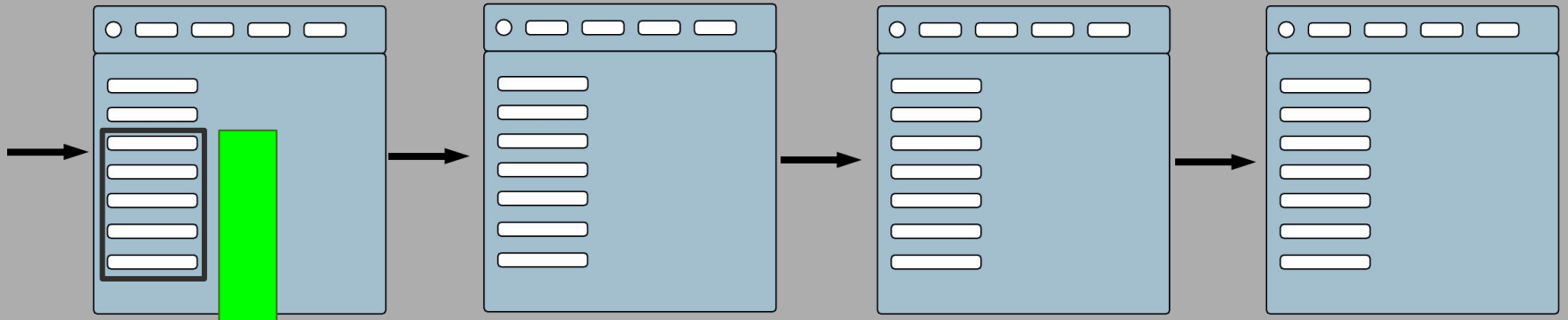
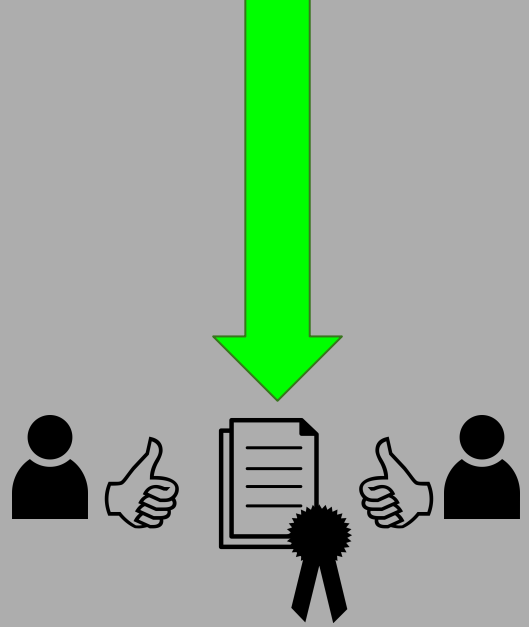
Fully Abstracted State Channels: an object oriented approach

Reminder:
How state channels work





State channels move **on-chain** operations (requiring many parties) **off-chain** (requiring just a few)

while maintaining or improving the guarantees of the blockchain

Question:

How can we apply the state channel model to a wide variety of problems without having to build individual implementations for each application?

Some properties we would like our solution to have:

- The solution should serve, unmodified, for **many different kinds of state channel applications**
- As far as possible, the design should remain **trust free**
- For the sake of privacy, cost, and responsive performance, we would like to **minimise on-chain storage and operations**
- Channels should be **composable** so that any application for a single channel will also work across several combined channels
- Operations should be **parallelisable** so that we can close out or rearrange some operations without disrupting others

(continued)

- It should be possible to leave **on-chain components unchanged for year-like timescales**
- Correspondingly, it should be possible to **initiate new applications** within an existing state channel **without requiring any on-chain operation**
- Use of a state channel **should not reduce the privacy of participants**
- As far as possible, removal of whole or partial state deposits from a state channel should be **indistinguishable from normal transactions**.
- The state tracked by the channel, and the potential obligations of its participants **should not needlessly balloon over time**

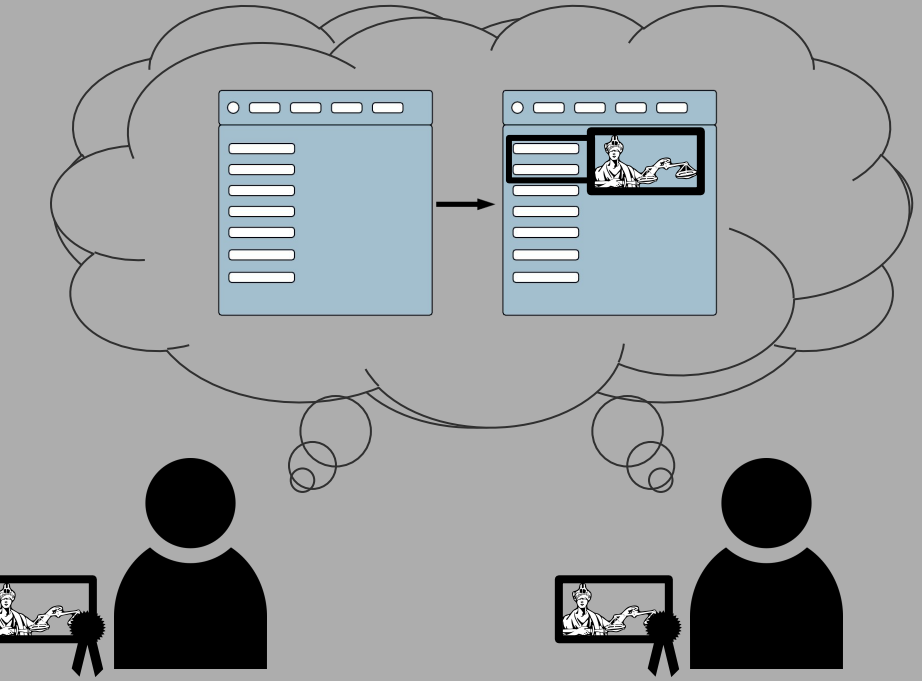
Answer:

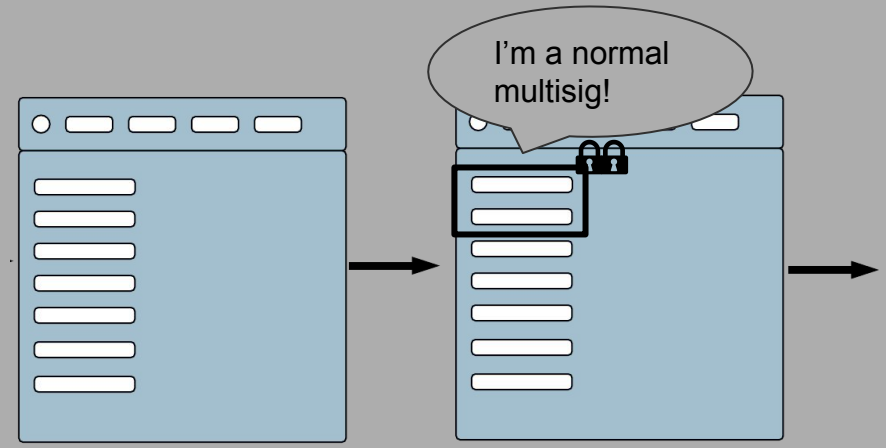
Counterfactual instantiation!

Or, “how to act as if what *hasn’t* happened is *already true*”

Disclaimer: if counterfactual instantiation *were* right for you, results *would* be guaranteed. Not all contracts can be counterfactually instantiated. Do not use counterfactual instantiation as a substitute for mechanism design. Please consult with your resident cryptoeconomist to see if counterfactual instantiation is an appropriate solution for your problem. Counterfactual instantiation does not solve the halting problem or provide infinite computing power. Counterfactual instantiation cannot answer a broken question. If learning more about counterfactual instantiation results in obsessive behaviour...welcome!

Basic idea: use incentives to make both parties act *as if* there was already a contract in the blockchain, even though there isn't

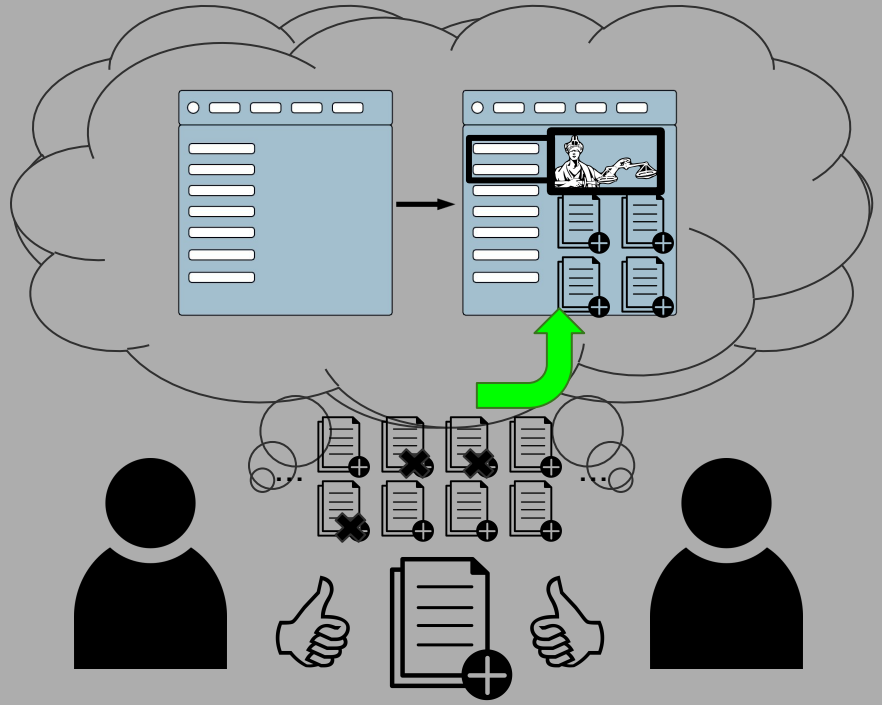


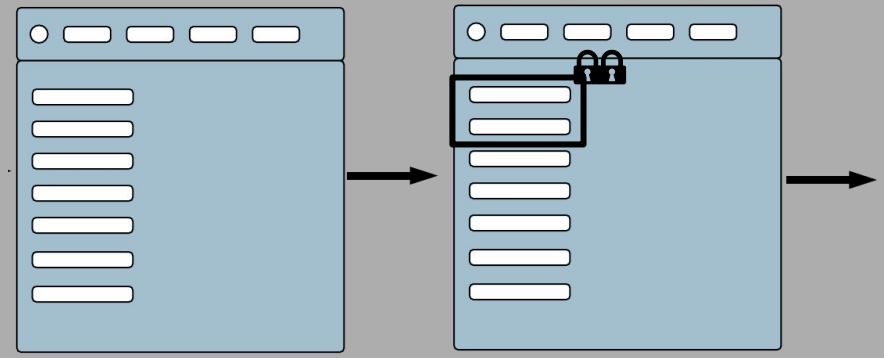


Give both parties the ability to put it in the blockchain if they *need* to (via presigned tx's that will create it)

But also give both parties the incentive *not* to put it there unless they *have* to (via fees and penalties)

By coming up with a general strategy for “counterfactual instantiation” of contracts, we allow the parties to add and change whatever they want





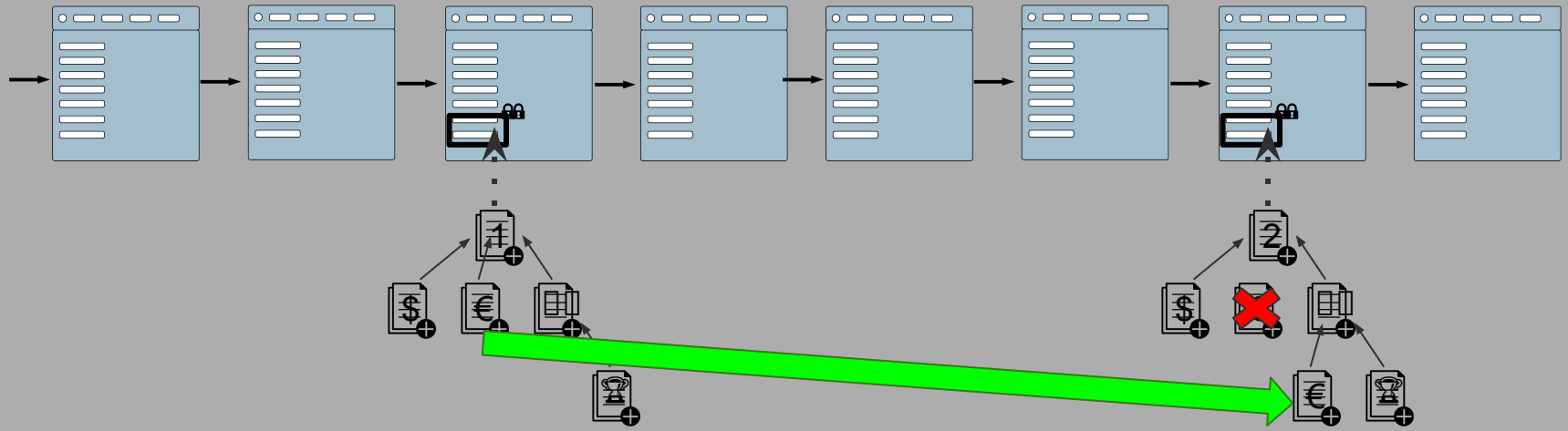
A similar technique, “counterfactual addressing”, allows any of these counterfactual contracts to refer to or depend on each other, even across channels.

Result: a highly generalised and abstract solution!

Under the hood:

To keep all these different contracts in order, we use a **hierarchical** and **versioned** “tree” of counterfactual subchannels.

Each contract requires certain states within all parent contracts in order to become finalised. To delete or move a contract, we exchange pre-signed state update transactions which would call one of its parents to make the old contract invalid if another party tried to publish it. This allows for periodic cleanup of the whole state tree just by incrementing the root version number.



To learn more, visit

github.com/ledgerlabs/state-channels/wiki

Questions?

To learn more, visit
github.com/ledgerlabs/state-channels/wiki

LedgerLabs

Thanks!

Jeff Coleman
Head of Technology
ledgerlabs.com
info@ledgerlabs.com