What are SNARKs and what are they good for?

Dan Boneh

Stanford University

What is a zk-SNARK ? (intuition)

SNARK: a succinct proof that a certain statement is true

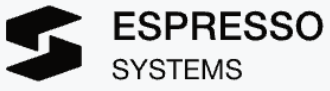
Example statement: "I know an m such that SHA256$(m) = 0$"

- **SNARK**: the proof is **"short"** and **"fast"** to verify

  [if m is 1GB then the trivial proof (the message m) is neither]

- **zk-SNARK**: the proof "reveals nothing" about m (privacy for m)

Commercial interest in SNARKs



Many more building applications that use SNARKs

Why so much commercial interest?

Babai-Fortnow-Levin-Szegedy 1991:

In this setup, a single reliable PC can monitor the operation of a herd of supercomputers working with unreliable software.

"Checking Computations in Polylogarithmic Time"

Why so much commercial interest?

Babai-Fortnow-Levin-Szegedy 1991:

a slow and expensive computer

In this setup, a ~~single reliable PC can~~ monitor the operation of a herd of ~~supercomputers~~ *GPUs* working with unreliable software.

"Checking Computations in Polylogarithmic Time"

Why so much commercial interest?

Babai-Fortnow-Levin-Szegedy 1991:

In this setup, a ~~single reliable PC can~~ *L1 blockchain* monitor the operation of a herd of ~~supercomputers~~ *GPUs* working with unreliable software.

"Checking Computations in Polylogarithmic Time"

Blockchain Applications I

Outsourcing computation: (no need for zero knowledge)

L1 chain quickly verifies the work of an off-chain service

To minimize gas: need a short proof, fast to verify

Examples:

- **Scalability:** proof-based Rollups (zkRollup)

  off-chain service processes a batch of Tx;

  L1 chain verifies a succinct proof that Tx were processed correctly

- **Bridging blockchains:** proof of consensus (zkBridge)

  Chain A produces a succinct proof about its state. Chain B verifies.

Blockchain Applications II

Some applications require zero knowledge (privacy):

- **Private Tx on a public blockchain**:
  - zk proof that a private Tx is valid (Tornado cash, Zcash, IronFish, Aleo)
- **Compliance:**
  - Proof that a private Tx is compliant with banking laws (Espresso)
  - Proof that an exchange is solvent in zero-knowledge (Raposa)

More on these blockchain applications in a minute

Many non-blockchain applications

Blockchains drive the development of SNARKs

… but **many** non-blockchain applications benefit

Why is all this possible now?

The breakthrough: new fast SNARK provers

- Proof generation time is linear (or quasilinear) in computation size
- **Many** beautiful ideas … will cover during the day

a large bibliography: a16zcrypto.com/zero-knowledge-canon

What is a SNARK?

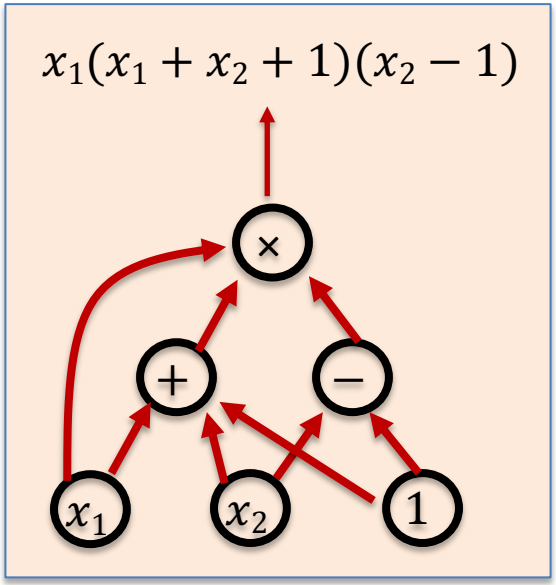
Review: arithmetic circuits

Fix a finite field $\mathbb{F} = \{0, \ldots, p-1\}$ for some prime p>2.

Arithmetic circuit: $C: \mathbb{F}^n \to \mathbb{F}$

- directed acyclic graph (DAG) where
  internal nodes are labeled +, −, or ×
  inputs are labeled $1, x_1, \ldots, x_n$
- defines an n-variate polynomial with an evaluation recipe

$|C|$ = # gates in C

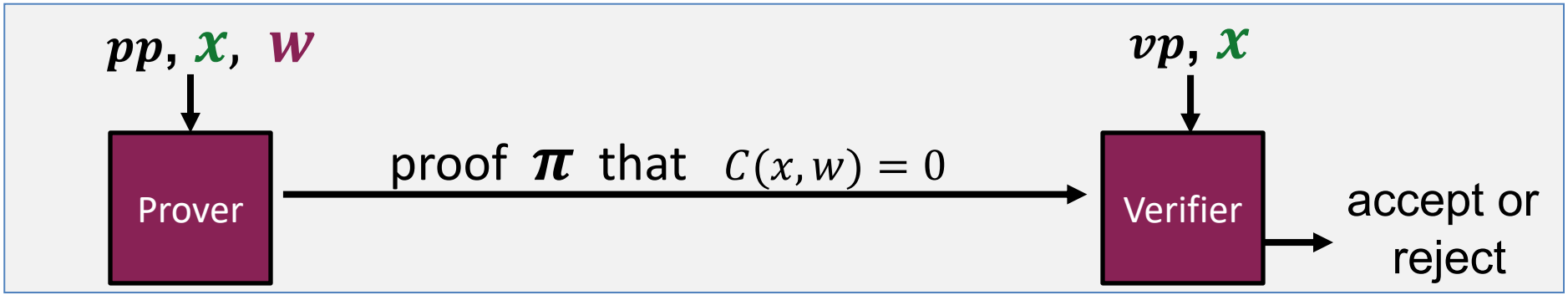


(preprocessing) NARK: Non-interactive ARgument of Knowledge

Public arithmetic circuit: $C(\boldsymbol{x}, \boldsymbol{w}) \rightarrow \mathbb{F}$

public statement in $\mathbb{F}^n$ — secret witness in $\mathbb{F}^m$

Preprocessing (setup): $\mathbf{S}(C) \rightarrow$ public parameters $(\boldsymbol{pp}, \boldsymbol{vp})$



(preprocessing) NARK: Non-interactive ARgument of Knowledge

A **preprocessing NARK** is a triple (S, P, V):

- **S**$(C) \rightarrow$ public parameters (pp, vp) for prover and verifier
- **P**$(pp, \boldsymbol{x}, \boldsymbol{w}) \rightarrow$ proof π
- **V**$(vp, \boldsymbol{x}, \boldsymbol{\pi}) \rightarrow$ accept or reject

all algs. and adversary have access to a random oracle

NARK: requirements (informal)

Prover P(pp, $\boldsymbol{x}$, $\boldsymbol{w}$) — Verifier V (vp, $\boldsymbol{x}$, $\boldsymbol{\pi}$)

proof π →

accept or reject

Complete: $\forall x, w$: $C(\boldsymbol{x}, \boldsymbol{w}) = 0 \Rightarrow$ Pr[V(vp, x, P(pp, $\boldsymbol{x}$, $\boldsymbol{w}$)) = accept] = 1

Adaptively **knowledge sound**: V accepts ⇒ P "knows" $\boldsymbol{w}$ s.t. $C(\boldsymbol{x}, \boldsymbol{w}) = 0$

(an extractor E can extract a valid $\boldsymbol{w}$ from P)

Optional: **Zero knowledge**: ($C, pp, vp, \boldsymbol{x}, \pi$) "reveal nothing new" about $\boldsymbol{w}$

(witness exists ⇒ can simulate the proof)

SNARK: a Succinct ARgument of Knowledge

A **succinct preprocessing NARK** is a triple (S, P, V):

- **S**$(C) \rightarrow$ public parameters (pp, vp) for prover and verifier
- **P**$(pp, \boldsymbol{x}, \boldsymbol{w}) \rightarrow$ **short** proof π ; $\text{len}(\pi) = O_\lambda(\,\textbf{polylog}(|\boldsymbol{C}|))$
- **V**$(vp, \boldsymbol{x}, \boldsymbol{\pi})$ **fast to verify** ; $\text{time(V)} = O_\lambda(|x|,\ \textbf{polylog}(|\boldsymbol{C}|))$

short "summary" of circuit

V has no time to read C !!

[for some SNARKs, $\text{len}(\pi) = \text{time}(V) = O_\lambda(1)$]

SNARK: a Succinct ARgument of Knowledge

SNARK: a NARC (complete and knowledge sound) that is **succinct**

zk-SNARK: a SNARK that is also **zero knowledge**

The trivial SNARK is not a SNARK

(a) Prover sends $\boldsymbol{w}$ to verifier,

(b) Verifier checks if $C(\boldsymbol{x}, \boldsymbol{w}) = 0$ and accepts if so.

Problems with this:

(1) $\boldsymbol{w}$ might be long: we want a "short" proof

(2) computing $C(\boldsymbol{x}, \boldsymbol{w})$ may be hard: we want a "fast" verifier

(3) $\boldsymbol{w}$ might be secret: prover might not want to reveal $\boldsymbol{w}$ to verifier

The SNARK zoo … next lecture

STARK

Plonky2

Breakdown

Orion

Spartan

Bulletproofs

Halo2

Nova

Hyrax

Groth16

Plonk

Marlin

Sonic

Gemini

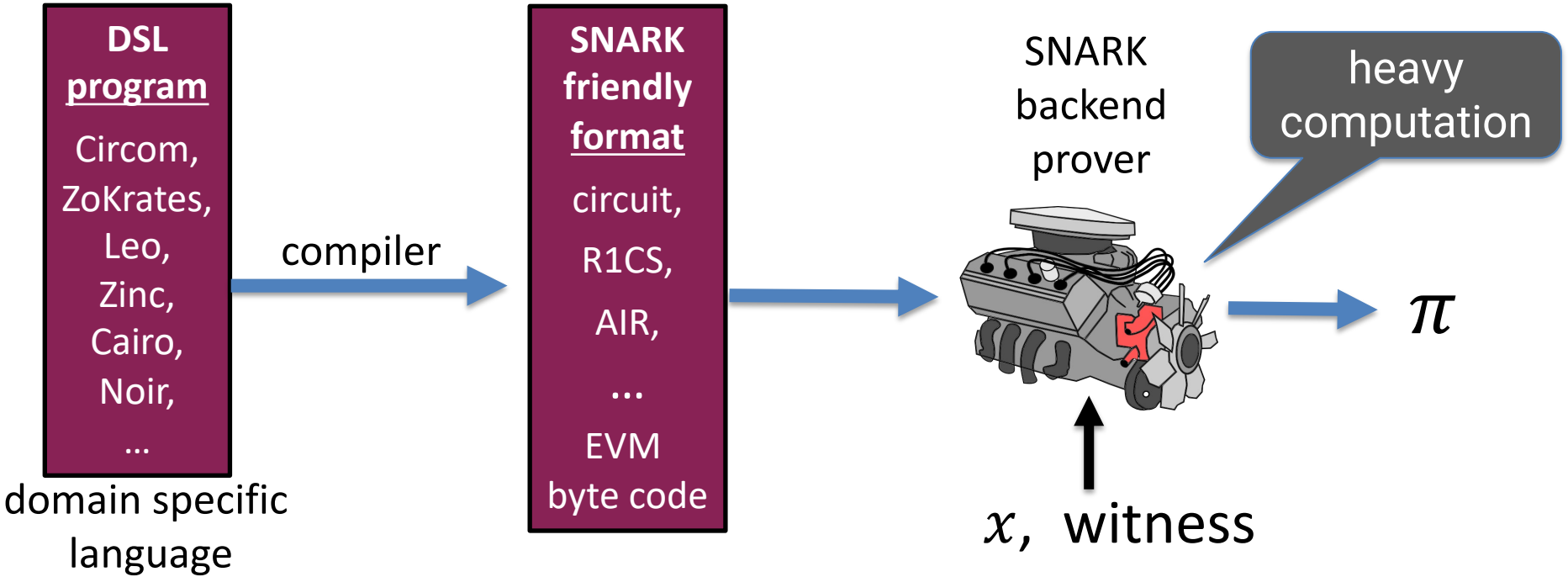
DARK

Hyperplonk

⋮

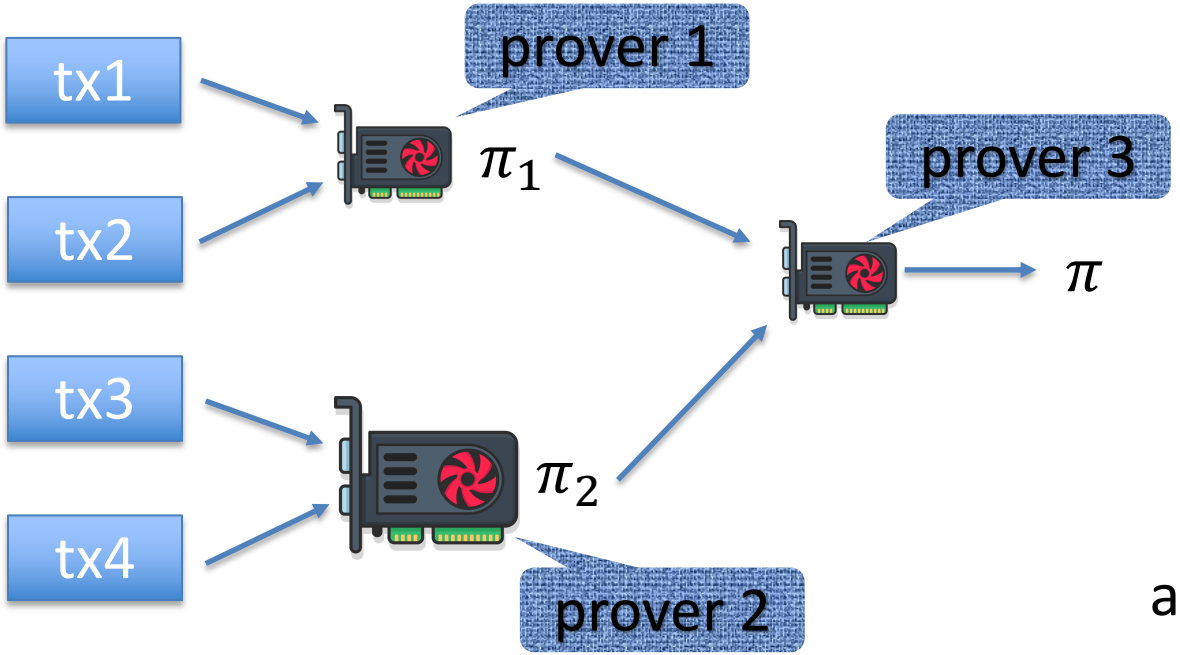
Open: one SNARK to rule them all

SNARKs in practice



The future: a market for ZK provers

Anyone with a GPU will be paid to create ZK proofs



market

selects provers
and distributes rewards

Applications of SNARKs

Three examples: (briefly)

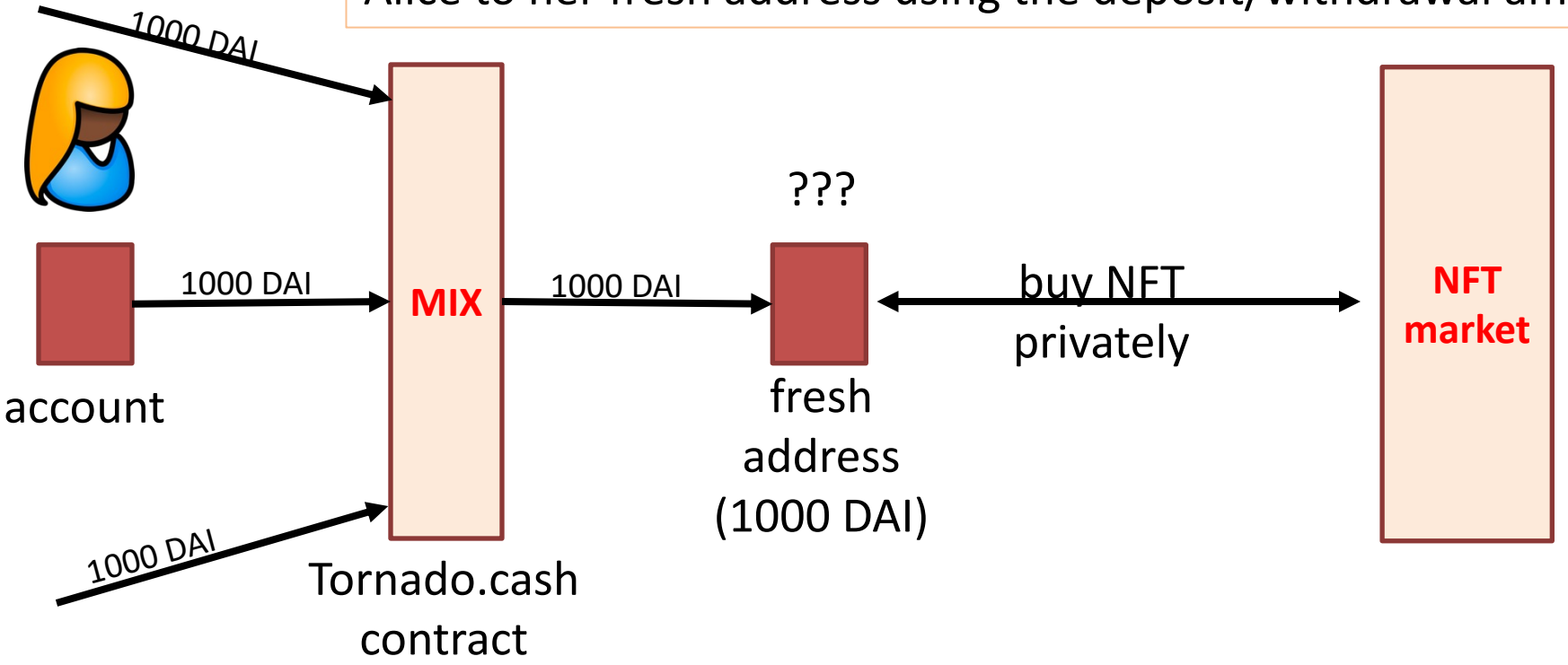
zkRollup, **zkBridge**, **Tornado**

(actually using ZK)

The Tornado Story

Privacy: Tornado – a ZK mixer

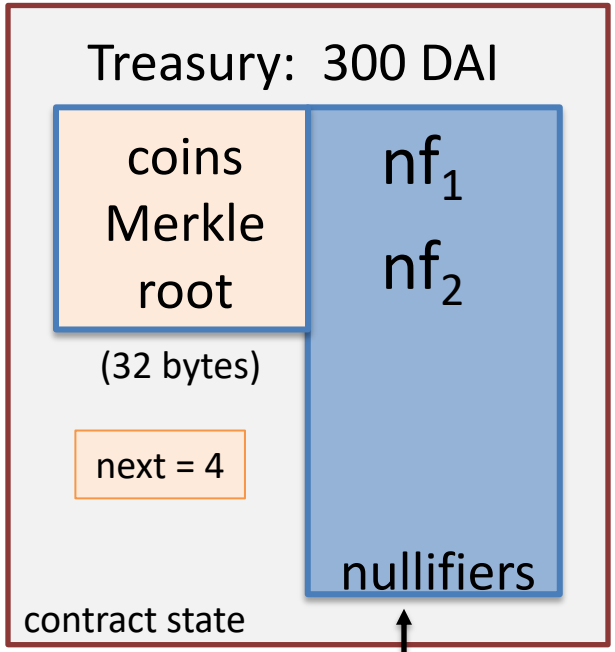
A common denomination (1000 DAI) is needed to prevent linking Alice to her fresh address using the deposit/withdrawal amount



The tornado cash contract (simplified)

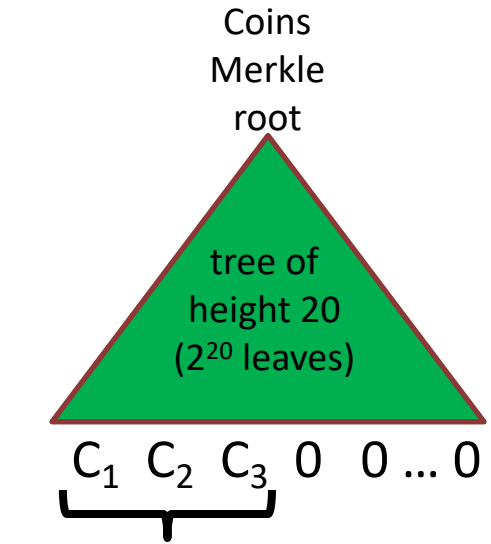
100 DAI pool:
each coin = 100 DAI

Currently:
- three coins in pool
- contract has 300 DAI
- two nullifiers stored



explicit list:
one entry per **spent coin**

$H_1, H_2: R \to \{0,1\}^{256}$ CRHF

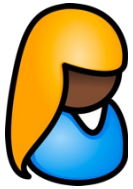


public list of coins

Tornado cash: deposit (simplified)

100 DAI pool:
each coin = 100 DAI

Alice deposits 100 DAI:



100 DAI

C_4 , MerkleProof(4)

Treasury: 300 DAI

coins Merkle root

(32 bytes)

next = 4

nf_1

nf_2

nullifiers

contract state

$H_1, H_2: R \rightarrow \{0,1\}^{256}$

Coins Merkle root

tree of height 20 (2^{20} leaves)

C_1 C_2 C_3 0 0 … 0

public list of coins

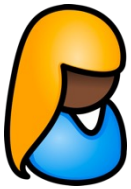
Build Merkle proof for leaf #4:
MerkleProof(4) (leaf=0)

choose random k, r in R

set $C_4 = H_1(k, r)$

explicit list:
one entry per **spent coin**

Tornado cash: deposit (simplified)



100 DAI

C_4 , MerkleProof(4)

coins Merkle root

(32 bytes)

next = 4

Tornado contract

$H_1, H_2: R \to \{0,1\}^{256}$

Coins Merkle root

tree of height 20 (2^{20} leaves)

C_1 C_2 C_3 0 0 … 0

public list of coins

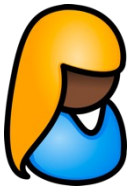
Tornado contract does:

(1) verify MerkleProof(4) with respect to current stored root

(2) use C_4 and MerkleProof(4) to compute updated Merkle root

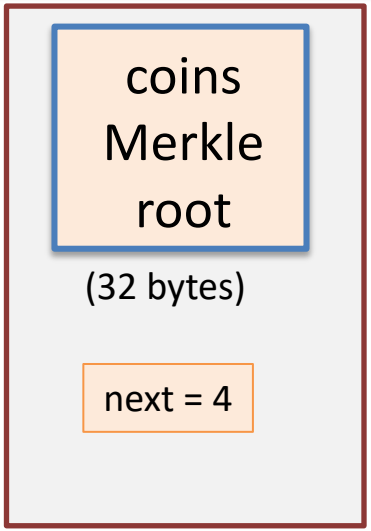
(3) update state

Tornado cash: deposit (simplified)

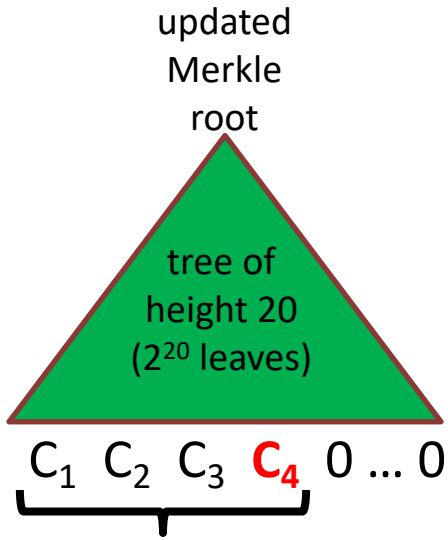


100 DAI

C_4 , MerkleProof(4)



$H_1, H_2: R \to \{0,1\}^{256}$



public list of coins

Tornado contract does:

(1) verify MerkleProof(4) with respect to current stored root

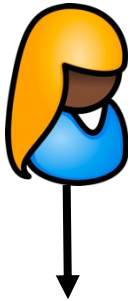
(2) use C_4 and MerkleProof(4) to compute updated Merkle root

(3) update state

Tornado cash: deposit (simplified)

100 DAI pool:
each coin = 100 DAI

Alice deposits 100 DAI:



100 DAI

C_4 , MerkleProof(4)

note: (k, r)
Alice keeps secret
(one note per coin)

Treasury: **400** DAI

updated Merkle root

(32 bytes)

next = **5**

nf_1

nf_2

nullifiers

updated contract state

Every deposit: new Coin added sequentially to tree

updated Merkle root

tree of height 20 (2^{20} leaves)

C_1 C_2 C_3 C_4 0 … 0

public list of coins

an observer sees who owns which leaves

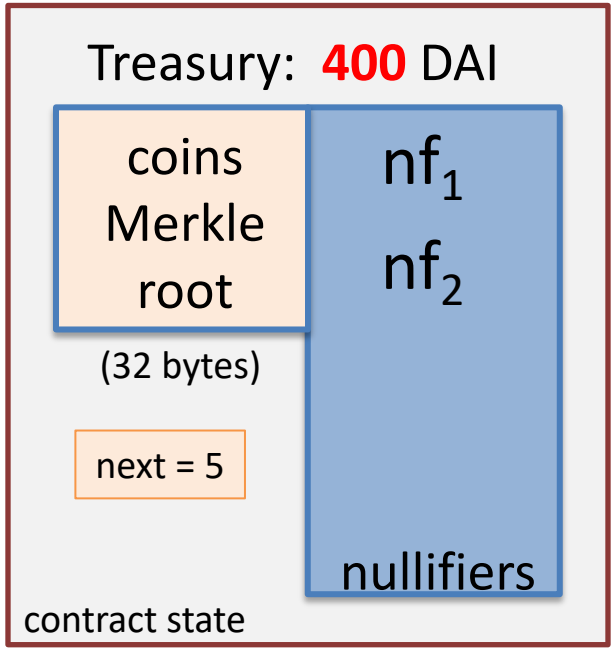
Tornado cash: withdrawal (simplified)

100 DAI pool:
each coin = 100 DAI

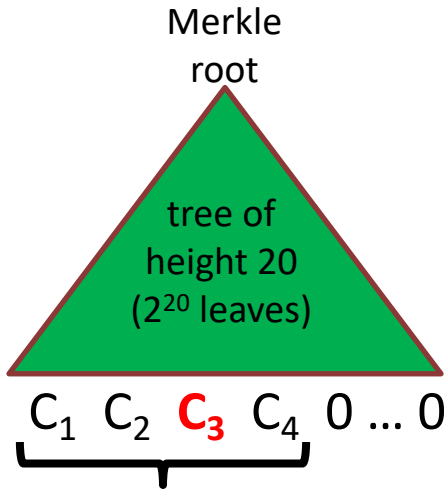
Withdraw coin #3 to addr A:

has note= (k', r')

set **nf** = $H_2(k')$



H_1, H_2: $R \to \{0,1\}^{256}$



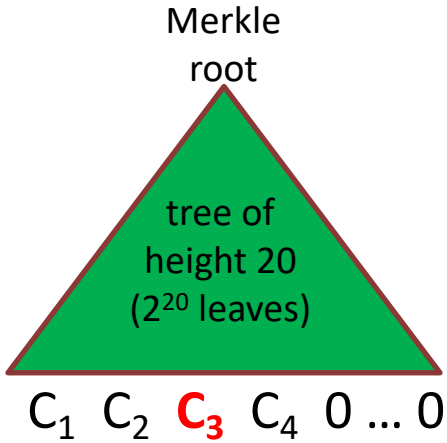
Bob proves "I have a note for some leaf in the coins tree, and its nullifier is **nf**"
(without revealing which coin)

Tornado cash: withdrawal (simplified)

H_1, H_2: $R \rightarrow \{0,1\}^{256}$

Withdraw coin #3 to addr A:

has note= (k', r') set **nf** = $H_2(k')$



Bob builds zk-SNARK proof π for

public statement x = (**root**, **nf**, **A**)

secret witness w = (k', r', C_3, MerkleProof(C_3))

where Circuit(x,w)=0 iff:

(i) C_3 = (leaf #3 of **root**), i.e. MerkleProof(C_3) is valid,

(ii) $C_3 = H_1(k', r')$, and

(iii) **nf** = $H_2(k')$.

(address A not used in Circuit)

Tornado cash: withdrawal (simplified)

H_1, H_2: $R \to \{0,1\}^{256}$

Withdr

The address A is part of the statement to ensure that a miner cannot change A to its own address and steal funds

Assumes the SNARK is **non-malleable**:

adversary cannot use proof π for x to build a proof π' for some "related" x'
(where in x' the address A is replaced by some A')

C_1 C_2 C_3 C_4 0 … 0

Bob builds zk-SNARK proof π for

public statement x = (**root**, **nf**, **A**)

secret witness w = (k', r', C_3 , MerkleProof(C_3))

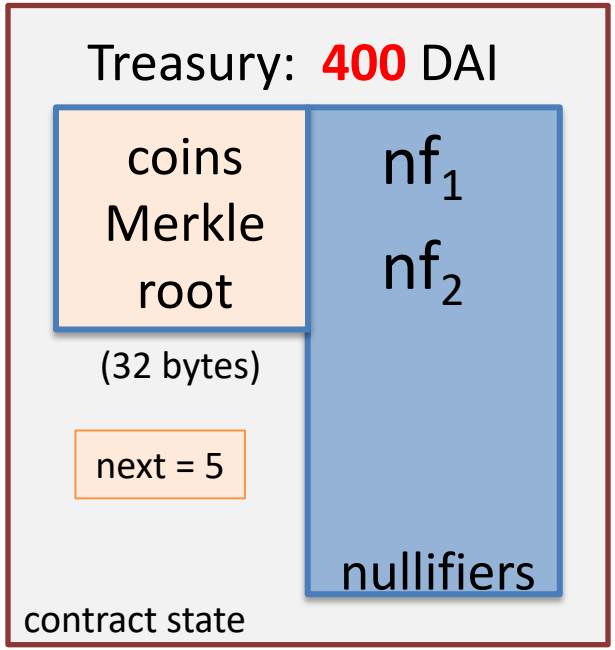
Tornado cash: withdrawal (simplified)

100 DAI pool:
each coin = 100 DAI

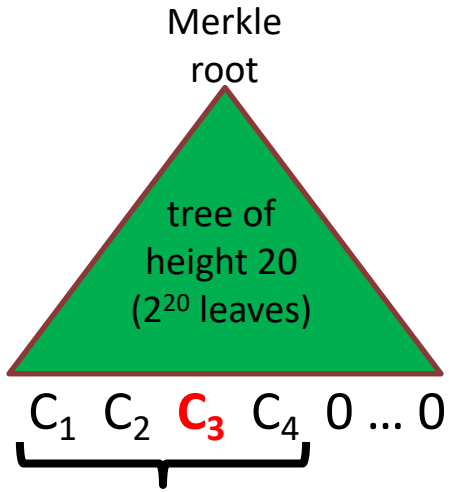
Withdraw coin #3 to addr A:

nf, proof π, **A**
(over Tor)

Bob's ID and coin C_3 are not revealed



$H_1, H_2: R \to \{0,1\}^{256}$



Contract checks (i) proof π is valid for (root, **nf,** **A**), and
(ii) **nf** is not in the list of nullifiers

Tornado cash: withdrawal (simplified)

100 DAI pool:
each coin = 100 DAI

Withdraw coin #3 to addr A:

nf, proof π, **A**
(over Tor)

100 DAI
to address **A**

Treasury: **300** DAI

coins Merkle root
(32 bytes)

next = 5

nf_1
nf_2
nf

nullifiers

contract state

$H_1, H_2: R \to \{0,1\}^{256}$

Merkle root

tree of height 20 (2^{20} leaves)

C_1 C_2 C_3 C_4 0 … 0

public list of coins
… but observer does not know which are spent

nf and π reveal nothing about which coin was spent.

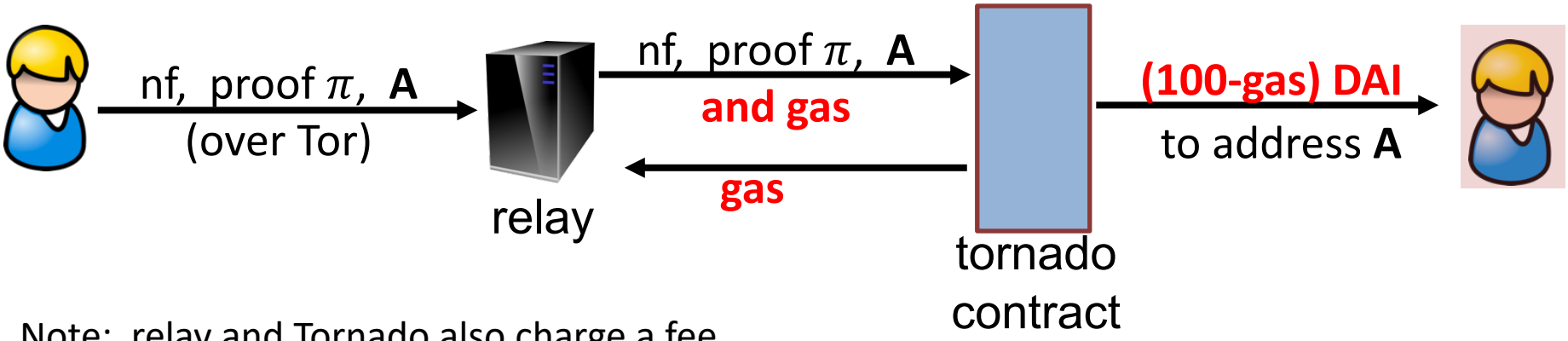
But, coin #3 cannot be spent again, because **$nf = H_2(k')$** is now nullified.

Who pays the withdrawal gas fee?

Problem: how does Bob pay for gas for the withdrawal Tx?

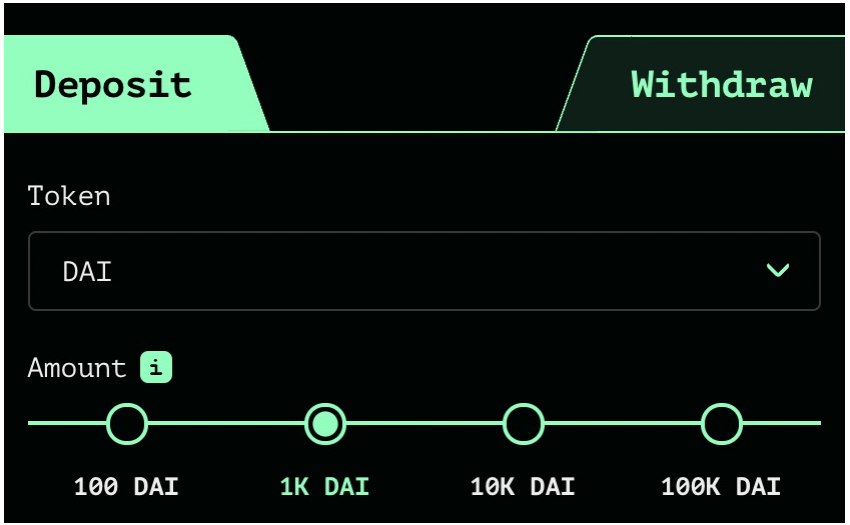
- If paid from Bob's address, then fresh address is linked to Bob

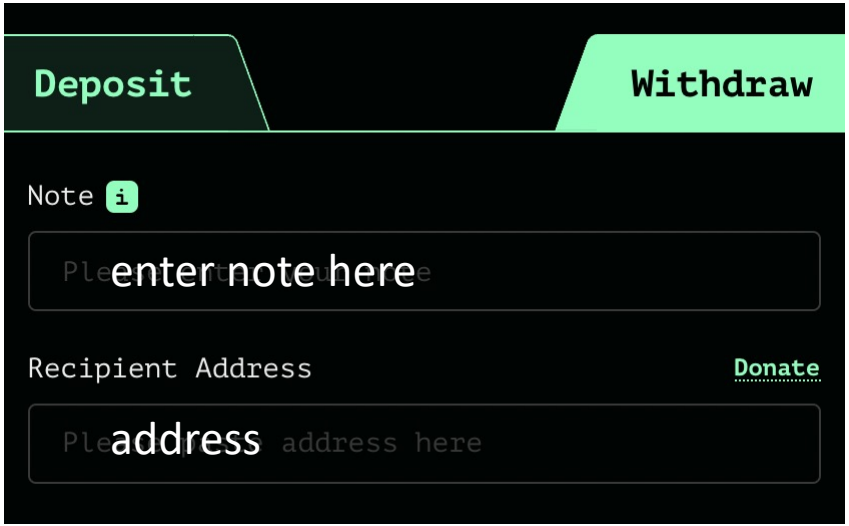
Tornado's solution: **Bob uses a relay**



Note: relay and Tornado also charge a fee

Tornado Cash: the UI





After deposit: get a note

Later, use note to withdraw

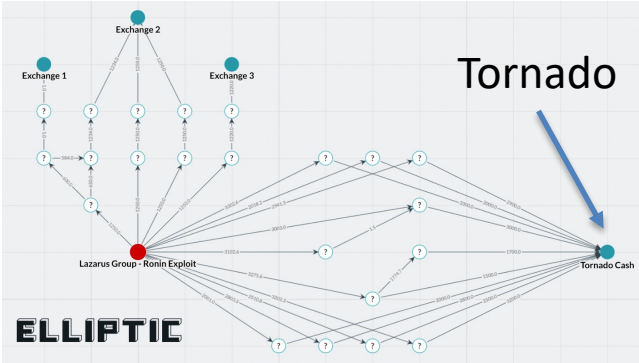
(SNARK proof generated in browser)

Tornado trouble … U.S. sanctions

The Ronin-bridge hack (2022):

- In late March: ≈600M USD stolen … $80M USD sent to Tornado
- April: Lazarus Group suspected of hack
- August: "U.S. Treasury Sanctions Virtual Currency Mixer Tornado Cash"
  - Lots of collateral damage … and two lawsuits

The lesson: complete anonymity in the payment system is problematic



Designing a compliant Tornado??

(1) deposit filtering: ensure incoming funds are not sanctioned

Chainalysis **SanctionsList** contract:

```
function isSanctioned(address addr) public view returns (bool) {
        return sanctionedAddresses[addr] == true ;
}
```

Reject funds coming from a sanctioned address.

Difficulties: (1) centralization, (2) slow updates

Designing a compliant Tornado??

(2) Withdrawal filtering: at withdrawal, require a ZK proof that the source of funds is not currently on sanctioned list.

How?

- modify the way Tornado computes Merkle leaves during deposit to include **msg.sender**.

  in our example Alice sets: $\mathbf{C_4} = [\, H_1(k, r),\ \mathbf{msg.sender}\,]$

- During withdrawal Bob proves in ZK that **msg.sender** in his leaf is not currently on sanctions list.

THE END

Scalability: zkRollup

Transaction rates (Tx/sec):

- Bitcoin: can process about **7 (Tx/sec)**
- Ethereum: can process about **15 (Tx/sec)**

Tx Fees fluctuate:
2$ to 60$
for simple Tx

- The visa network: can process up to **24,000 (Tx/sec)**

Can we scale blockchains to visa speeds? … with low Tx fees

How to process more Tx per second

Many ideas:

- Use a faster consensus protocol

reduces composability

- Parallelize: split the chain into independent **shards**
- Rollups: move the work somewhere else
- Payment channels: reduce the need to touch the chain

Recall: a basic layer-1 blockchain

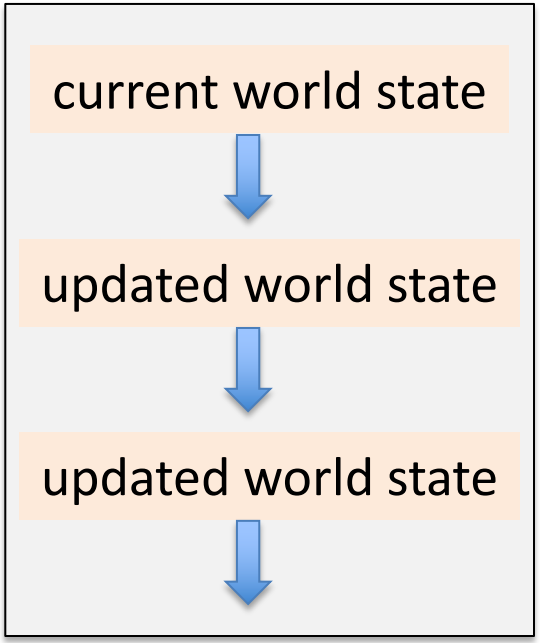
Can handle 15 Tx/sec …

Tx_A

Tx_B

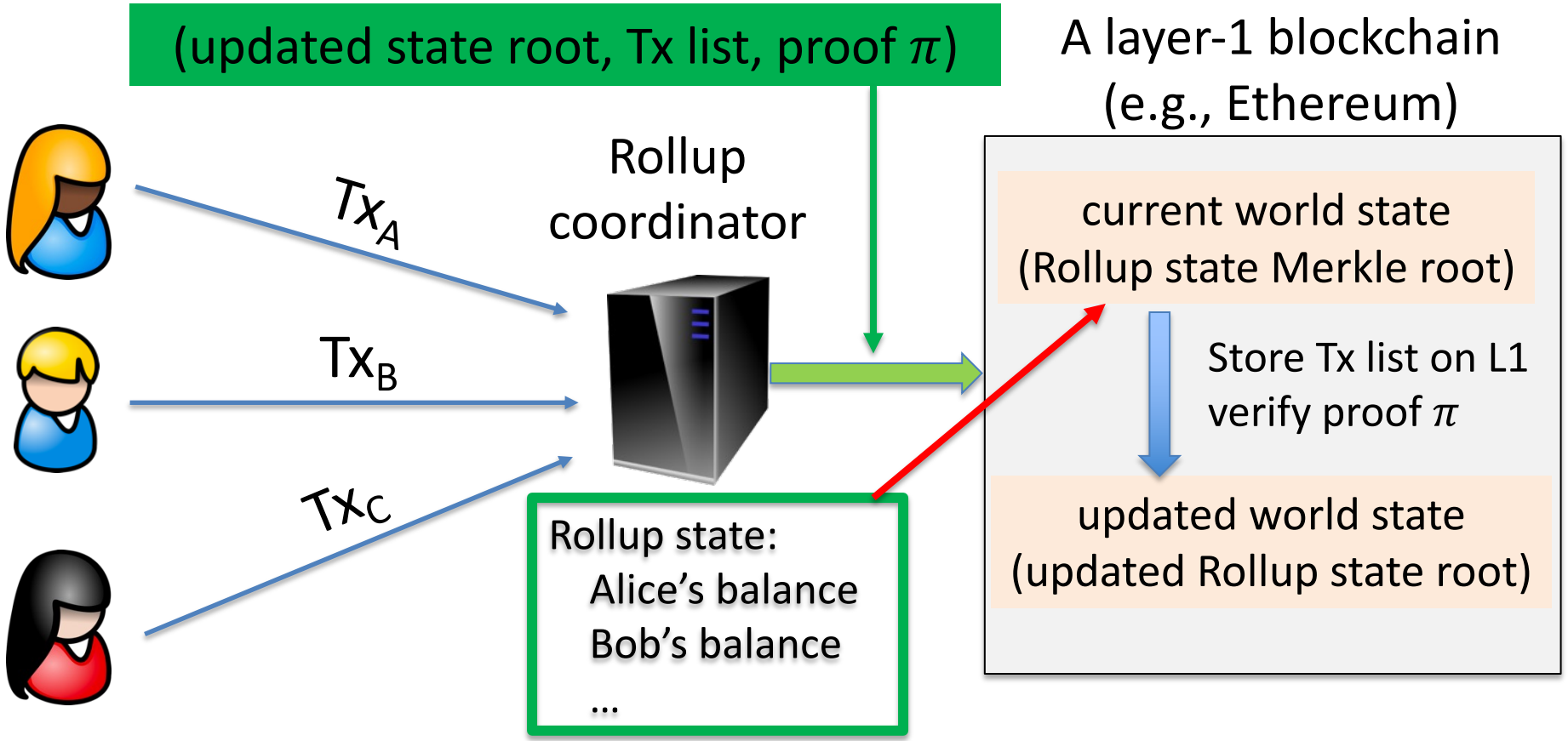
⋮

A layer-1 blockchain
(e.g., Ethereum)



World state: balances, storage, etc.

Rollup: batch many Tx into one (briefly)



Rollup: batch many Tx into one (briefly)

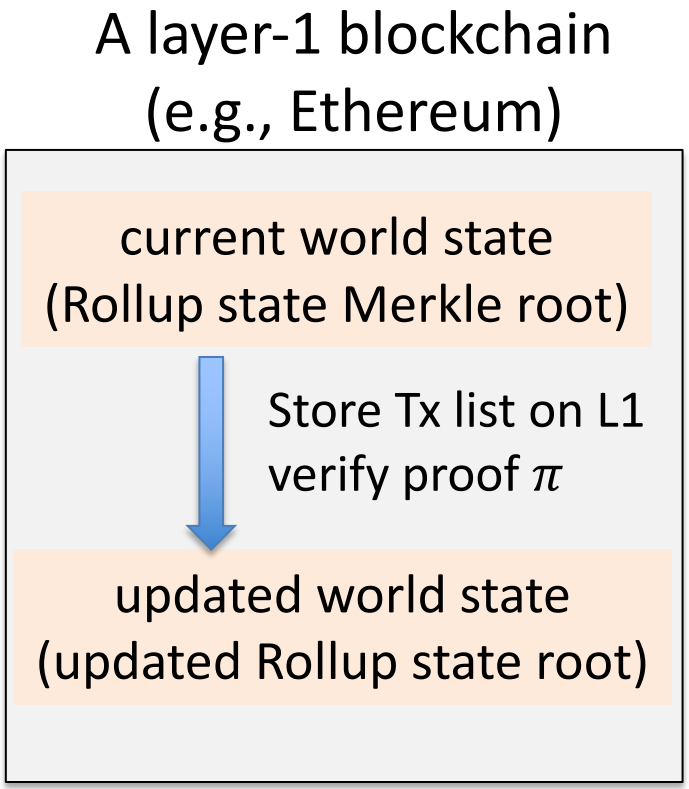
Key point:

- *Hundreds* of transactions on Rollup state are batched into a *single* transaction on layer-1

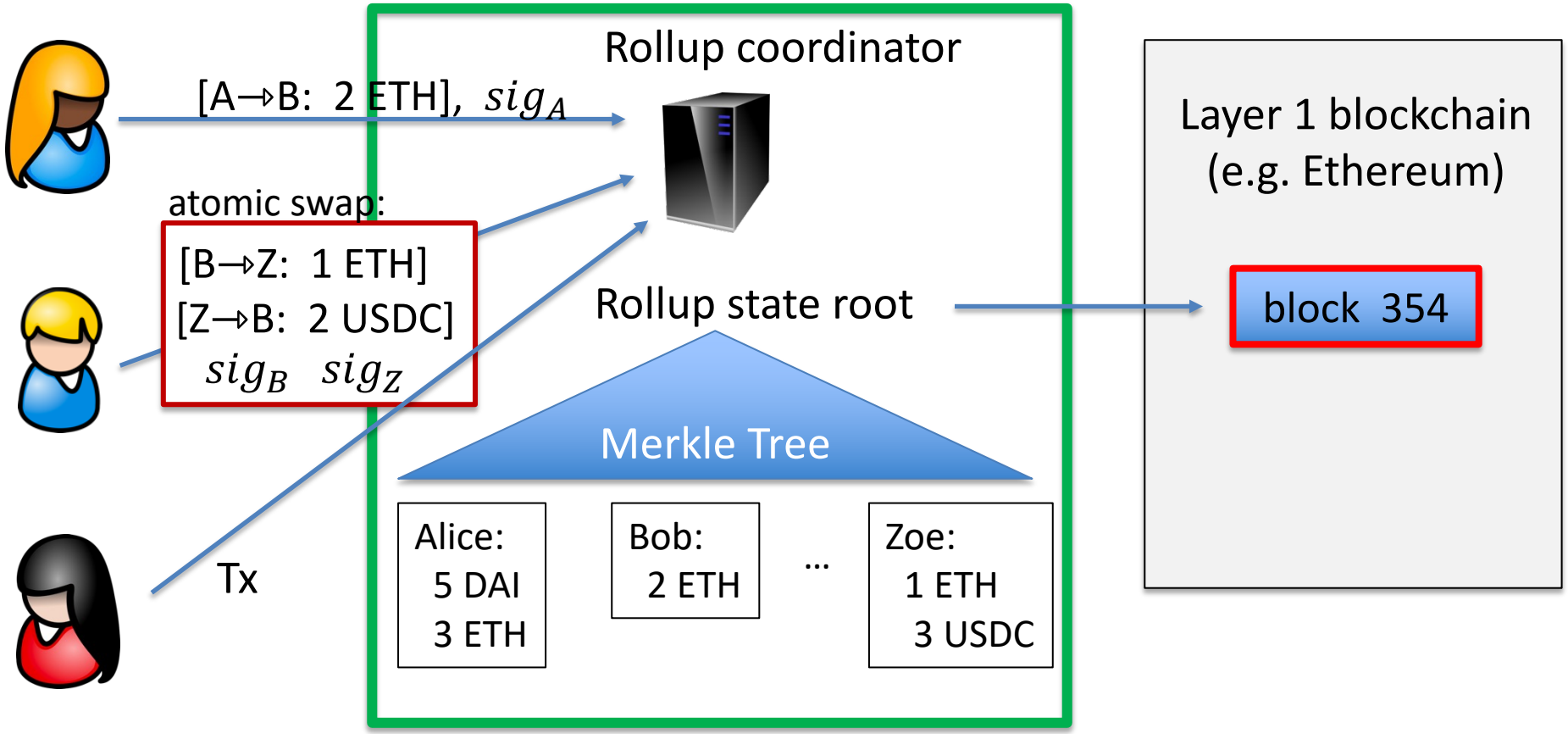
  ⇒ 100x speed up in Tx/sec

- Let's see how …

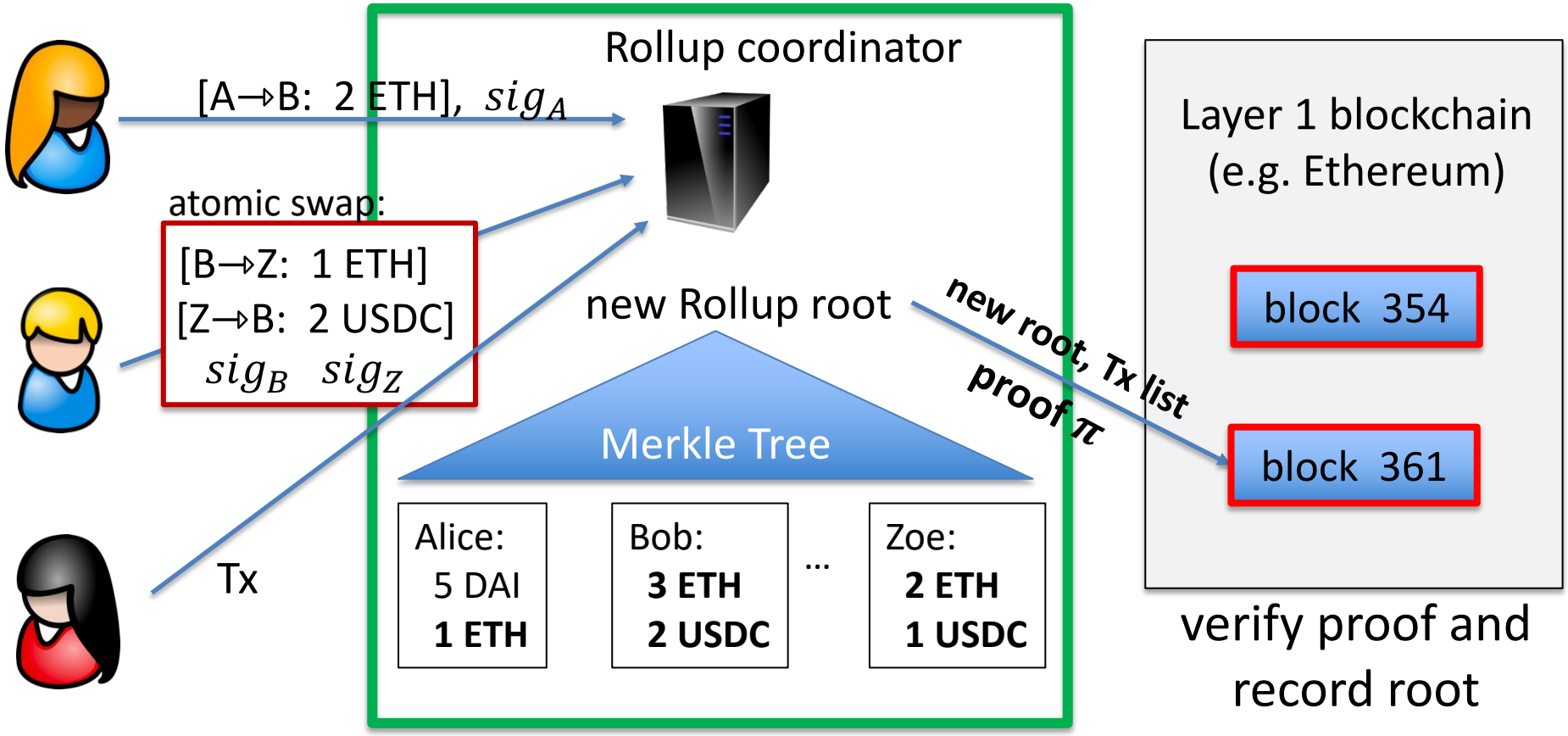
Rollup state:
Alice's balance
Bob's balance
…



Rollup operation (simplified)



Rollup operation (simplified)



What the SNARK proof proves

SNARK proof is **short** and **fast** to verify:

⇒ Cheap to verify proof on the slow L1 chain (with EVM support)

(usually not a zero knowledge proof)

Public statement: (old state root, new state root, Tx list)

Witness: (state of each touched account pre- and post- batch,
Merkle proofs for touched accounts, user sigs)

SNARK proof proves that:

(1) all user sigs on Tx are valid, (2) all Merkle proofs are valid,
(3) post-state is the result of applying Tx list to pre-state

An example (StarkNet -- using STARK proofs)

Block

Number	Hash	Status	Num. of Txs	Age
PENDING	PENDING	PENDING	64	3min
13011	0x0432…2380	ACCEPTED_ON_L2	82	8min
13010	0x0492…f0d1	ACCEPTED_ON_L2	122	15min
13009	0x0081…b7af	ACCEPTED_ON_L2	127	24min
		•••		
12868	0x060c…15eb	ACCEPTED_ON_L2	58	8h
12867	0x0654…3b0f	ACCEPTED_ON_L1	72	9h
12866	0x0779…57d6	ACCEPTED_ON_L1	63	9h
12865	0x06ae…943f	ACCEPTED_ON_L1	97	9h

Tx posted on L1 (Ethereum) about every eight hours

Source: starkscan.co

zkEVM

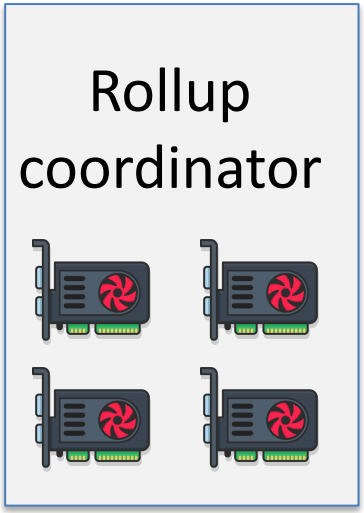
When a contract (e.g. Uniswap) runs on a Rollup:

- coordinator builds a SNARK proof of correct execution of an EVM program ⇒ called a **zkEVM**
- Generating proof is a heavyweight computation

  … verifying proof is fast

Two flavors of zkEVM:

- Prove that EVM bytecode ran correctly
  (Polygon zkEVM, Scroll)
- Compile Solidity to a SNARK-friendly circuit
  (MatterLabs)



(lots of GPUs)

Why write Tx list to L1?

Coordinate cannot steal funds, but can deny service …

What happens if coordinator fails and/or disappears?

- Solution: start a new coordinator

  ⇒ need the entire transaction history to reconstitute state

Writing Tx list on chain uses the L1 for data availability

- Other solutions: data availability committee.