State Channels: making your application practical

LedgerLabs

Problem:

In the age of Ethereum, your application now has access to the world’s most secure, decentralised, and trust free mobile phone from 1999

How do you deliver an amazing user experience?

Answer:

State channels!

Disclaimer: state channels may not be right for you; results not guaranteed. Do not use state channels as a substitute for engineering or optimisation. Please consult with your resident cryptoeconomist to see if state channels are an appropriate solution for your problem. State channels cannot take measures over public participation or resolve questions of data availability. State channels do not solve the halting problem or provide infinite computing power. Do not use state channels if your question is broken. State channels may be habit forming. If learning more about state channels results in obsessive behaviour...welcome!

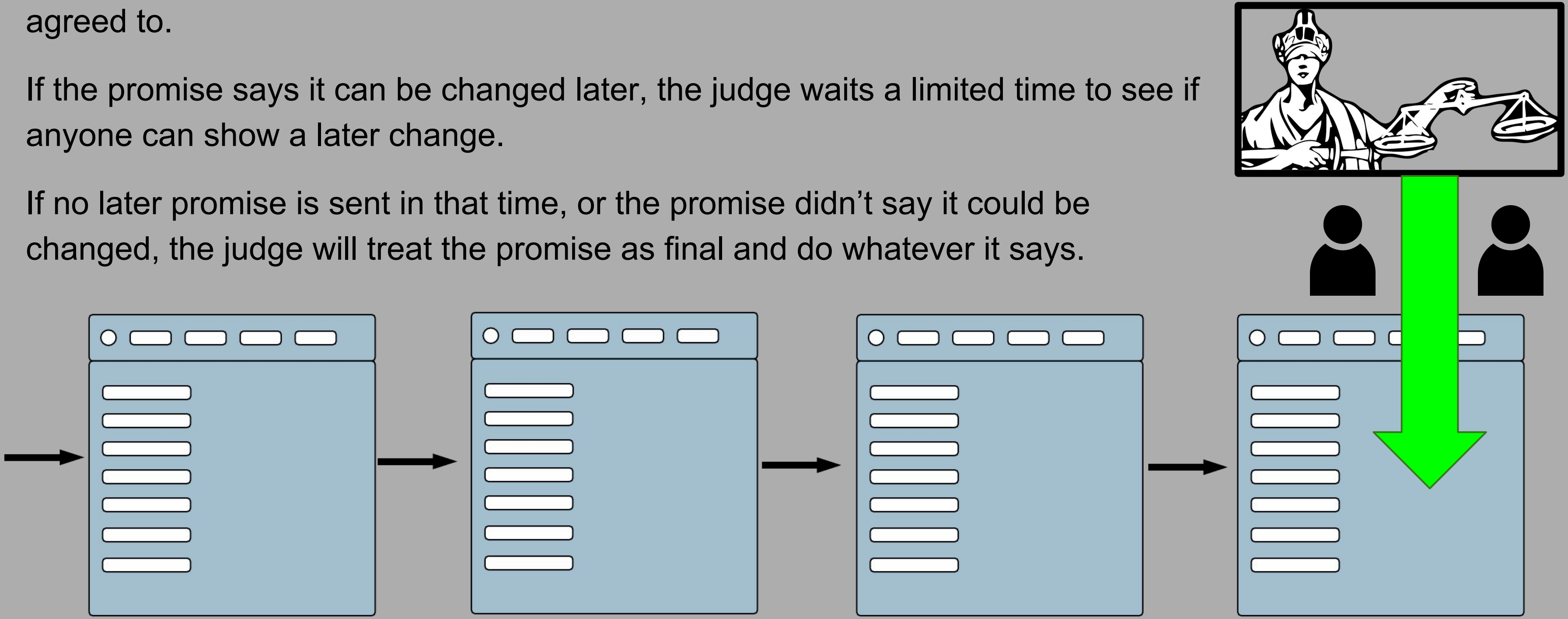
Reminder:
How to make a state channel

Step 1: find someone else you want to use Ethereum with, and create a special smart contract

This contract is like a **simple judge** that accepts **promises** both parties have agreed to.

If the promise says it can be changed later, the judge waits a limited time to see if anyone can show a later change.

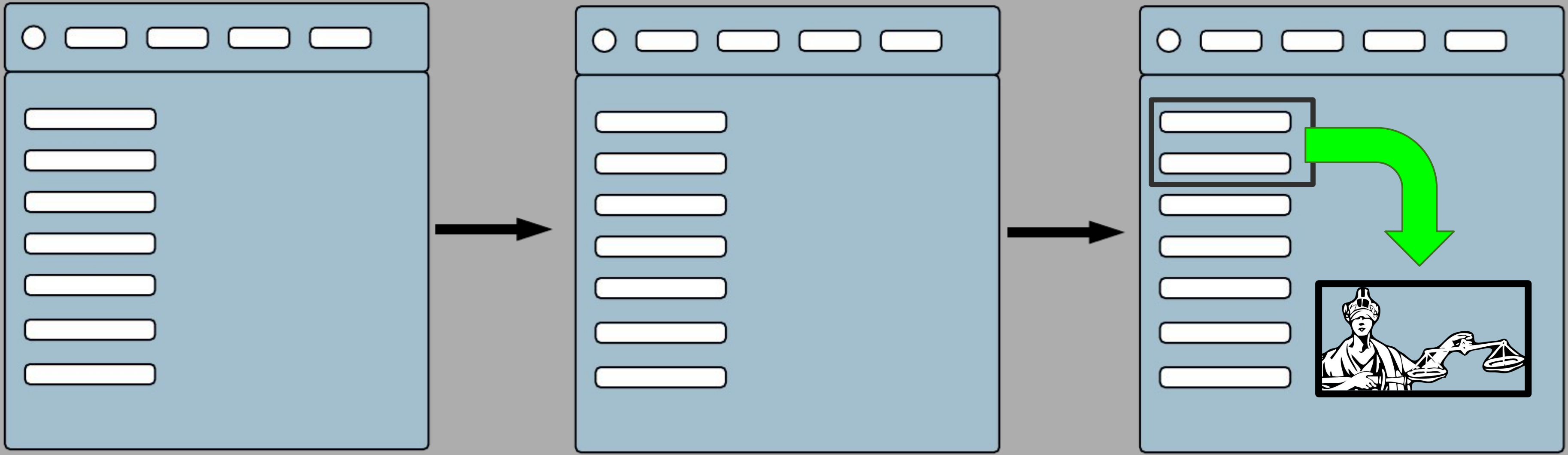
If no later promise is sent in that time, or the promise didn't say it could be changed, the judge will treat the promise as final and do whatever it says.



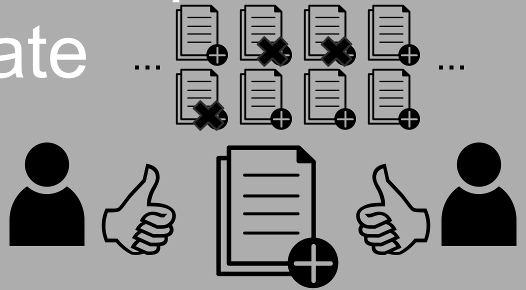
Step 2: take some state and put the smart contract in control of it

Before you hand over control, have the other person agree to promise **you'll get it back**.

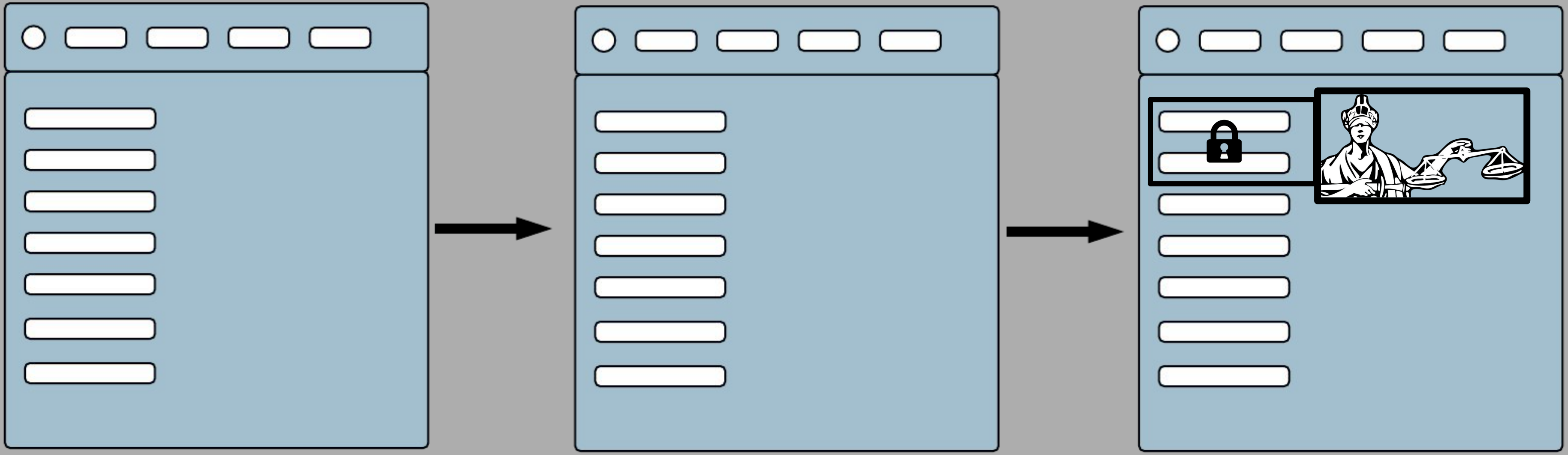
Because the judge enforces the promises, and nothing will happen without your permission, **having the judge be in charge is now just as safe as being in charge yourself**.



Step 3: have the two parties agree on new promises every time they want to change the state

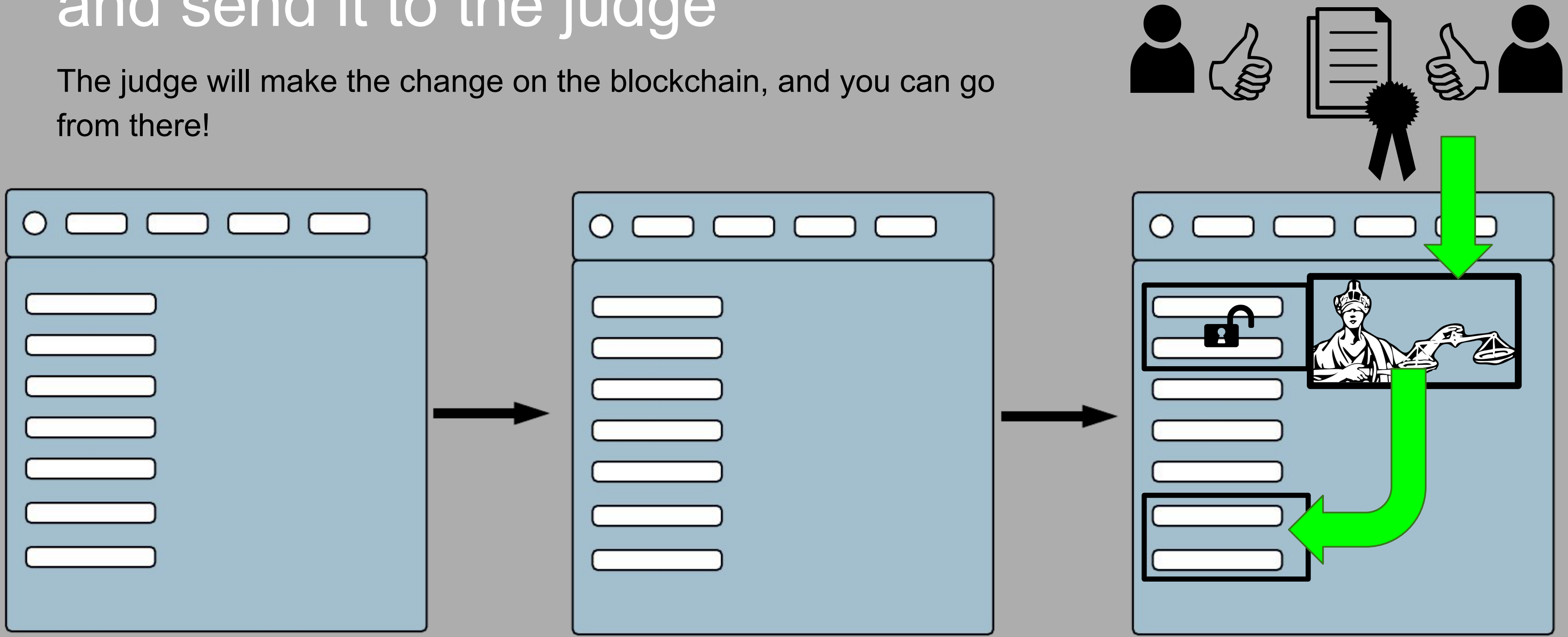


Because the judge will always give you time to tell your side of the story, you can consider a received promise **instantly final**, even if you haven't send it to the judge yet. Instead, they can stay **off of the blockchain**, and be updated many times with **no transaction fees**.



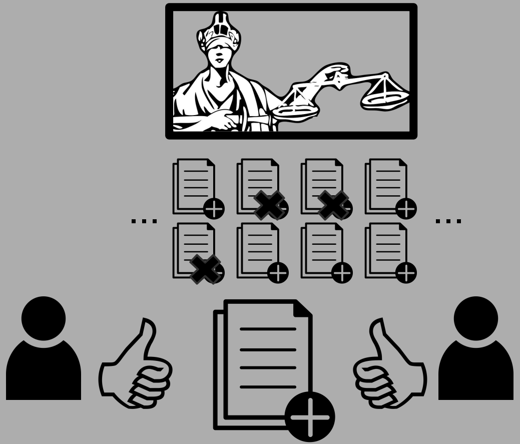
All done! When you want to use some state outside of the state channel, just agree on a final promise and send it to the judge

The judge will make the change on the blockchain, and you can go from there!

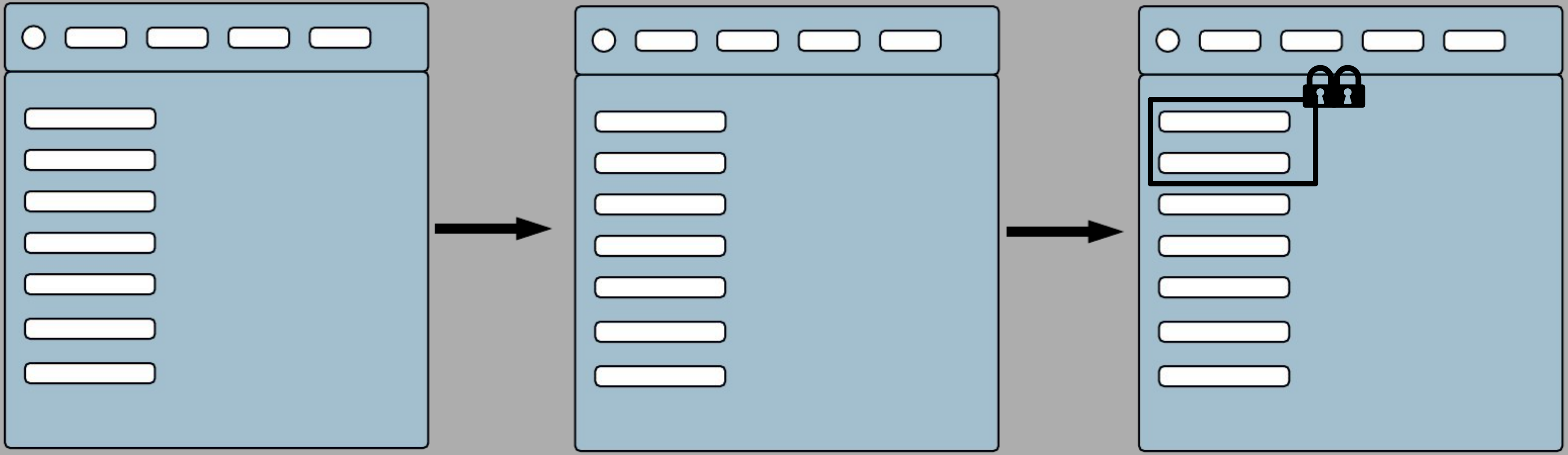


How to make a *better* state channel

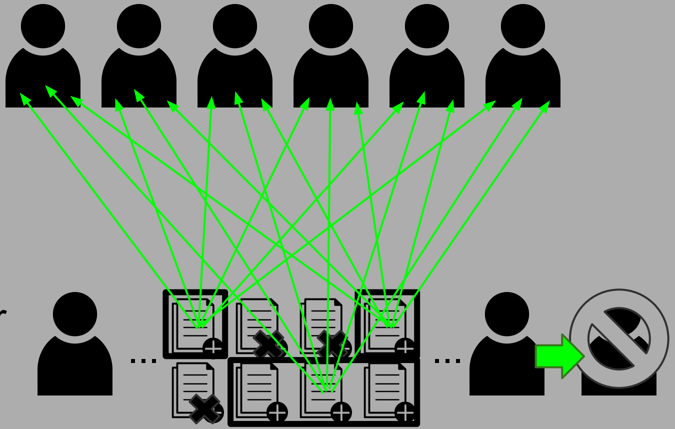
First, we can move the “judge” part out of the blockchain and into the promises, leaving just a simple multi-sig on the chain



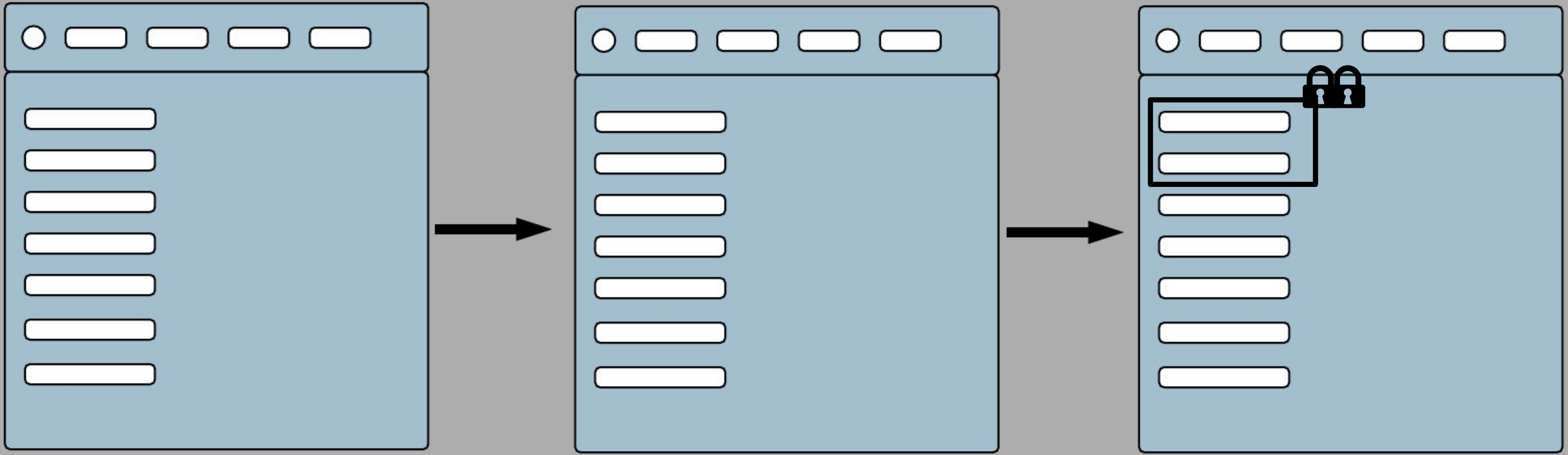
With a little bit of game theory, final promises can be regular transactions now!



Second, users who tend to go offline can just leave the latest promises with hired “bystanders”

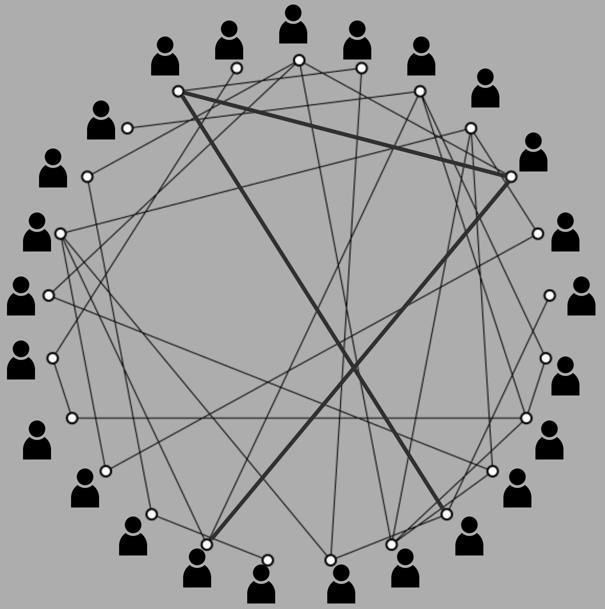


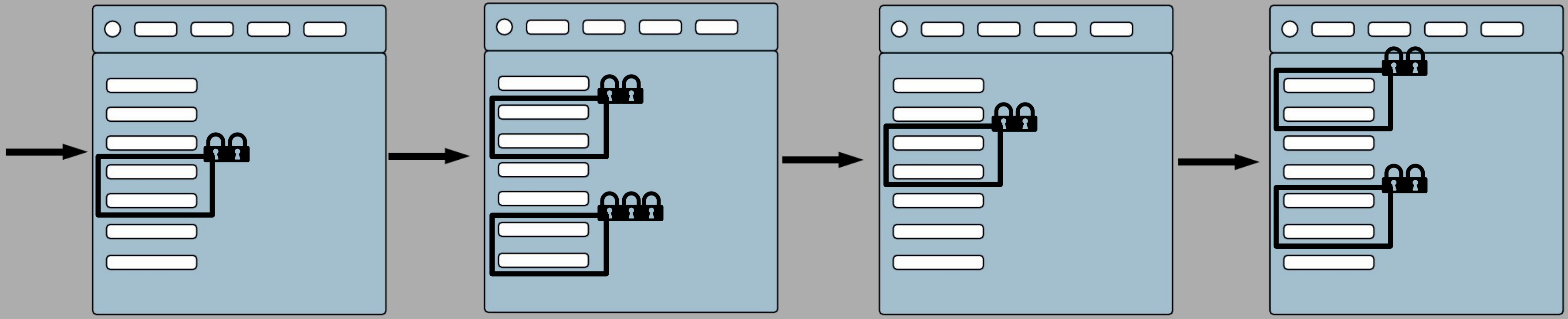
State channeled agreements with many different bystanders can offer to pay a reward if your channel partner tries to submit an old promise and someone corrects it with the latest.



Third, build channels into giant networks (like Raiden!) so you can use them with lots of people

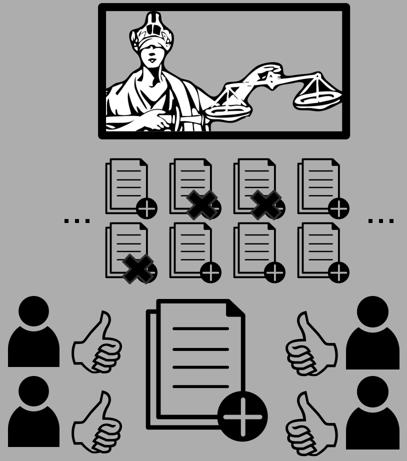
Now, you don't need to already have a dedicated channel with someone in order to use Ethereum with them. Instead, you can **combine multiple existing channels together** to reach them with a few short hops. Participants in the middle only need to help when you start or stop using state, not with every update.

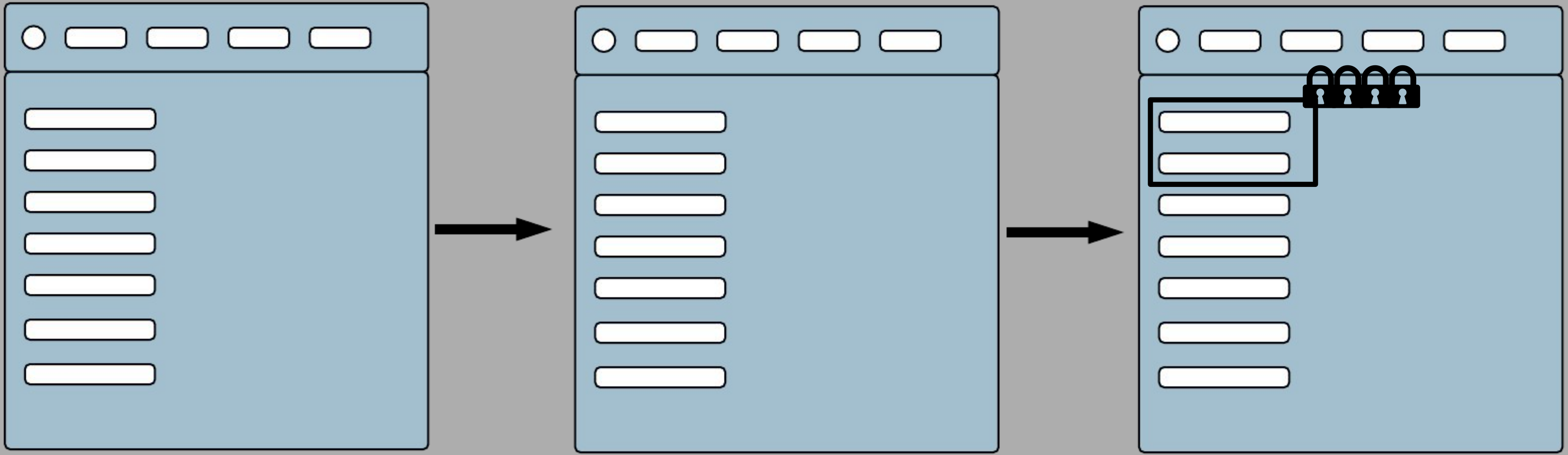




Sometimes it also makes sense to just put more people into one channel

Channels with more participants use less resources than connecting the same number of parties across multiple channels, but the likelihood of having to publish to chain because someone is unavailable goes up as you continue to add more participants.





Concrete example

This is Etherdice.io:

- The entropy generator submits their hash(es) into the smart contract.
- Users submit bets.
- Bets stop being submitted.
- Once bets have confirmed deeply enough to be safe (depends on what size of bets you want to support), the entropy generator submits the secrets to the contract.
- Users claim their rewards.

Result: multiple on chain transactions, plenty of transaction fees, minimum practical bets, no instant gratification for users, not possible to support millions of users.

This is Etherdice.io on state channels:

- The entropy generator hashes their secret.
- Users connect to the gambling operator via a channel network (i.e. Raiden) and contract for the bet offer in a subchannel, including the hash of the secret.
- User chooses their bet (and associated entropy), then transmits it to the operator.
- Gambling operator releases the secret to the user.
- Balances are updated, and either the user plays again or the channel is closed.

Result: zero on chain transactions, minute or nonexistent transaction fees (costs can be absorbed cheaply by the house), very small bets are practical, users can bet and get a result at the speed of network latency, millions of users can be supported with only a small level of server investment by the gambling service operator.

Questions?

To learn more, visit

github.com/ledgerlabs/state-channels

LedgerLabs

Thanks!

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