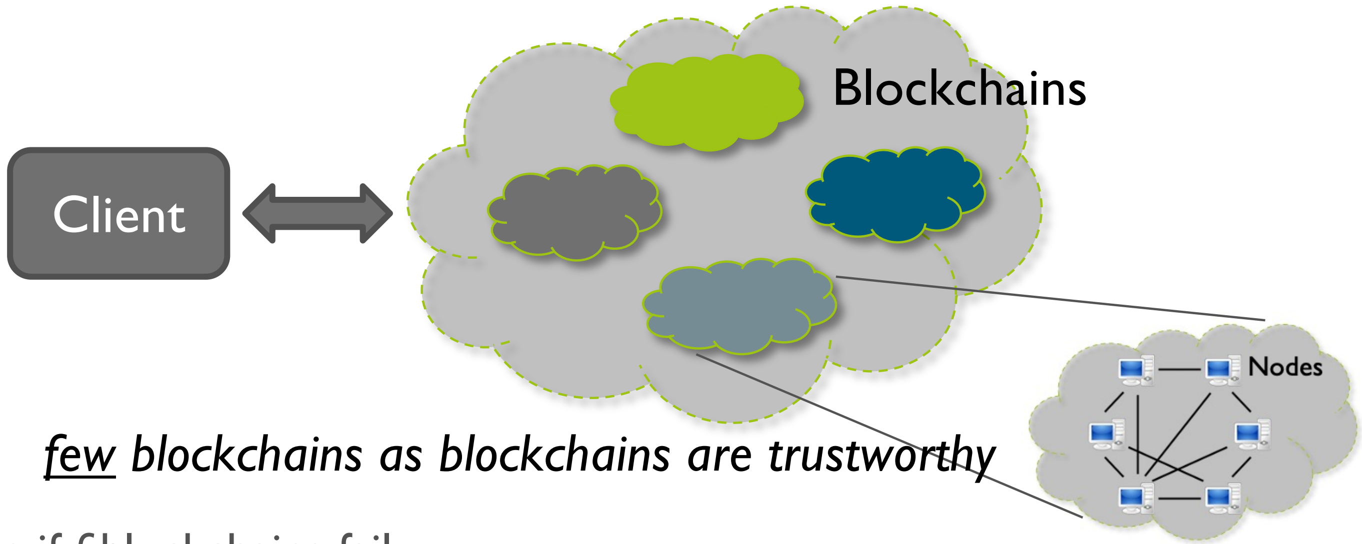
- Client accesses nodes that run a BFT consensus protocol (PoW, PoS, classical SMR, …)
  - Benefit: a dependable system out of untrusted nodes



<u>many</u> nodes as nodes are not trustworthy in permissionless blockchains

Today: smart contract replication

- Client accesses different blockchains
- Contracts replicated in several blockchains instead of just one



__few__ blockchains as blockchains are trustworthy

- Benefit 1: dependability even if *f* blockchains fail
- Benefit 2: enhance dependability provided by individual blockchains
- Benefit 3: allow using low(er) quality blockchains: Blockchain-of-Blockchains

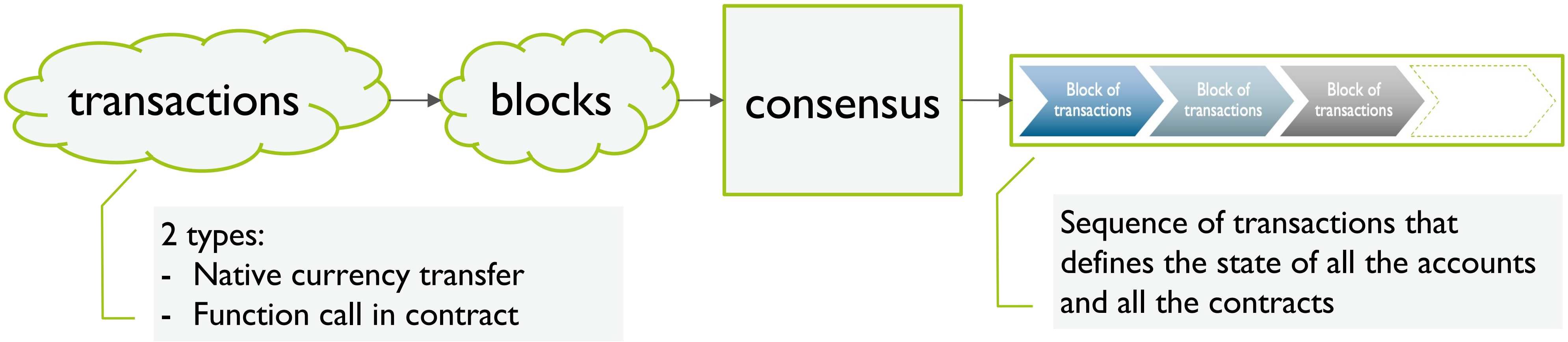
Outline

- The problem
- Preliminaries
- V1: Register Contract Replication
- V2: Generalized Contract Replication
- Key takeaways

The problem

Permissionless Blockchains

- Bitcoin, Ethereum,…



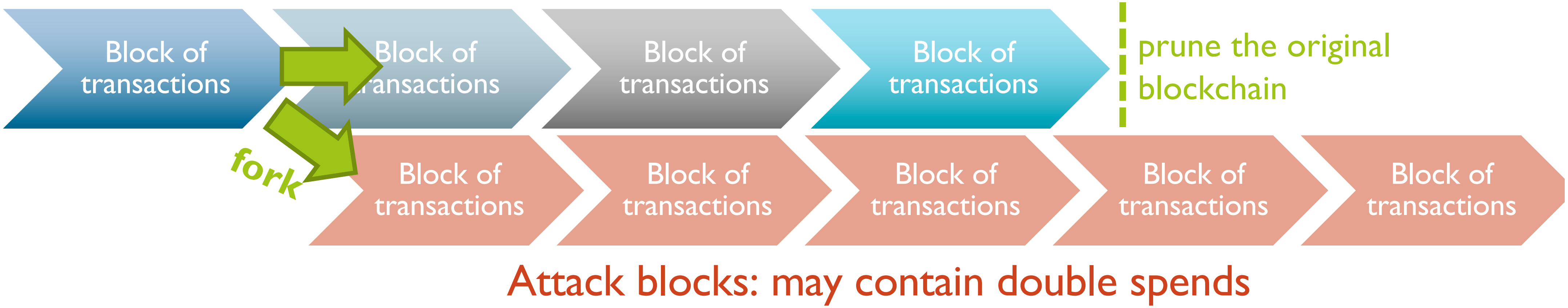
Proof of Work (PoW)

- "If a majority of CPU power is controlled by honest nodes, the honest chain will grow the fastest and outpace any competing chains." (Nakamoto's Bitcoin whitepaper)

- What if a majority of CPU power is controlled by malicious nodes?

Chain reorganization / 51% attack

- Attacker creates new blocks at depths ("positions") already considered stable and manages to prune the original chain:



Byzantine failure: state of the system is modified!

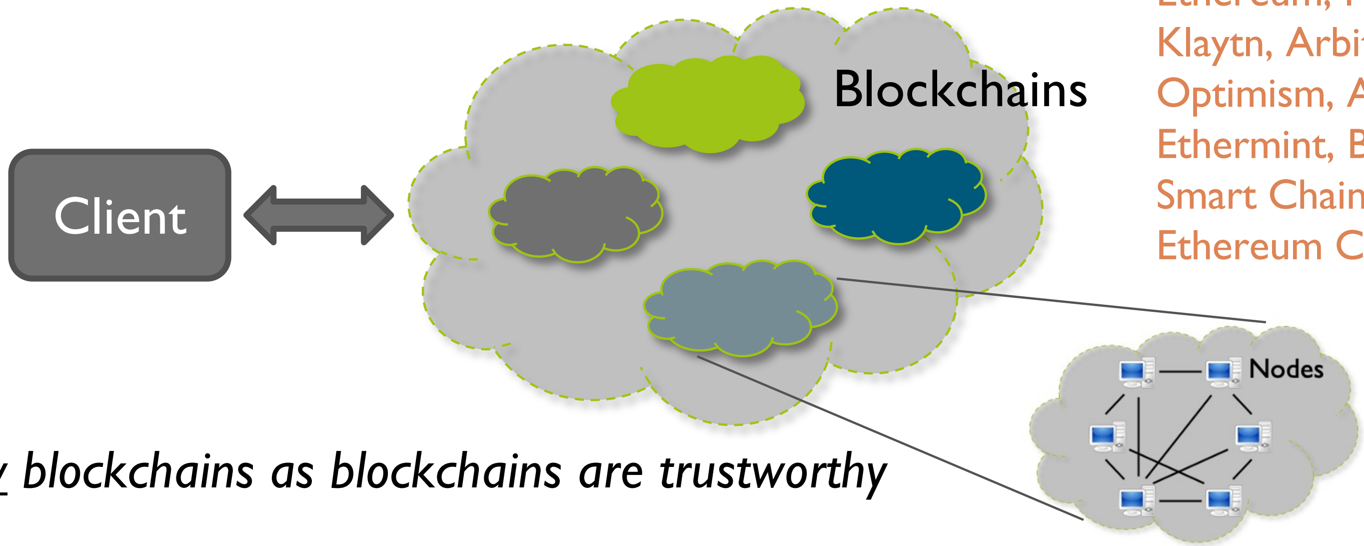
Are these attacks possible?

- Not if the blockchain system is “huge”, e.g., Bitcoin
  – ~14K nodes and more than 2×10^{20} hashes per second
- Possible with smaller blockchains:
  – Bitcoin Gold (Bitcoin hard fork 2017)
    - May 2018: ~18M USD double-spent; 76 nodes
  – Ethereum Classic (Ethereum hard fork 2016)
    - Jan. 2019: 15 reorganizations, ~1M USD double-spent; 532 nodes
- Proof-of-Stake:
  – Same problem in smaller blockchains, i.e., if the stakes are not high enough

Preliminaries

Today: Blockchain / contract replication

- Client accesses different blockchains
- Contracts replicated in several blockchains instead of just 1



Ethereum, Polygon, Klaytn, Arbitrum, Optimism, Avalanche, Ethermint, Binance Smart Chain, Ethereum Classic, …

few blockchains as blockchains are trustworthy

Challenges for contract replication protocols

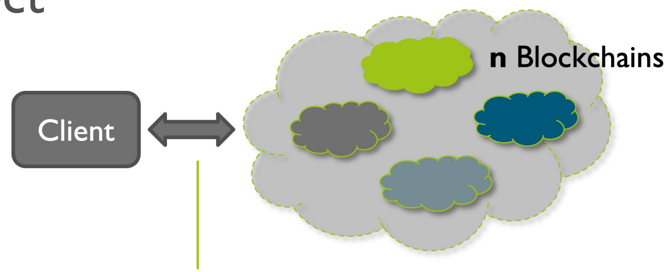
- Blockchains are distributed machines, not individual servers
- Blockchains can't be modified (only contracts can be added)
- Contracts can't communicate with contracts in other blockchains
- Contracts can't sign data
- Operations on contracts have weak finality
- Native cryptocurrencies have different prices

- Minor: interoperability, as Blockchains and contracts are heterogeneous
  - Solved considering single VM (e.g., EVM) and a client-side library

Parameters

- System-wide:
  - **n** – number of blockchains used for replication: $B_1, B_2, \ldots B_n$
  - **f** – maximum number of faulty blockchains (out of n)

- Blockchain-specific:
  - **a** – min. num. nodes to access for operation to be correct (a=1 if client trusts or runs the node)
  - **d** – min. depth for block to be final



a nodes accessed in each blockchain, waiting for depth to be >= **d**

Assumptions

- Blockchains: no more than **f** blockchains can be faulty
- Clients: always correct; follow the protocol and private keys are not disclosed
- Clients and nodes communicate through authenticated reliable channels
- Operation requests are authentic and non-repudiable (signed)
- Cryptographic schemes are trusted
- Contract starts created in all blockchains and in the same state

V1: Register Contract Replication

Simplifications of v1

- Constraints on the data stored in the contract:
- Data is self-verifiable
- Just reads and writes over individual registers
  - SC is as a multi-writer, multi-reader multi-register
  - Consensus number 1

Contract

- Contract that stores document data (for many docs)
  - Not the full documents (expensive)

- SC stores the following data for each document:
  - Doc ID
  - Doc authenticator (hash)
  - Other document metadata
  - Signer ID
  - Short Signature of doc-data
  - Version of the document

doc-data

sign-data

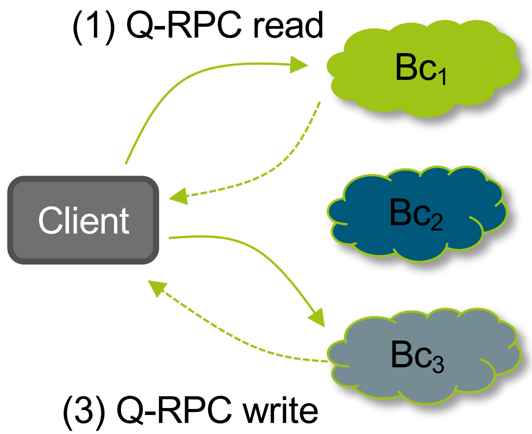
Protocol

- BFT quorum protocol
  - Quorum – set of subsets of blockchains, e.g., all sets of **n-f** blockchains
  - Clients communicate with quorums of blockchains
- Basic primitive:
  - Q-RPC(op, valid()) – invokes operation **op** in replicas of the contract until
    - there are replies (rep) from **a** nodes, with depth at least **d** for each blockchain
    - that satisfy the predicate **valid(rep)**
    - for **n-f** blockchains

Protocol – write

- Write doc-data
  - (1) Q-RPC read version of the doc-data stored; valid() checks the signature
    - Replicas return the highest version, using Signer ID to break ties
  - (2) new-version = max{versions}+1 *or* 0 if none
  - (3) Q-RPC write doc-data with version new-version

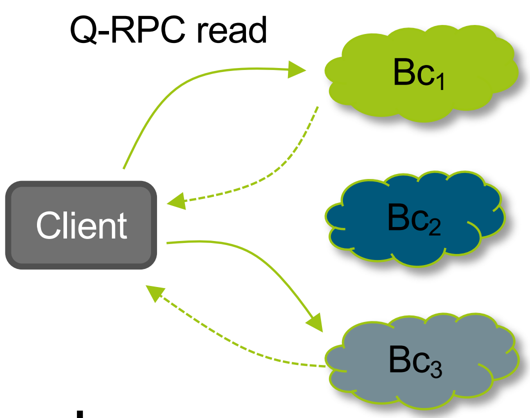
  - The protocol ensures **n-f** blockchains store the latest version
    - For **f**=1 and **n**=2f+1=3 → n-f=2 blockchains



Protocol – read

- Read doc-data
  - Q-RPC read version of the doc-data stored; valid() checks the signature
  - return doc-data corresponding to max{versions}

- The protocol ensures that
  - candidate doc-data values come from **n-f** blockchains,
  - which must intersect with the **n-f** in which it was written,
  - so the version returned must be the most recent



- NB: the "value of the register" is that returned by read

Consistency

- Consistency = Regular
  - a read concurrent with two or more writes returns any of the values being written or the previous value
  - n >= 2f+1

Replicated register contract

- Data structure:
  - Table (map) indexed by Doc ID (doc-id) and containing the data above

- Methods:
  - Implement the SC functionality & the BFT quorum protocol
  - registerDoc(doc-data, sign-data, version) – write protocol
  - getDoc(doc-id) returns doc-data, sign-data – read protocol
  - deleteDoc(doc-id) – write protocol

V2: Generalized Contract Replication

Token contracts

- **Token** – blockchain-based abstraction that can be owned
  - Represents some asset: collectible, identity, resource,…
  - Created and managed in contracts; structure usually standard:
    - ERC20 – fungible tokens
    - ERC721 – non-fungible tokens (NFCs)

- All have functions like:
  - Balance of the contract
  - Transfer token

Replicating tokens – challenges for v2

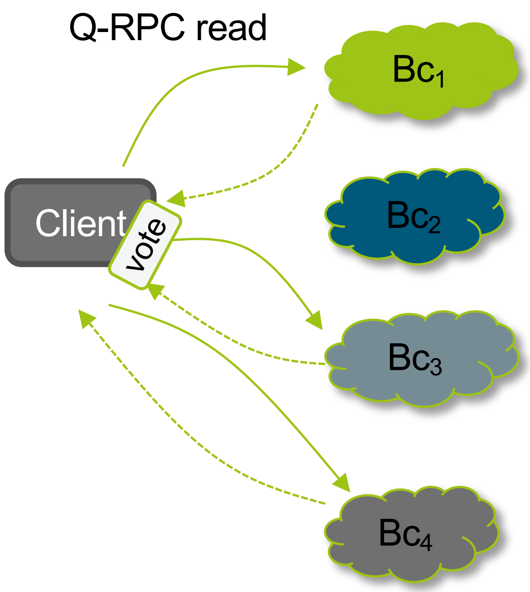
- Data is not self-verifiable
  - e.g., token balance is just a number
- Operations on multiple variables and not idempotent
  - consensus number > 1
- Replicating payments in cryptocurrencies
- Dealing with faulty clients

Data not self-verifiable

- Example – variable is an integer (from ERC20):
  - `balances[_to] += _value;`

- Solution: modify protocol to not require self-verifiable data

- Read/write protocols & Q-RPC similar with:
  - **n >= 3f+1** and the result is the most voted
  - Quorum intersections must have at least **2f+1** blockchains, so that a majority is correct



Operations on multiple variables: problem

- Example from ERC20:
  - Moves _value tokens from caller's account to account _to; returns a Boolean (success yes/no)

```
function transfer(address _to, uint256 _value) … {
    …
    balances[msg.sender] -= _value;
    balances[_to] += _value;
    …
    return true;
}
```

operation over 2 variables

Ops on multiple variables: solution

- Accept that replicas (updated with Q-RPC) will converge later

- CCRDTs – Computation Conflict-free Replicated Data Types
  – Data types that allow operations over updates (e.g., integer inc./dec.) +
  – Replicas converge to the same result when all operations are applied

- We model the contract state as a CCRDT

Ops on multiple variables: CCRDT

- Contract state modeled as a single multi-register
  - There is a single version number used for reads/writes
  - All write/update operations are stored on a queue
  - All operations are executed when received

- Data type = multi-register composed of registers of:
  - Numeric types with a single operation: addition
    - Addition is commutative => two sequences of the same additions over the same initial value give the same result, independently of the order
  - Numeric or non-numeric types with single operation: assignation

Ops on multiple variables: inconsistencies

- Clients access n-f replicas => (temporary) inconsistencies:

Replica	Op1	Op2	Op3	Op4	…
SC1	W1	W2	R1		
SC2		R1	W2	W3	
SC3	W1			W3	

- Owner periodically sends missing operations to the replicas
  - QueueCleanUp protocol: gets queued ops from replicas and updates

Dealing with faulty clients

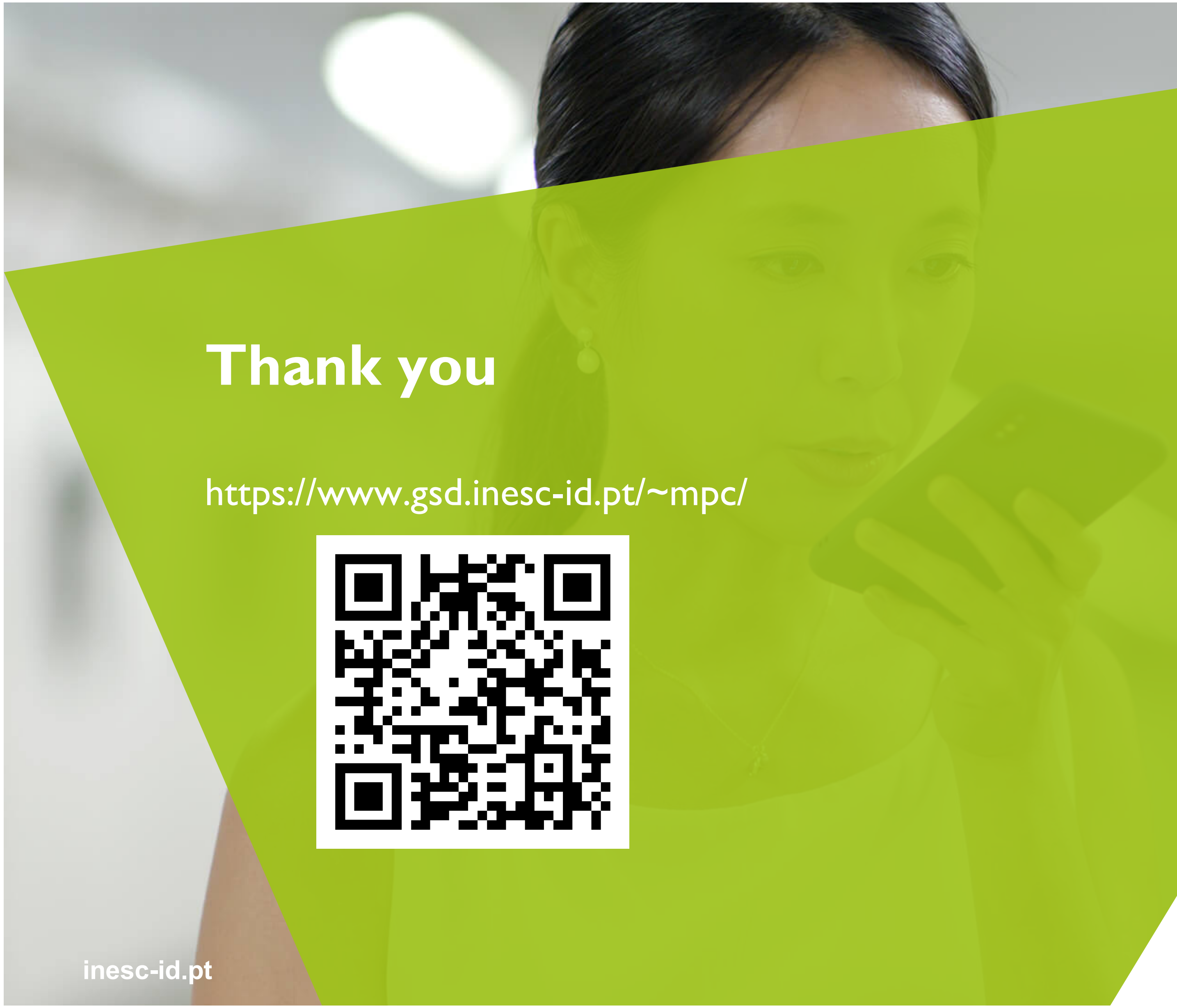
- Owner
  - Substitute it by a Decentralized Autonomous Organization (DAO)
  - i.e., a contract in which actions are decided cooperatively, e.g., by voting

- Other clients (e.g., buyers):
  - Owner or DAO uses queues returned obtained by the QueueCleanUp protocol to detect faulty clients
    - e.g., that write different values in different replicas
  - Q-RPC to function BlockClients to add faulty clients to a blacklist

Key takeaways

Key takeaways

- A first shot at replicating contracts in different Blockchains
  - To increase dependability and/or allow using smaller Blockchains
- Challenges
  - Many: limited server-side code, not possible to modify blockchains, contracts can't communicate or sign, …
- Key technical contributions
  - Fitting Byzantine quorum protocols in the constraints of Blockchain / SCs
  - Combination of Byzantine quorum protocols with CCRDTs



Thank you

https://www.gsd.inesc-id.pt/~mpc/

inesc id lisboa 20 YEARS

TÉCNICO LISBOA