

CS251 Fall 2025
(cs251.stanford.edu)



Final Topics:

Bridging, Post-quantum Blockchains, and DAOs

Dan Boneh

Invited talk final lecture. Final exam next Tuesday (on Gradescope)

Many more topics to cover ...

- Blockchain interoperability (bridging)
- Governance: How to decide on updates to, say, on Uniswap
- Post-quantum blockchains: security against quantum computers
- Account abstraction
- Re-staking: using your stake to secure multiple applications
- Data Availability Sampling (DAS) to handle large blocks
- Recoverable tokens: how to protect your tokens from theft (ERC-20R)
- **Many more cryptography techniques** (see slides at end)

More topics ...

Where can I learn more?

- **CS255** and **CS355**: Cryptography (Winter and Spring)
- **EE374**: Scaling blockchains with fast consensus
- **MS&E247**: Economics and DeFi
- **Science of blockchain conference** ([SBC](#)): July 2026
- **Stanford blockchain club**

A career in blockchains? Where to start? [[link](#)]

Interoperability between blockchain

How to bridge chains

Many L1 blockchains

Bitcoin: Bitcoin scripting language

Ethereum: EVM.

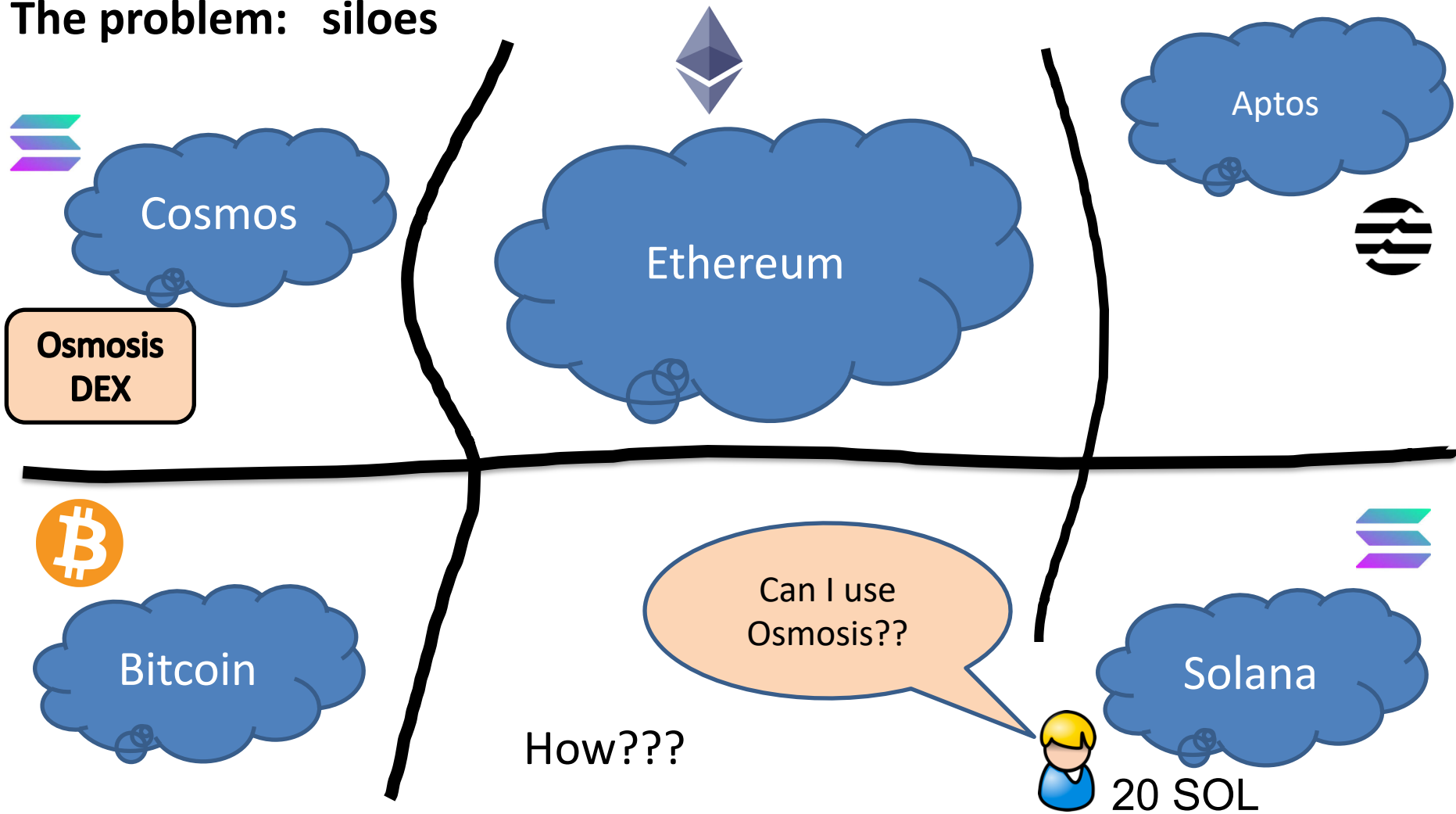
EVM compatible blockchains: **Celo, Avalanche, BSC, ...**

- Higher Tx rate $\Rightarrow$ lower Tx fees
- EVM compatibility $\Rightarrow$ easy project migration and user support

Other fast non-EVM blockchains: **Solana, Aptos, Sui, Flow, ...**

- Higher Tx rate $\Rightarrow$ lower Tx fees

The problem: siloes



Interoperability

Interoperability:

- User owns funds or assets on one blockchain system
Goal: enable user to move assets to another chain

Composability:

- Enable a DAPP on one chain to call a DAPP on another

Both are easy if the entire world used Ethereum

- In reality: many blockchain systems that need to interoperate
- The solution: **bridges**

A first example: BTC in Ethereum

How to move BTC to Ethereum ?? Goal: enable BTC in DeFi.

⇒ need new ERC20 on Ethereum pegged to BTC

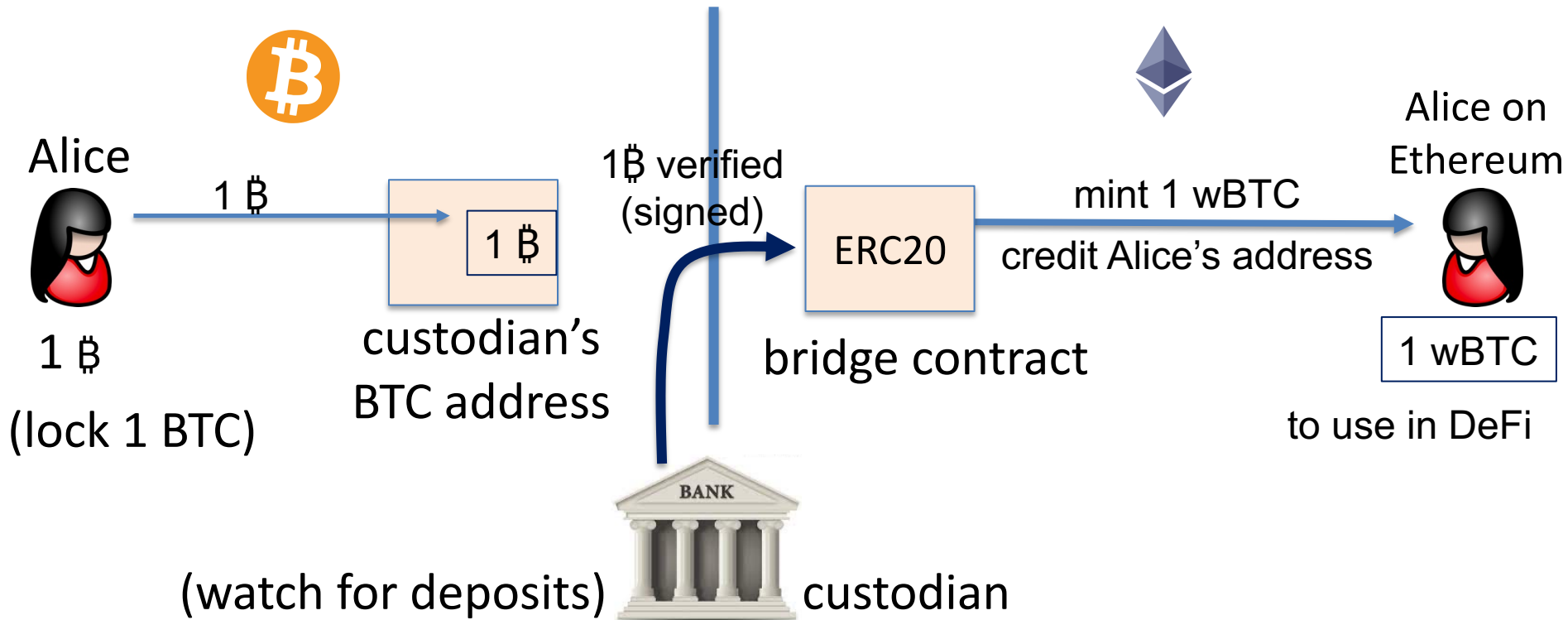
(e.g., use it for providing liquidity in DeFi projects)

The solution: **wrapped coins**

- Asset X on one chain appear as wrapped-X on another chain
- For BTC: several solutions (e.g., wBTC, cbBTC, tBTC, ...)

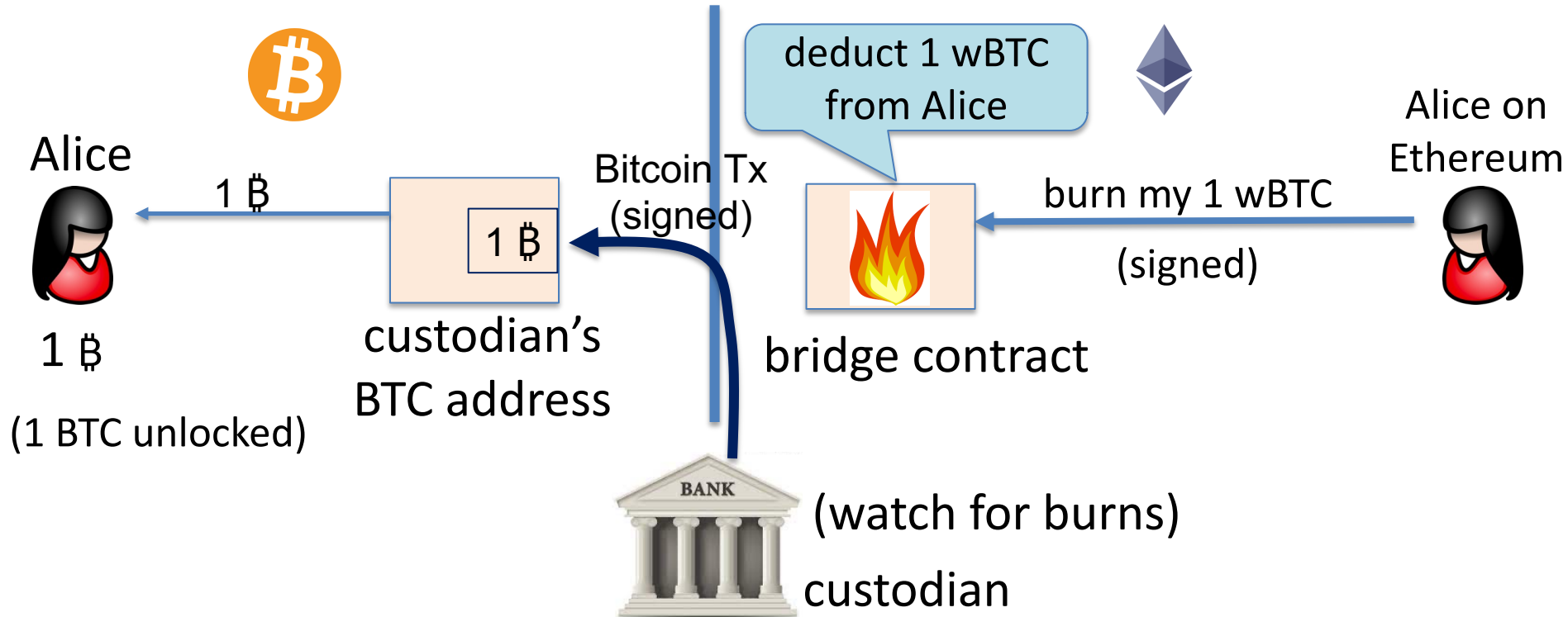
wBTC and cbBTC: a lock-and-mint bridge

Let's start with wBTC: **moving 1 BTC to Ethereum**



Alice wants her 1 BTC back

Moving 1 wBTC back to the Bitcoin network:



wBTC

Example BTC → Ethereum:

Nov 26 2021 - 07:36	FUNDS SENT TO CUSTODIAN	(Bitcoin Tx: ≈4,000 BTC)
		<code>c605b4f2f0948e7deae0c5d7c27b3256b97120be760e2b81136eb95c819570f6</code>
Nov 26 2021 - 09:50	MINT COMPLETED BY CUSTODIAN	(Ethereum Tx:)
		<code>0x70475eca8be89b67143f1b52df013fc1df7d254e836c836c8f368fc516aca76b</code>

Why two hours? ... make sure no Bitcoin re-org

The problem: trusted custodian

Can we do better?

tBTC (v2): no single point of trust

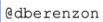
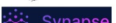
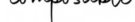
A new bridge address created every two weeks

- Managed by 100 custodians selected from a pool of signers
- A threshold of custodians must sign minting and burning transactions
- Bridging operations take several hours

Bridging smart chains (with Dapp support)

A very active area:

- Many super interesting ideas

Asset-specific	Chain-specific	Application-specific	Generalized
      	             	          	             

<https://medium.com/1kxnetwork/blockchain-bridges-5db6afac44f8>

Two types of bridges

Type 1: a lock-and-mint bridge

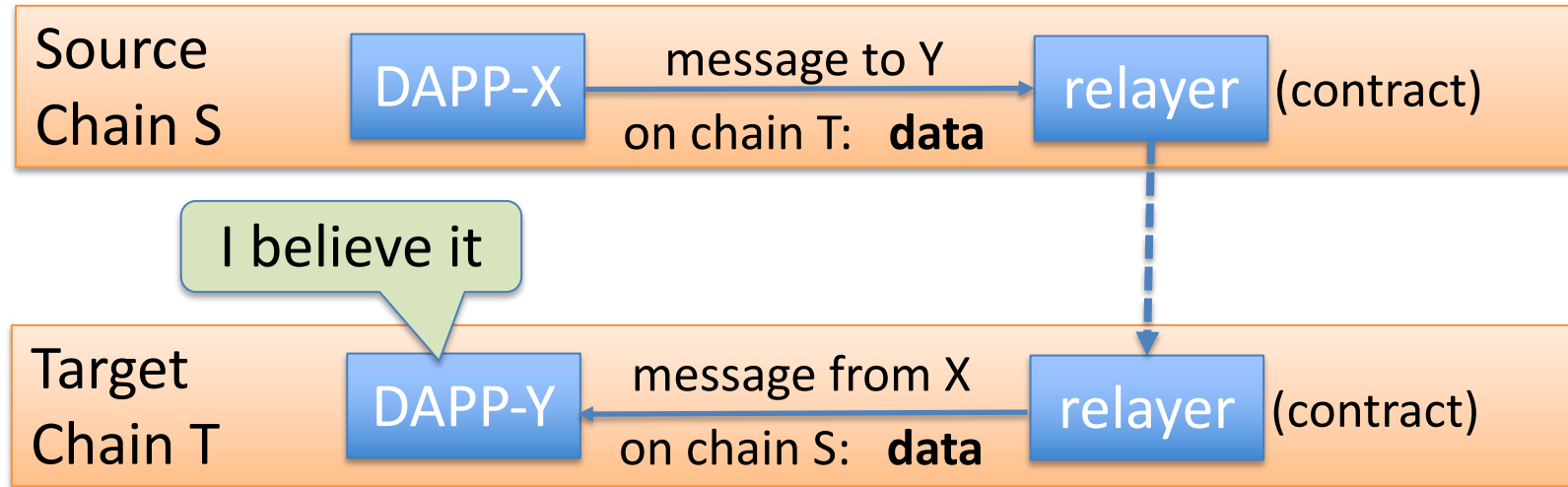
- SRC → DEST: user locks funds on SRC side,
wrapped tokens are minted on the DEST side
- DEST → SRC: funds are burned on the DEST side,
and released from lock on the SRC Side

Type 2: a liquidity pool bridge

- Liquidity providers provide liquidity on both sides
- SRC → DEST: user sends funds on SRC side,
equivalent amount released from pool on DEST side

Bridging smart chains (with Dapp support)

Step 1 (hard): a secure cross-chain messaging system



Step 2 (easier): build a bridge using messaging system

Bridging smart chains (with Dapp support)

Step 1 (hard): a secure cross-chain messaging system



Step 2 (easier): build a bridge using messaging system

- DAPP-X → DAPP-Y: “I received 3 CELO, ok to mint 3 wCELO”
- DAPP-Y → DAPP-X: “I burned 3 wCELO, ok to release 3 CELO”

If messaging system is secure, no one can steal locked funds at S

Primarily two types of messaging systems

(1) **Externally verified:** external parties verify message on chain S



RelayerT dispatches only if all trustees signed

⇒ if DAPP-Y trusts trustees, it knows DAPP-X sent message

Primarily two types of messaging systems

(1) **Externally verified:** external parties verify message on chain S



What if trustees sign and post a fake message to relayerT?

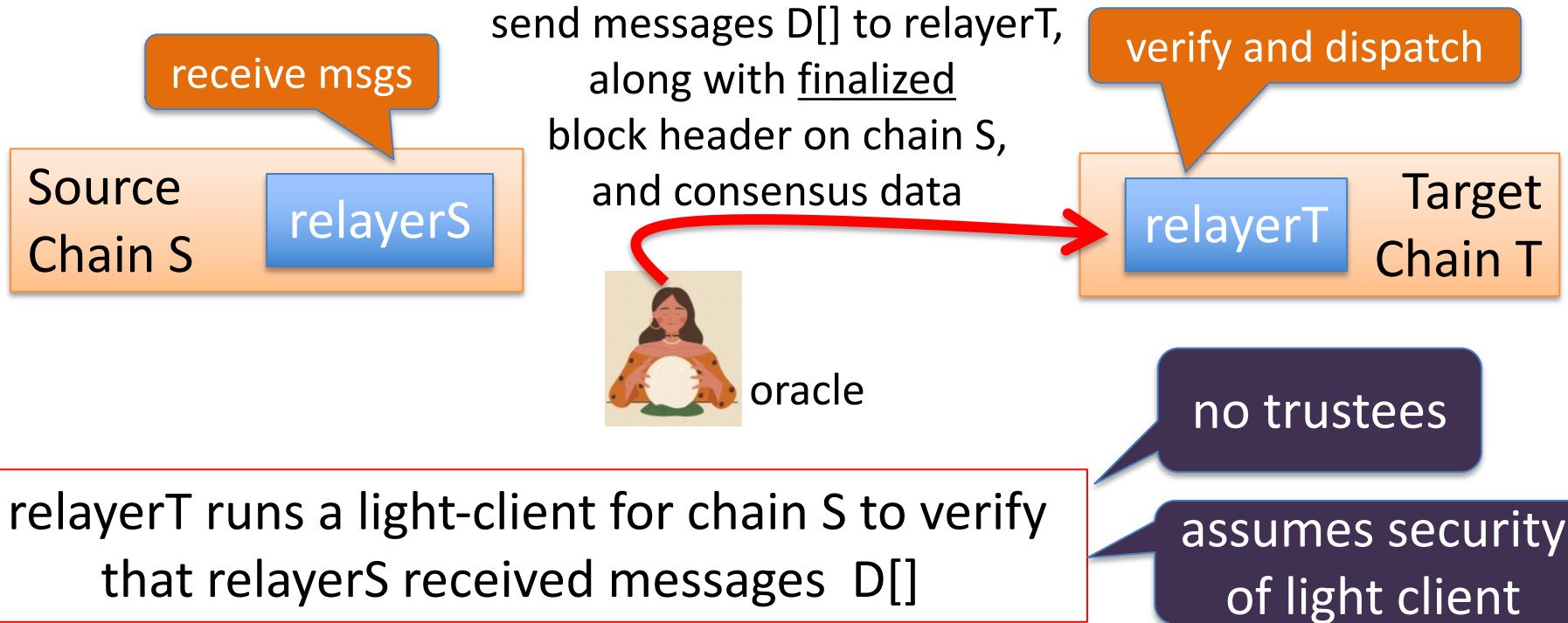
- anyone can send trustee's signature to relayerS $\Rightarrow$ trustee slashed on S

Activity

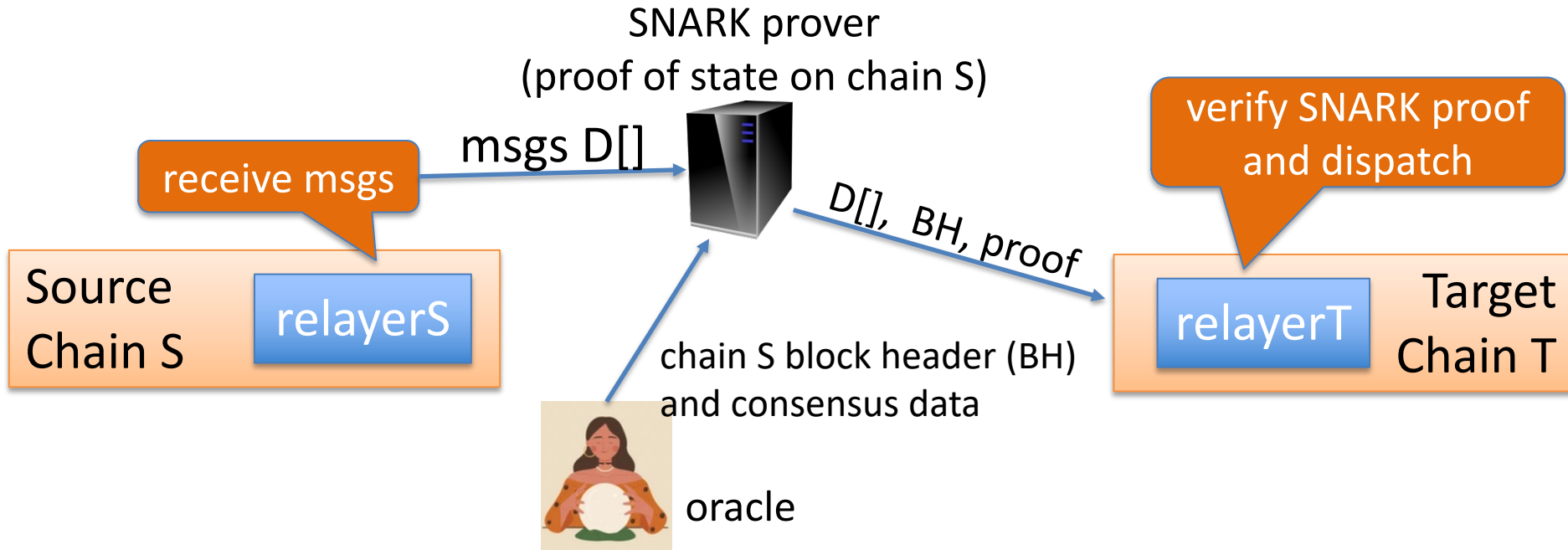
##	NAME	VALUE EXPORTED	VALUE IMPORTED
01	 Ethereum	\$14,354,678,228	\$3,802,063,122
02	 Bitcoin	\$4,738,234,009	\$0

Primarily two types of messaging systems

(2) **On-chain verified:** chain T verifies block header of chain S

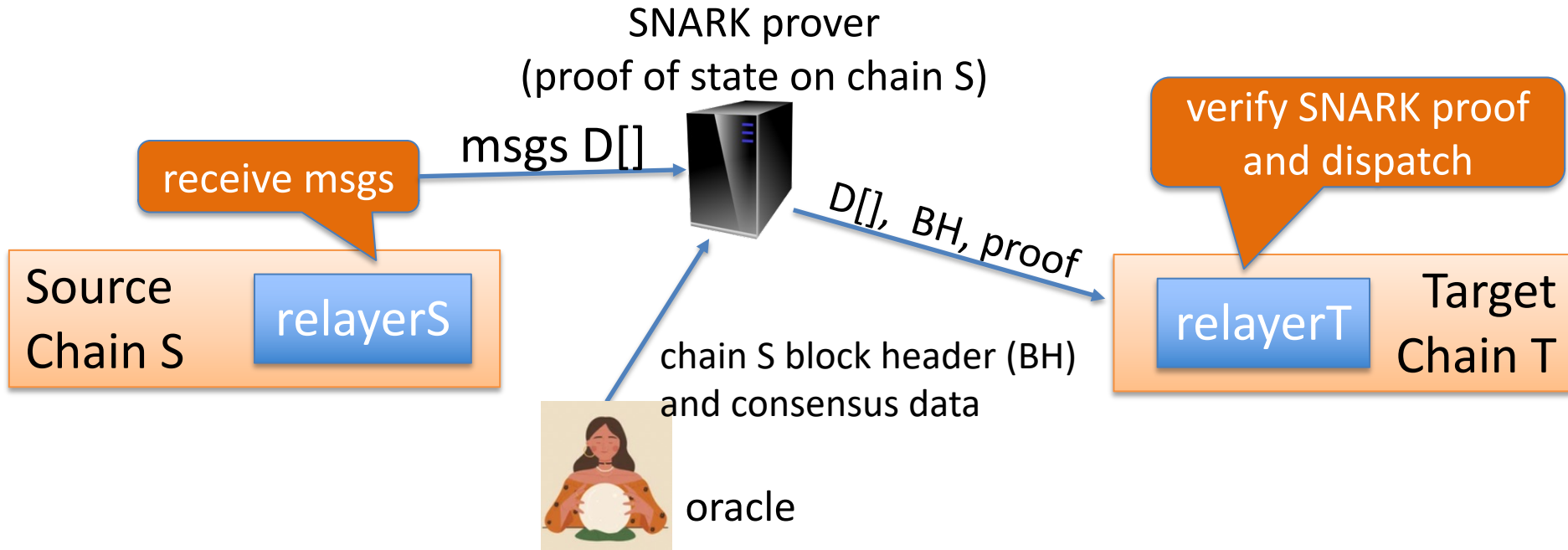


Primarily two types of messaging systems



Problem: high gas costs on chain T to verify state of source chain S.
Solution: zkBridge: use SNARK to reduce work for relayerT

Primarily two types of messaging systems



This is called a “proof-of-consensus”

Bridging: the future vision

User can hold assets on any chain

- Assets move cheaply and quickly from chain to chain
- A project's liquidity is available on all chains
- Users and projects choose the chain that is best suited for their application and asset type

We are not there yet ...

Post-Quantum Blockchains

Recall: signature schemes

$\text{KeyGen}() \rightarrow (pk, sk)$

sign a message m with sk , verify a signature with public pk

What signatures schemes are used in practice:

- Bitcoin uses **ECDSA** and **Schnorr** to sign Tx
... but UTXO may only reveal $\text{Hash}(pk)$
- Ethereum uses **ECDSA** for native ETH, and **BLS** for PoS consensus
- Solana uses **EdDSA** (a variant of Schnorr)

The quantum threat

Shor's algorithm (1994): for all these signature schemes,
given pk anyone can compute sk in a few days,
using a large enough **quantum computer**

What is large enough? Quite large (compared to what we have today)

When is Q-day?  (lots of guesses on $\mathbb{X}$)

(Is a quantum computer good for anything else?)

What is at stake?

Total BTC at Risk of Quantum Attack:

6,743,168 BTC

Total USD at Risk of Quantum Attack:

\$579,486,954,099

... also **Proof-of-Stake consensus**, if validator signing keys are exposed

The risks ...

- What if a rogue nation is the first to develop a large QC?
- What if whoever builds a QC steals the Bitcoin and dumps it all?
⇒ will collapse the price of Bitcoin
- The positive: people who lost their keys, can get their coins back

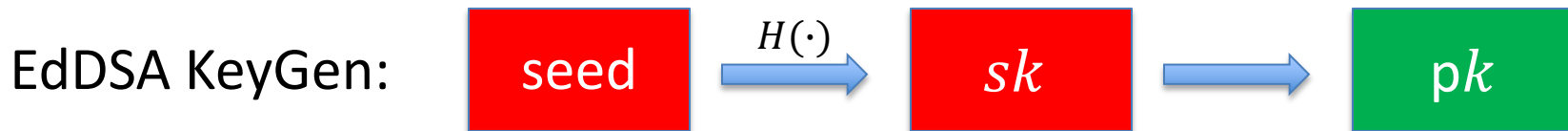


**Man Fails to Buy Landfill With His Lost \$923M
Bitcoin—Here's His New Plan**

(8000 BTC)

What to do?

The “lucky” EdDSA trick:



Idea: once a QC is imminent, blockchain rejects EdDSA sigs by sk

Instead: to sign a Tx, user provides a SNARK proof that it knows a hash preimage of sk (such a proof can be made into a signature)

What to do? The NIST sig. standards

ML-DSA: a signature scheme that remains secure even if adversary has a large QC (as far as we know)

	sig size	<i>pk</i> size
ML-DSA-44 (128-bit security)	2420 bytes	1312 bytes
ML-DSA-65 (192-bit security)	3309 bytes	1952 bytes
ML-DSA-87 (256-bit security)	4627 bytes	2592 bytes
EdDSA	48 bytes	32 bytes

ML-DSA is much bigger than pre-quantum signatures

- An open problem: post-quantum sig. and *pk* as short as EdDSA

Exciting times ahead

Moving blockchains to a post-quantum signature scheme

⇒ Everyone (the entire world) will need to update their accounts:

- **On Bitcoin:** move to a new address (BIP-360)
- **On Ethereum:** move all funds to an on-chain wallet (AA)
 - Can be done with a single transaction using EIP-7702
 - Requires Ethereum to deprecate EOA signing after a flag day
- **On Aptos:** update the authorization key for your account
- ...

What happens to people who don't update?

Option 1: nothing, until a large Quantum Computer is built

Option 2: introduce a flag date, after which all pre-quantum addresses are canceled

Which option will Bitcoin take?

- Probably fork: one fork with option (1) and another with (2)
- Let the market choose which fork to assign value to
... not hard to guess which fork will be chosen

Blockchain Governance (DAOs)

Decentralized Orgs (DAO)

What is a DAO?

- A Dapp deployed on-chain at a specific address
- Anyone (globally) can send funds to DAO treasury
- Anyone can submit a proposal to DAO
 - ⇒ participants vote
 - ⇒ approved → proposal executes



snapshot.org

Examples of DAOs

- **Collector DAOs:** PleasrDAO, flamingoDAO, ConstitutionDAO, ...
(see art collection at <https://gallery.so/pleasrdao>)

PleasrDAO: 103 members.

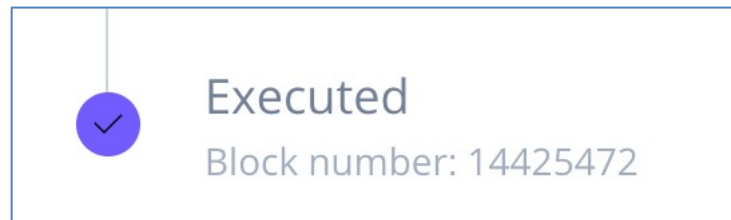
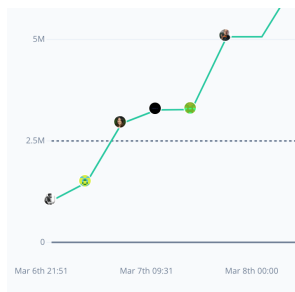
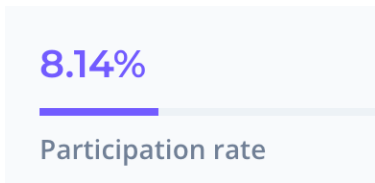
Manages a treasury, has full time employees.

Deliberations over what to acquire over telegram.

Examples of DAOs

- Collector DAOs: PleasrDAO, flamingoDAO, ConstitutionDAO, ...
- **Grants DAOs:** gitcoin (83K members), ...

Proposal ID 21: This proposal looks to ratify the allocation of 30,000 GTC from the Community Treasury to the MMM workstream.



(tally.com)

Examples of DAOs

- Collector DAOs: PleasrDAO, flamingoDAO, ConstitutionDAO, ...
- Grants DAOs: gitcoin, ...
- **Protocol DAOs:** manages operation of a specific protocol
Uniswap DAO (74K), Compound DAO (8K), ...
- **Social DAOs:** FWB, ...
- **Investment DAOs:** many

Many DAO governance experiments

Who can vote? How to vote? What voting mechanism?

Lightspeed Democracy: What web3 organizations can learn from the history of governance

by Andrew Hall and Porter Smith

June 29, 2022

DAOs: a platform for experimenting with governance mechanisms

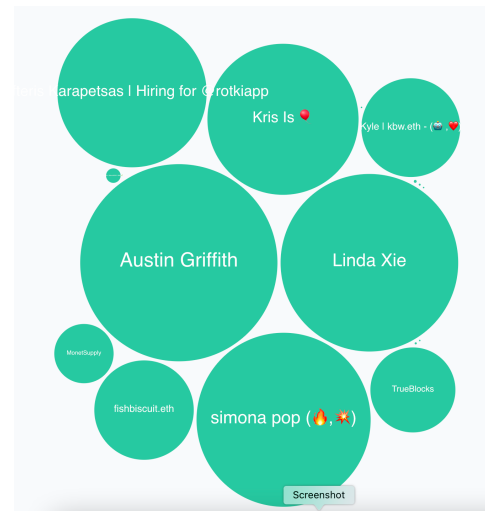
Governance method

One token one vote: (most common)

- Members receive tokens based on their contribution
- Everyone can vote

In reality: low participation rate (8%)

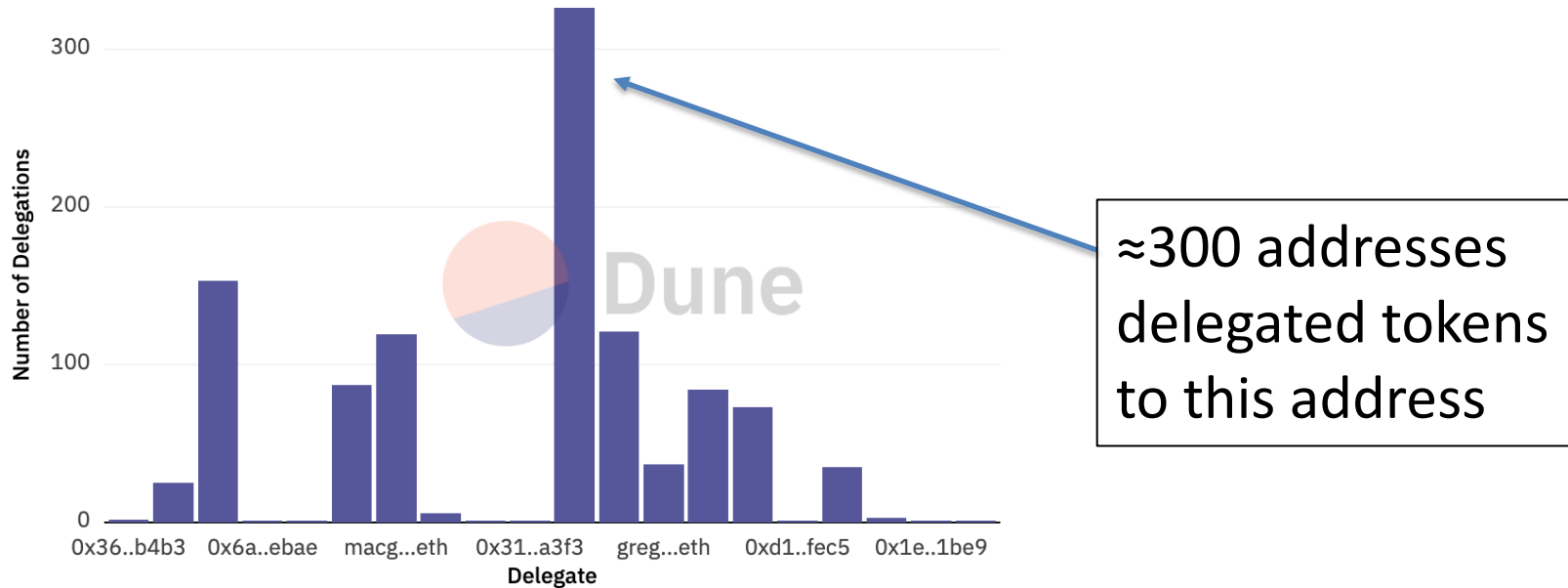
⇒ delegation



proposal 21 (tally.com)

Delegation example: element

Number of Delegations per Delegate (Sorted by Voting Power)



Governor Bravo

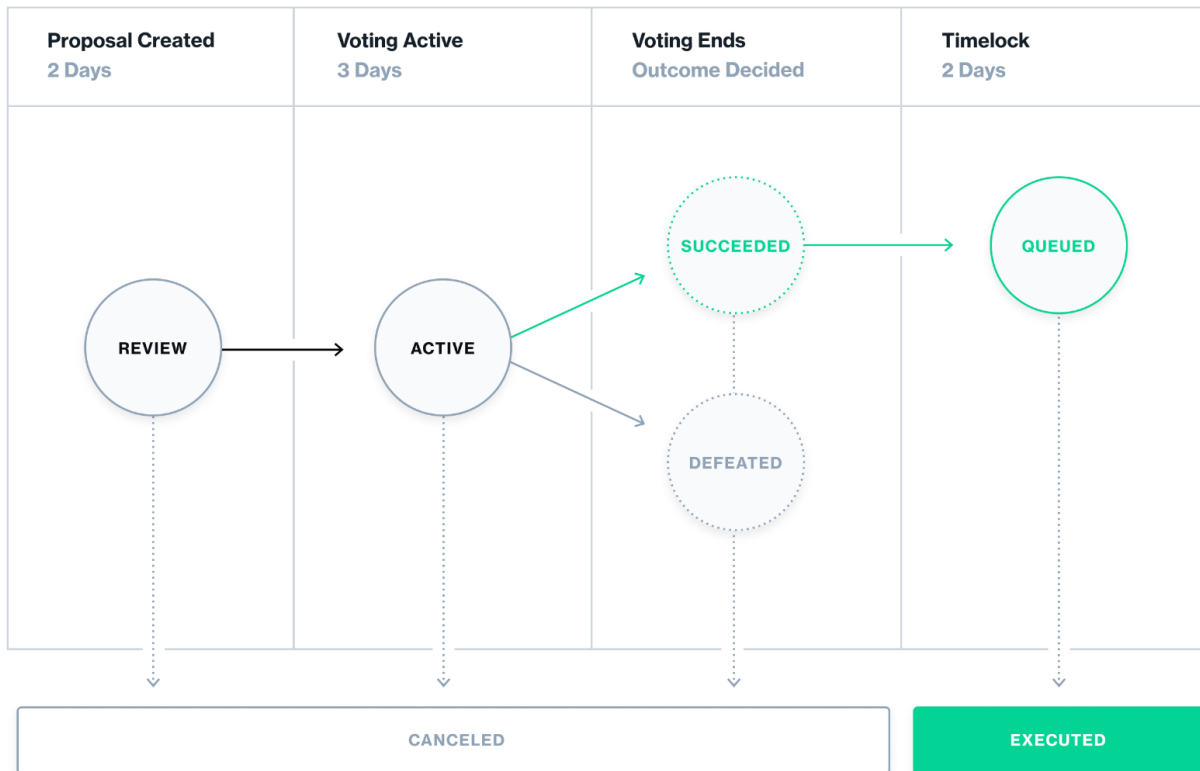
Governor alpha (Jan. 2020) updated to [Bravo](#) (March 2021):

- A collection of smart contracts used for governance
- Forked and used by many other protocols

Other alternatives:

- [OpenZeppelin Governor](#) protocol

The life cycle of a proposal



Governor Bravo

Implements an ERC-20 governance token. Let's call it GOV.

- An address that owns GOV tokens can vote on proposals

The proposal I am voting for
(with all my allotted voting tokens)

0: against
1: in favor
2: abstain

```
function castVote(uint proposalID, uint8 support)
```

(cannot vote partially)

Governor Bravo

Implements an ERC-20 governance token. Let's call it GOV.

... or delegate its GOV tokens to another address

`function delegate(address delegatee)`



delegate all my
voting tokens

can re-delegate at any time by calling function again

An address can only delegate all its tokens to one address at a time

... but I can hold my GOV tokens at multiple addresses

Creating a proposal

function **propose**(

The function to call on governor contract

address[] targets,
uint[] values,
string[] signatures,
bytes[] calldatas,

If proposal passes,
how to execute it:
what functions to call (actions)
and with what arguments

string description) // proposal description
(human readable)

returns (uint)

The ID of newly created proposal

Global contract parameters

- **quorumVotes:** the minimum # of votes for a proposal to pass
- **proposalThreshold:** min # votes needed to create a proposal
- **votingDelay:** # blocks to wait until voting can begin after a proposal is created (e.g., two days)
- **votingPeriod:** # blocks when voting is open (e.g., three days)

All these parameters can be changed by governance in Bravo

Proposal execution

Once proposal gets enough votes and delays expire:

- anyone can call: `function execute(uint proposalID)`

⇒ prescribed functions in proposal get called

The limits of democracy

Anytime before execution

- “guardian” can call **cancel()** to cancel a proposal

Who is this guardian??

- set at Governor contract creation time
- can abdicate at any time by calling **_abdicate()**

DAO privacy questions

- **Private DAO participation:** keep membership list private
- **Private voting:** keep who voted how on each proposal private
- **Privately delegate** voting rights
- **Private treasury**

... while complying with all relevant laws.

Some DAOs made mistakes and were attacked

- Attack on Beanstalk governance: \$182M
- Attack on Yam Finance governance: thwarted
- Attack on Build Finance governance \$1.5M
- Attack on Mango governance: \$115M
- Steem governance drama

What happened? What can we learn?

(1) Beanstalk

(Apr. 2022)

An Ethereum-based stablecoin **Bean**. Governance token **Stalk**.

The problem:

- an attacker can buy Stalk tokens,
- submit a proposal,
- vote on proposal, and
- have proposal execute

all in a single
transaction

Beanstalk: the attack

Recall: **flashloan** (has many applications)

- a loan that is taken out and repaid back in a single transaction
- No risk to lender $\Rightarrow$ unbounded amount can be borrowed

The attack:

- Attacker took a huge flashloan from Aave, bought lots of Stalk,
- Passed a proposal to pay \$80M to the attacker from the treasury,
- Sold the Stalk, and repaid the flashloan,
- Sent the proceeds to Tornado cash (donated \$250K to Ukraine)

The lesson

The protocol relaunched four months later.

The lesson: governance requires delays

- Require token holders to hold token for a long period of time
- Build a delay between proposal genesis, voting, and execution

(2) Yam Finance

(July 2022)

- A DeFi project running on Ethereum. Governed token **YAM**.

What happened? (within a single day)

- Attacker bought 224739 YAM with borrowed funds
- Attacker submitted a governance proposal
granting it control of project reserves (\$3M USD)
- Voted on proposal with borrowed 224739 YAM, hitting quorum
- Borrowed YAM exchanged for Eth and paid back

What happened next?

Yam Finance team: noticed the proposal shortly after it hit quorum

⇒ retained **cancel** power, and canceled the proposal.

Lesson:

- Illustrates the positive power of a veto
- The problem: power can never be taken away
(unless voluntarily given away)

END OF LECTURE

Next lecture: super cool final guest lecture

Account abstraction

... as designed in ERC-4337

The Goals of Account Abstraction (AA)

Recall: an asset held in an Externally Owned Account (EOA) is completely controlled by the EOA's signing key

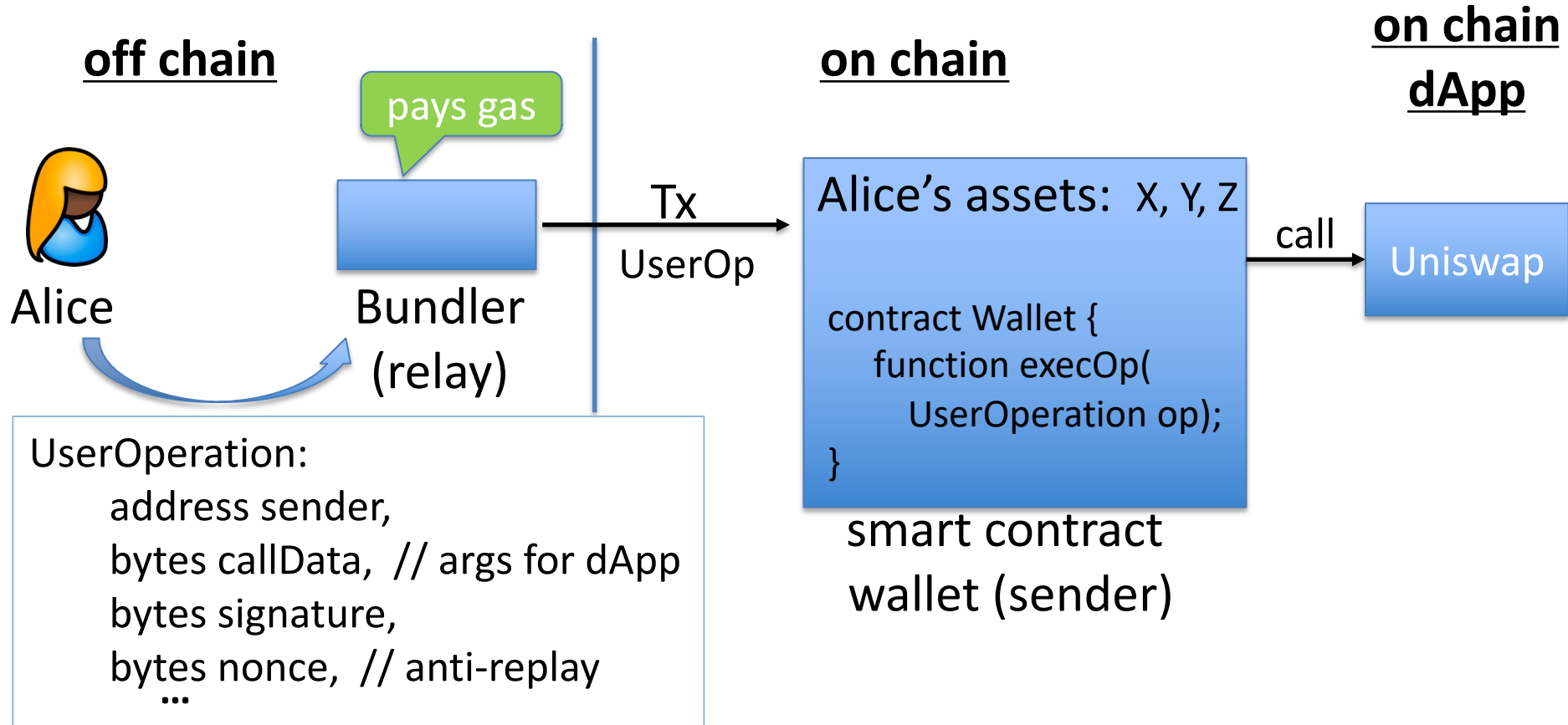
- What if I want two keys to control the asset? Or rate limits? ...

⇒ asset cannot be held by an EOA

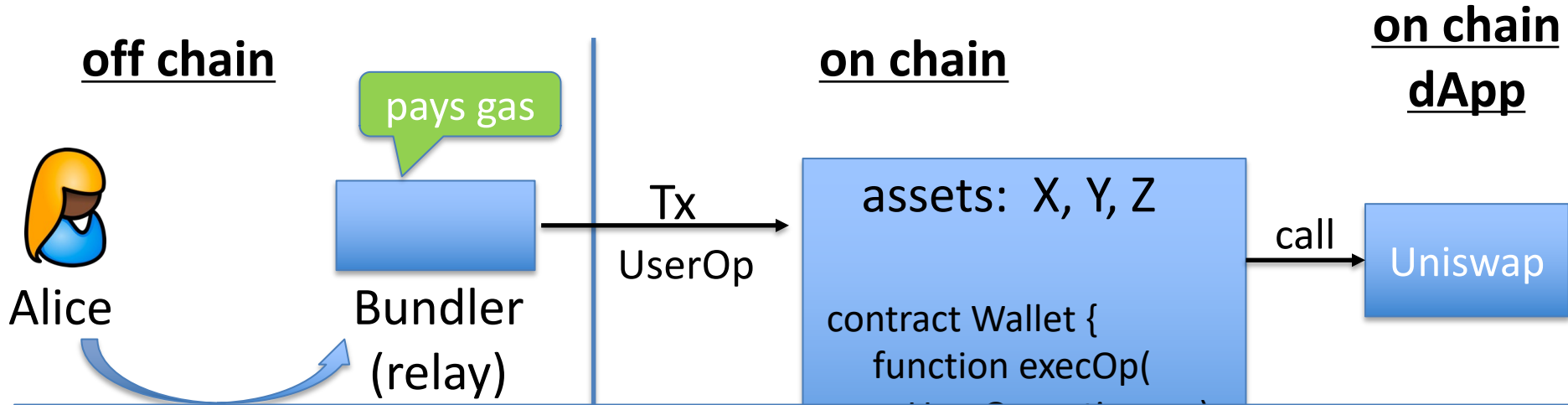
Two primary goals for AA:

- Goal 1: arbitrary complex policies for controlling an asset
- Goal 2: a 3rd party (bundler, paymaster) pays Tx gas for users
 - ⇒ improves user experience – no need to get ETH

Assets are held by a smart contract wallet



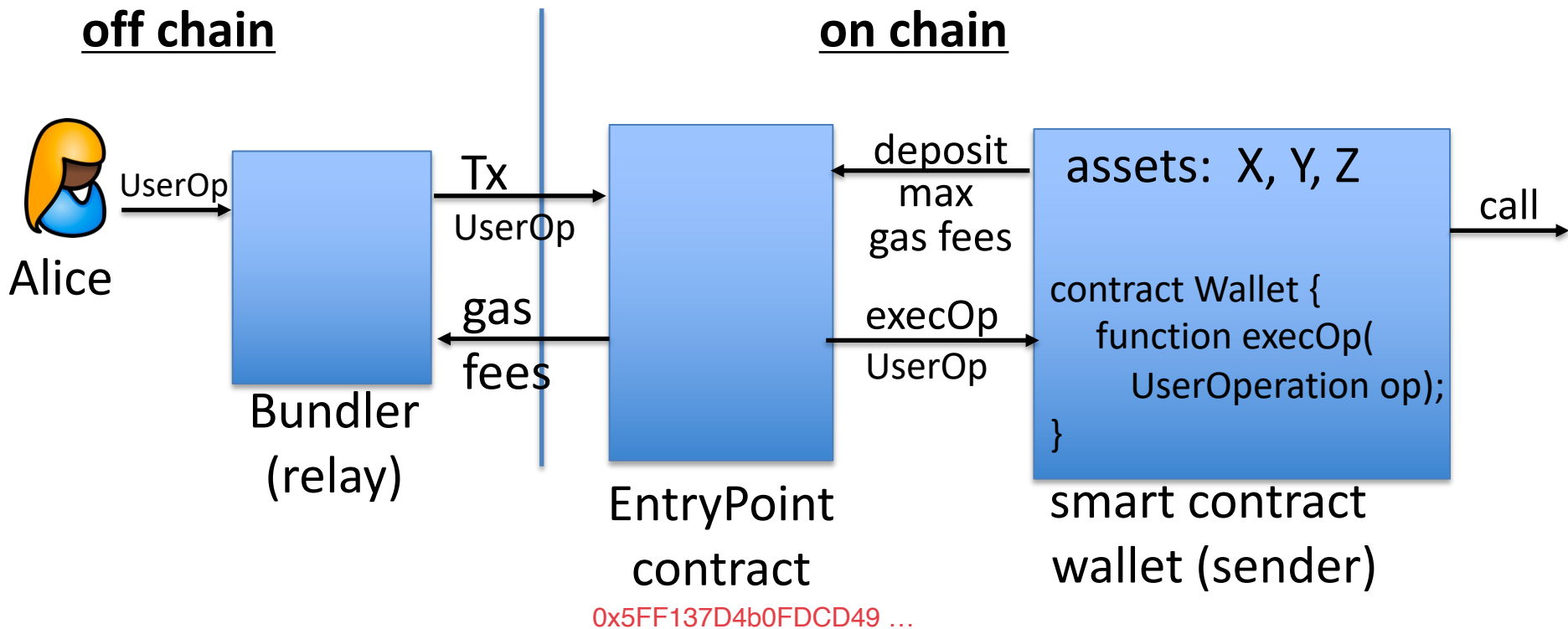
Smart contract wallet



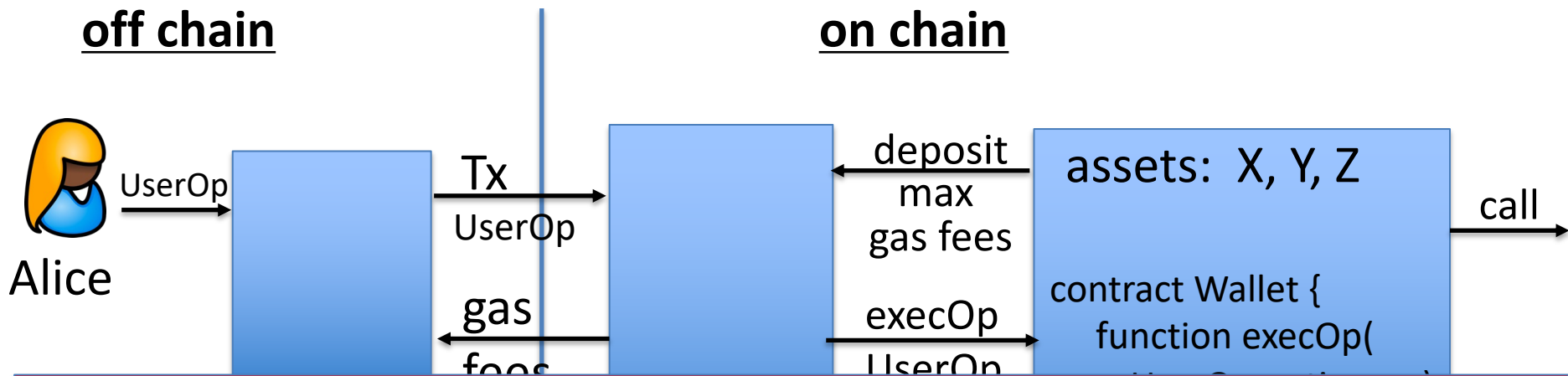
Problems:

- How to reimburse the bundler for the gas it paid?
- Bundler doesn't want to call (potentially malicious) wallet code

The EntryPoint trusted contract



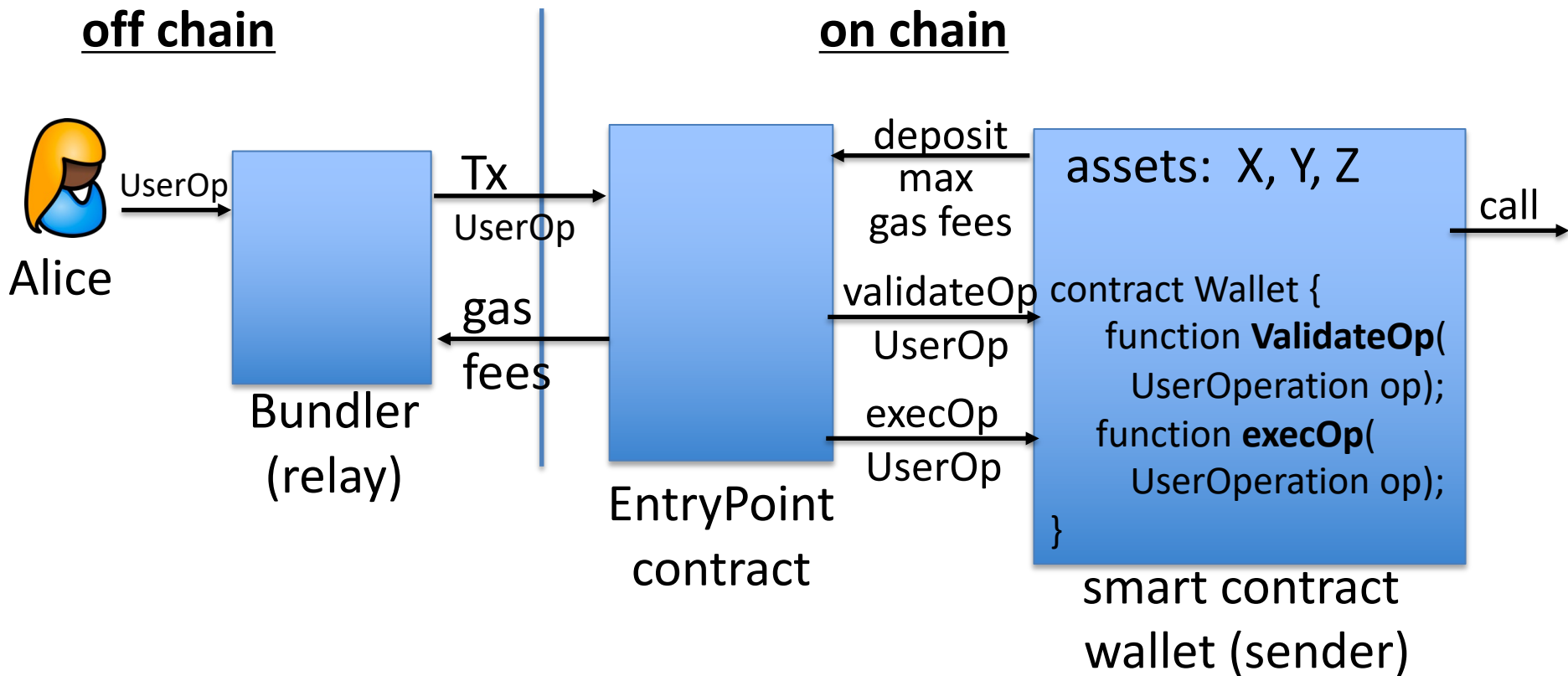
The EntryPoint trusted contract



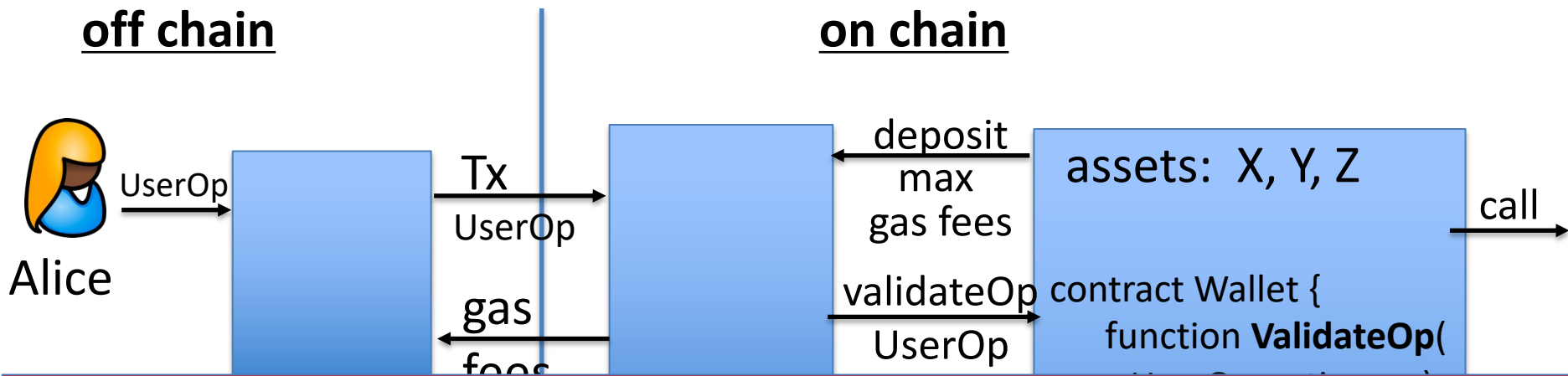
More problems:

- DoS: Anyone can issue invalid UserOp (e.g., with a bad sig) to EntryPoint, causing Wallet to keep paying gas to bundler.

Solution: validateOp



Solution: validateOp



EntryPoint steps: (simplified)

step 1: call **validateOp(UserOp)**

step 2: if fail, stop and Bundler pays the gas

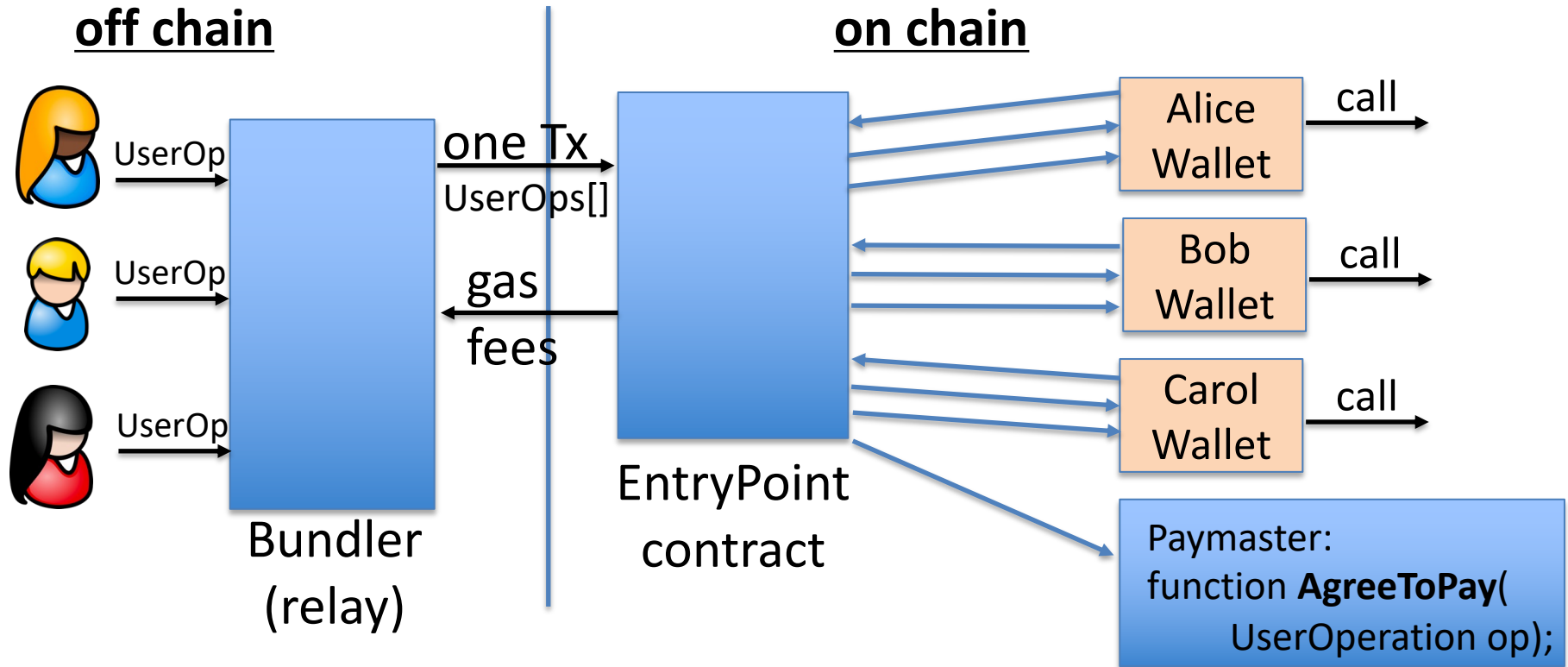
step 3: call **ExecOp(UserOp)**, and Wallet reimburses Bundler for gas

More problems ...

1. Need to ensure that Wallet does not execute op in **validateOp**
⇒ Bundler will be stuck with execution fees
2. Before issuing Tx, Bundler needs to run **validateOp** in its head (a.k.a simulates validateOp) to ensure that UserOp is valid
⇒ need to ensure that simulation and on-chain execution of **validateOp** produce the same output

Solution: **validateOp** is restricted in what EVM opcodes it can use
[**execOp** is not restricted because Wallet is paying gas fees]

Bundling and Paymasters



As usual, Users pay Bundler a fee to be placed in Tx

Very complex ... is this real?

- Bundler's already exists, such as Pimlico
- Simple Tx now consume a lot of gas:
 - Simple ETH transfer from Alice→Bob: 4X more expensive
 - ERC20 transfer from Alice→Bob: 2X more expensive
 - Recall: gas on L2 is not as expensive as gas on Ethereum

Main reason for adoption:

- Can be deployed now with no changes to dApps !!

An example

ERC-4337 Entry Point 0.6.0 [🔗](#)

[Source Code](#)

Overview

ETH BALANCE

22.107107595731891311 ETH

ETH VALUE

\$49,187.84 (@ \$2,224.98/ETH)



0x482b0074fdb4eae8...

Handle Ops

1 hr 4 mins ago



0xb6b00a401b9cb999...

Handle Ops

1 hr 49 mins ago



0x398700619eab3f013...

Handle Ops

2 hrs 7 mins ago



0xa2668e4496238231...

Handle Ops

2 hrs 40 mins ago



0xe09e6cb46a02c6e1...

Handle Ops

2 hrs 43 mins ago

The EntryPoint contract
on Ethereum

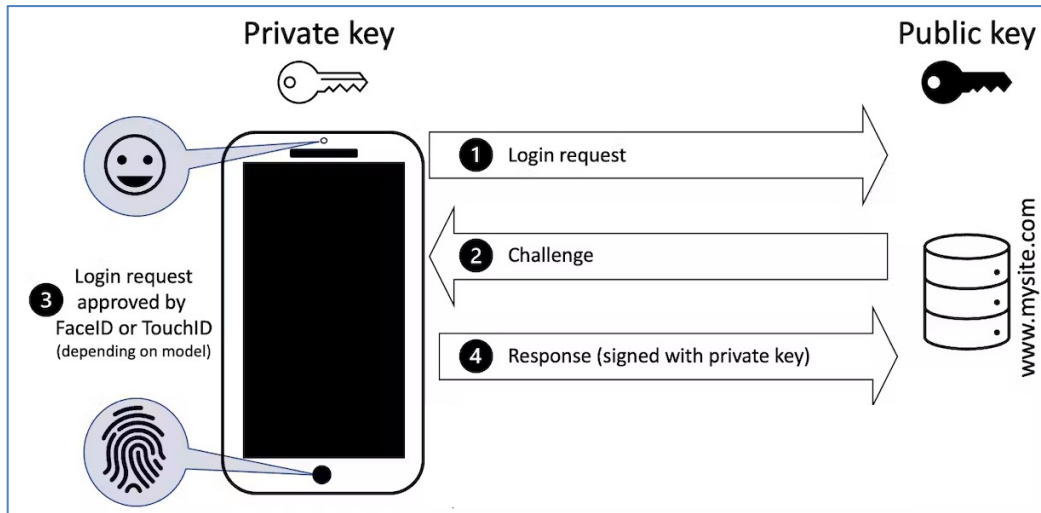
Some Tx from bundlers

An interesting application of AA: Passkey Tx

User authenticates Tx to AA wallet using TouchID

Good user experience:

- Signing key is backed up to iCloud
- Signing key is available on all user devices

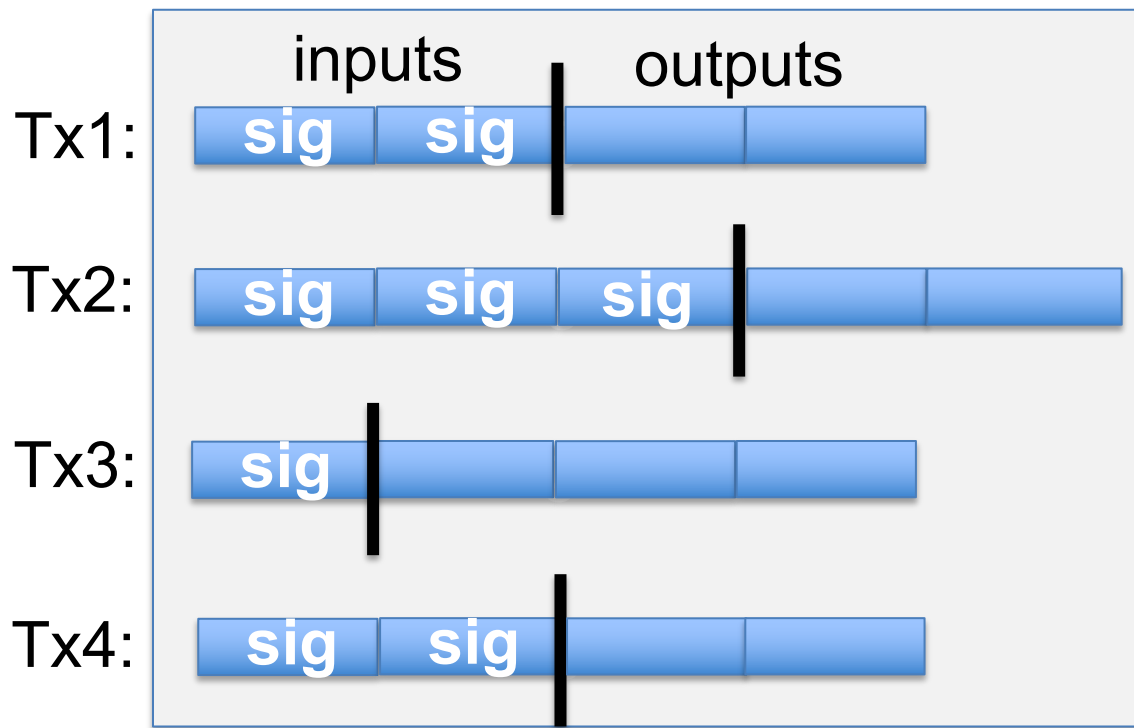


The challenge: Passkeys uses sig. scheme not supported in the EVM
⇒ need to use a SNARK to prove valid sig to on-chain wallet.

Fun crypto tricks

BLS signatures

one Bitcoin block



Signatures make up most of Tx data.

Can we compress signatures?

- Yes: aggregation!
- not possible for ECDSA

BLS Signatures

Used in modern blockchains: Ehtereum 2.0, Dfinity, Chia, etc.

The setup:

- $G = \{1, g, \dots, g^{q-1}\}$ a cyclic group of prime order q
- $H: M \times G \rightarrow G$ a hash function (e.g., based on SHA256)

BLS Signatures

KeyGen(): choose random α in $\{1, \dots, q\}$

output $sk = \alpha$, $pk = g^\alpha \in G$

Sign(sk, m): output $sig = H(m, pk)^\alpha \in G$

Verify(pk, m, sig): output accept if $\log_g(pk) = \log_{H(m, pk)}(sig)$

Note: signature on m is unique! (no malleability)

How does verify work?

A pairing: an efficiently computable function $e: G \times G \rightarrow G'$

such that $e(g^\alpha, g^\beta) = e(g, g)^{\alpha\beta}$ for all $\alpha, \beta \in \{1, \dots, q\}$

and is not degenerate: $e(g, g) \neq 1$

Observe: $\log_g(pk) = \log_{H(m,pk)}(sig)$

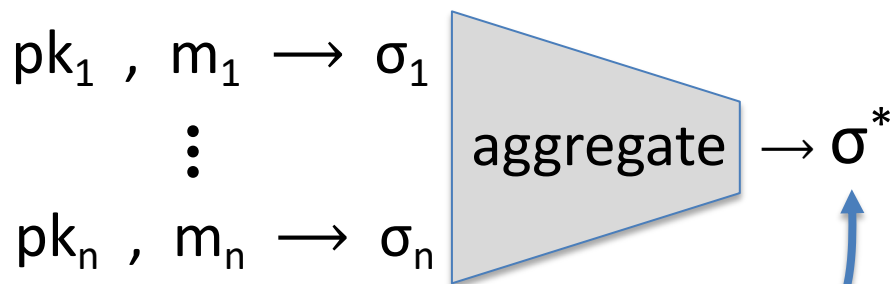
if and only if $e(g, sig) = e(pk, H(m,pk))$

$$\begin{array}{ccc} \parallel & & \parallel \\ e(g, H(m,pk)^\alpha) & = & e(g^\alpha, H(m,pk)) \end{array}$$

verify test

Properties: signature aggregation [BGLS'03]

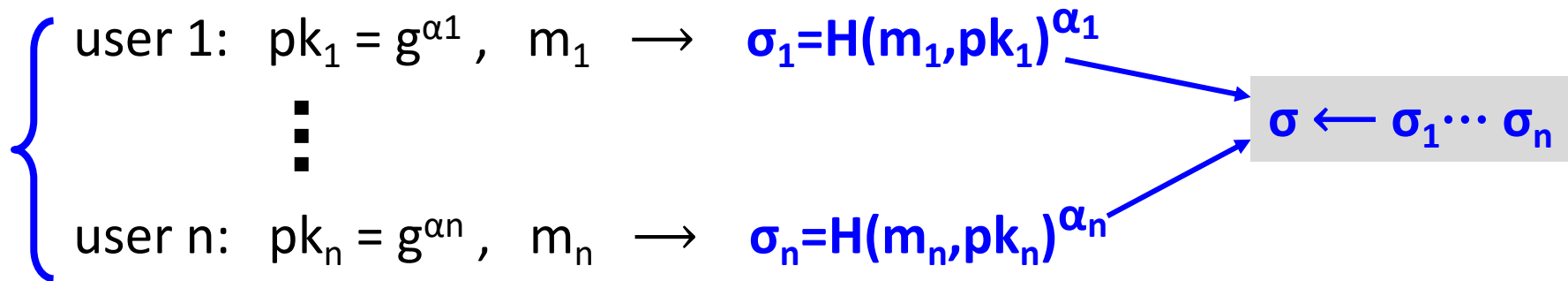
Anyone can compress n signatures into one



single short signature

$\text{Verify}(\bar{\mathbf{pk}}, \bar{\mathbf{m}}, \sigma^*) = \text{"accept"}$
convinces verifier that
for $i=1, \dots, n$:
user i signed msg m_i

Aggregation: how



Verifying an aggregate signature: (incomplete)

$$\prod_{i=1}^n e(H(m_i, pk_i), g^{\alpha_i}) \stackrel{?}{=} e(\sigma, g)$$

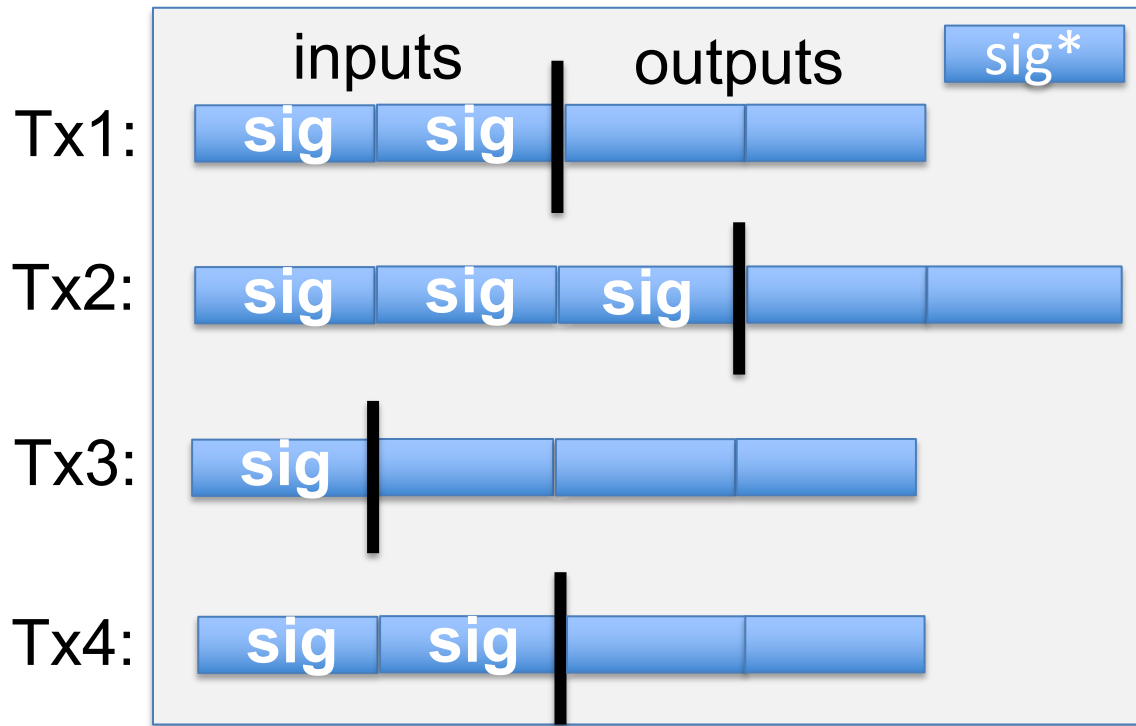
//

//

$$\prod_{i=1} e(H(m_i, pk_i)^{\alpha_i}, g) = e(\prod_{i=1} H(m_i, pk_i)^{\alpha_i}, g)$$

Compressing the blockchain with BLS

one Bitcoin block



if needed:

compress all
signatures in a block
into a single
aggregate signatures

⇒ shrink block

or: aggregate in smaller
batches

Taproot: semi-private scripts in Bitcoin

Script privacy

Currently: Bitcoin scripts must be fully revealed in spending Tx

Can we keep the script secret?

Answer: Yes, easily! when all goes well ...

How?

ECDSA and Schnorr public keys:

- KeyGen(): $sk = \alpha$, $pk = g^\alpha \in G$ for α in $\{1, \dots, q\}$

Suppose $sk_A = \alpha$, $sk_B = \beta$.

- Alice and Bob can sign with respect to $pk = pk_A \cdot pk_B = g^{\alpha+\beta}$
⇒ an interactive protocol between Alice and Bob
(note: much simpler with BLS)
⇒ Alice & Bob can imply consent to Tx by signing with $pk = g^{\alpha+\beta}$

How?

S: Bitcoin script that must be satisfied to spend a UTXO U

S involves only Alice and Bob. Let $pk_{AB} = pk_A \cdot pk_B$

Goal: keep S secret when possible.

How: modify S so that a signature with respect to

$$pk = pk_{AB} \cdot g^{H(pk_{AB}, S)}$$

is sufficient to spend UTXO, without revealing S !!

The main point

- If parties agree to spend UTXO,
 $\Rightarrow$ sign with respect to pk_{AB} and spend while keeping S secret
- If disagreement, Alice can reveal S
 and spend UTXO by proving that she can satisfy S .

Taproot pk compactly supports both ways to spend the UTXO