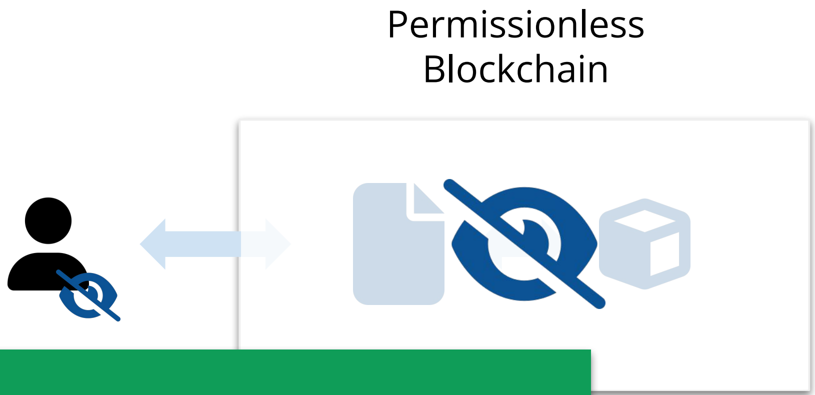
⚡ Zapper: Smart Contracts with Data and Identity Privacy

Samuel Steffen Benjamin Bichsel Martin Vechev

ETH Zurich, Switzerland

Introduction: Privacy for Smart Contracts



Goal: hide data and user identities

Existing work

- weak privacy guarantees
- strong trust assumptions
- manual instantiation of crypto

Idea

Zerocash [Sasson et al., 2014] / Zcash

strong privacy guarantees

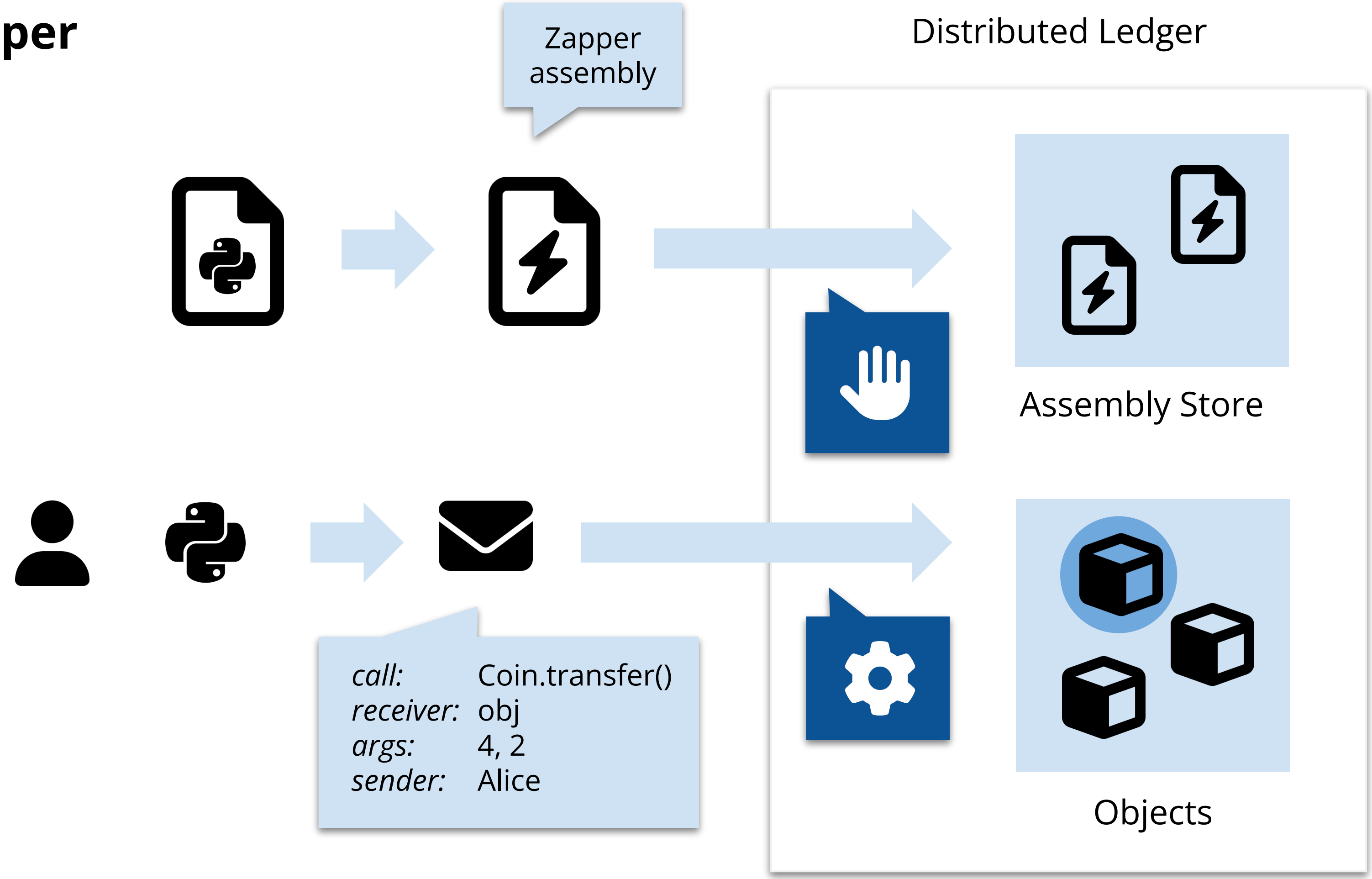
not programmable

make programmable

Zapper

but avoid limitations of previous work (e.g., ZEXE [Bowe et al., 2020])

Zapper



Zapper assembly

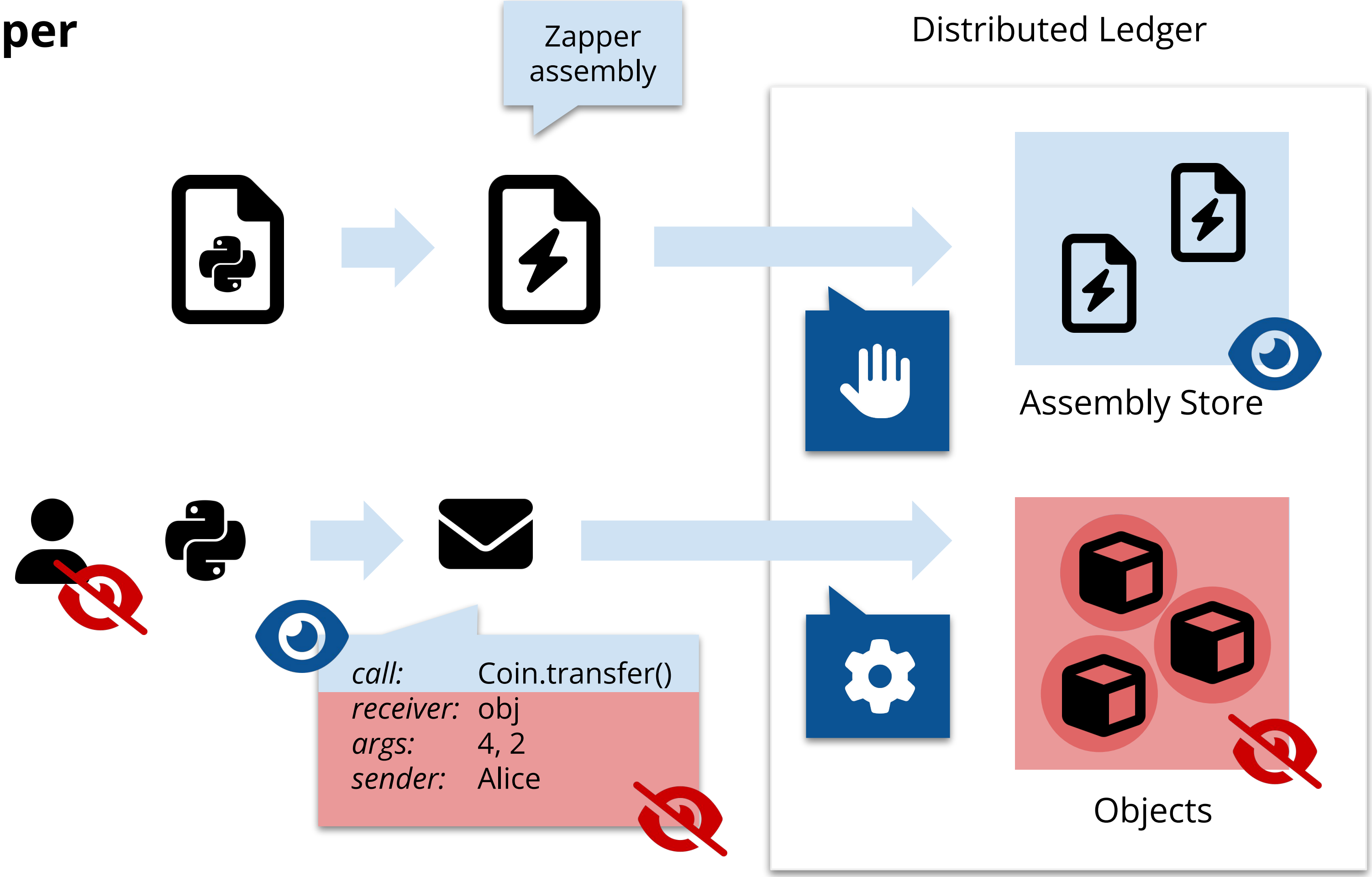
Distributed Ledger

Assembly Store

Objects

call:	Coin.transfer()
receiver:	obj
args:	4, 2
sender:	Alice

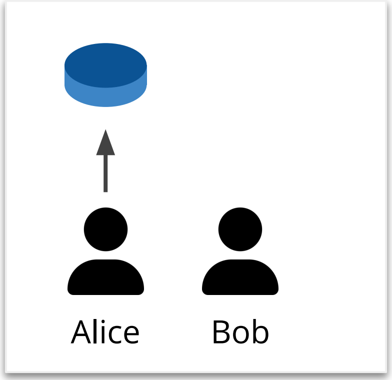
Zapper



call: Coin.transfer()
receiver: obj
args: 4, 2
sender: Alice

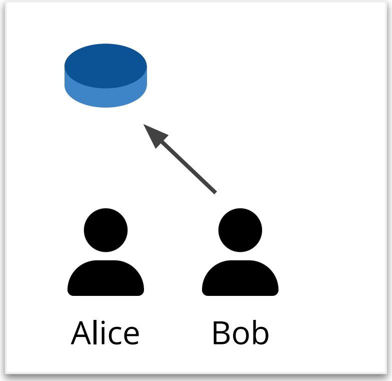
Example: Coin Puzzle

```
class Coin(Contract):
  val: Uint
  # owner: Address

  def transfer(self, to: Address):
    require(self.owner == self.me)
    self.owner = to
```



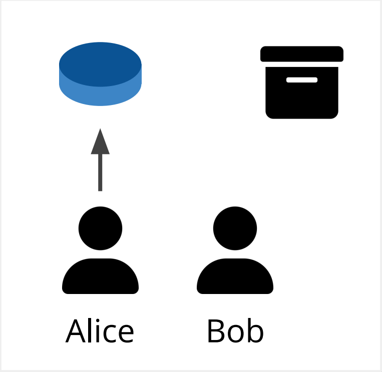
Example: Coin Puzzle

```
class Coin(Contract):
  val: Uint
  # owner: Address

  def transfer(self, to: Address):
    require(self.owner == self.me)
    self.owner = to
```



Example: Coin Puzzle

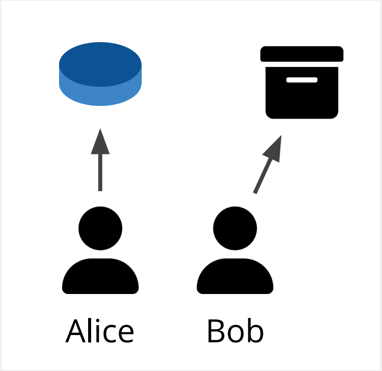
```
class Coin(Contract):
  val: Uint
  # owner: Address

  def transfer(self, to: Address):
    require(self.owner == self.me)
    self.owner = to
```



```
class Wrapper(Contract):
  coin: Coin
  # owner: Address

  def constructor(self, coin: Coin,
                        owner: Address):
    self.owner = owner
    self.coin = coin
    coin.transfer(self.address)

  def puzzle(self, sol: Uint) -> Bool:
    ...

  def open(self, sol: Uint):
    require(self.puzzle(sol))
    self.coin.transfer(self.me,
      sender_is_self=True)
```

Example: Coin Puzzle

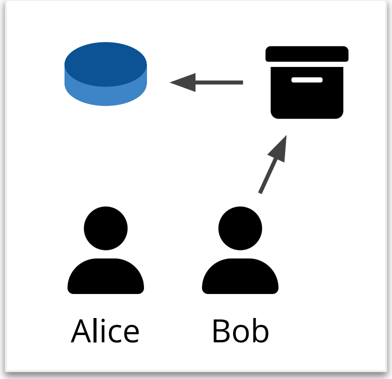
```
class Coin(Contract):
  val: Uint
  # owner: Address

  def transfer(self, to: Address):
    require(self.owner == self.me)
    self.owner = to
```



```
class Wrapper(Contract):
  coin: Coin
  # owner: Address

  def constructor(self, coin: Coin,
                  owner: Address):
    self.owner = owner
    self.coin = coin
    coin.transfer(self.address)

  def puzzle(self, sol: Uint) -> Bool:
    ...

  def open(self, sol: Uint):
    require(self.puzzle(sol))
    self.coin.transfer(self.me,
      sender_is_self=True)
```

Example: Coin Puzzle

```
class Coin(Contract):
  val: Uint
  # owner: Address

  def transfer(self, to: Address):
    require(self.owner == self.me)
    self.owner = to
```



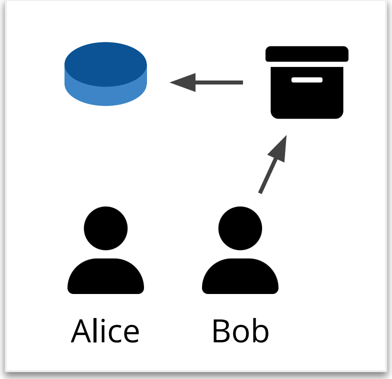
```
class Wrapper(Contract):
  coin: Coin
  # owner: Address

  def constructor(self, coin: Coin,
                  owner: Address):
    self.owner = owner
    self.coin = coin
    coin.transfer(self.address)
                  f,

  def open(self, sol: Uint):
    require(self.puzzle(sol))
    self.coin.transfer(self.me,
      sender_is_self=True)
```

pointers and *function calls*

objects can be owners

Example: Coin Puzzle

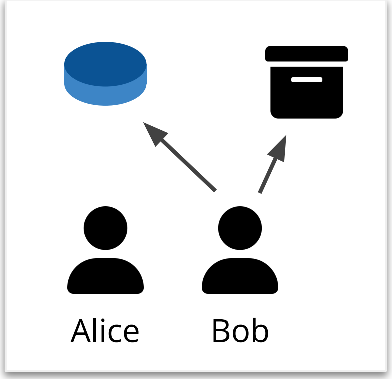
```
class Coin(Contract):
  val: Uint
  # owner: Address

  def transfer(self, to: Address):
    require(self.owner == self.me)
    self.owner = to
```



```
class Wrapper(Contract):
  coin: Coin
  # owner: Address

  def constructor(self, coin: Coin,
                        owner: Address):
    self.owner = owner
    self.coin = coin
    coin.transfer(self.address)

  def puzzle(self, sol: Uint) -> Bool:
    ...

  def open(self, sol: Uint):
    require(self.puzzle(sol))
    self.coin.transfer(self.me,
      sender_is_self=True)
```

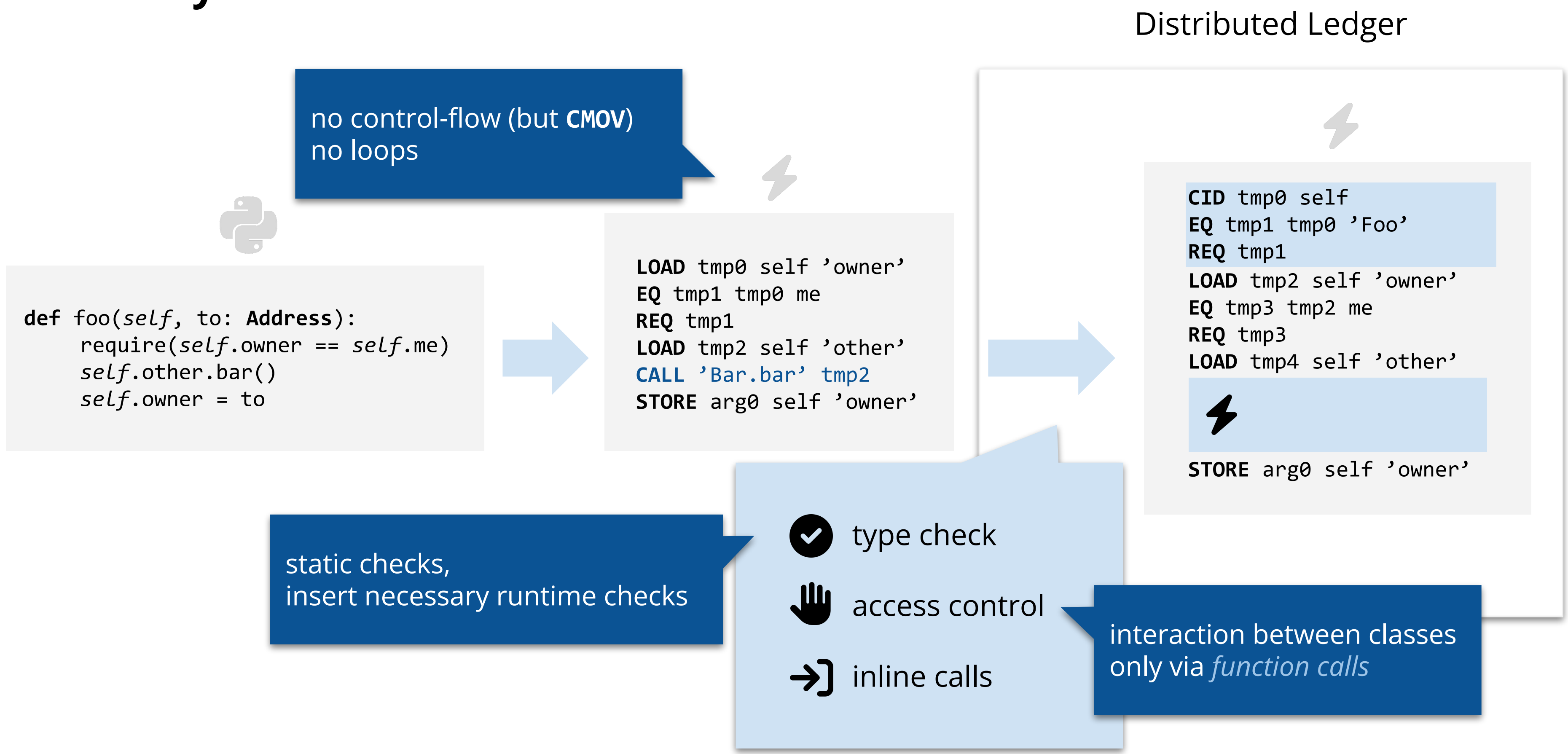
Example: Coin Puzzle

```
class Coin(Contract):
  val: Uint
  # owner: Address

  def transfer(self, to: Address):
    require(self.owner == self.me)
    self.owner = to
```



```
class Wrapper(Contract):
  coin: Coin
  # owner: Address

  def constructor(self, coin: Coin,
                        owner: Address):
    self.owner = owner
    self.coin = coin
    coin.transfer(self.address)

  def puzzle(self, sol: Uint) -> Bool:
    …

  def open(self, sol: Uint):
    require(self.puzzle(sol))
    self.coin.transfer(self.me,
      sender_is_self=True)
```

Assembly Code and Access Control



Distributed Ledger

no control-flow (but **CMOV**)
no loops

```
def foo(self, to: Address):
    require(self.owner == self.me)
    self.other.bar()
    self.owner = to
```

```
LOAD tmp0 self ’owner’
EQ tmp1 tmp0 me
REQ tmp1
LOAD tmp2 self ’other’
CALL ’Bar.bar’ tmp2
STORE arg0 self ’owner’
```

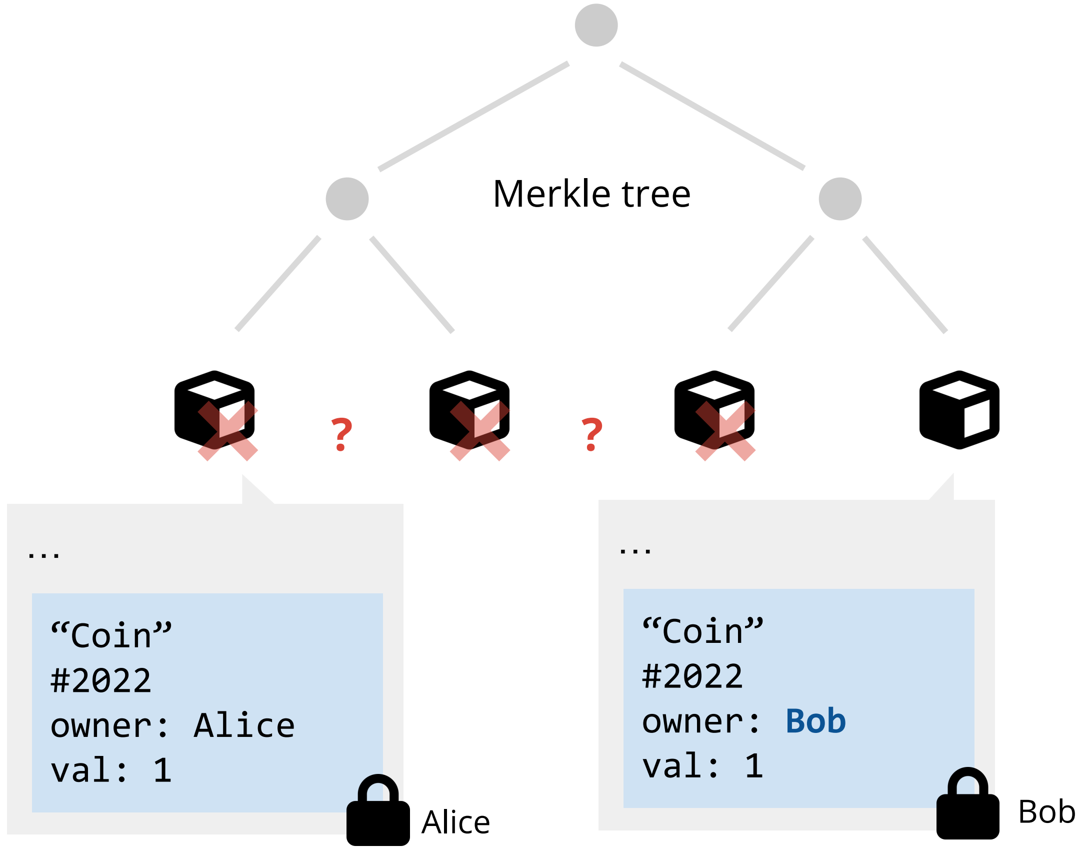
```
CID tmp0 self
EQ tmp1 tmp0 ’Foo’
REQ tmp1
LOAD tmp2 self ’owner’
EQ tmp3 tmp2 me
REQ tmp3
LOAD tmp4 self ’other’
 ⚡
STORE arg0 self ’owner’
```

static checks,
insert necessary runtime checks

- ✓ type check
- ✋ access control
- → inline calls

interaction between classes
only via *function calls*

Storing and Updating Objects



```
…
“Coin”
#2022
owner: Alice
val: 1
```

```
…
“Coin”
#2022
owner: Bob
val: 1
```

generalize Zerocash / Zcash to *objects*

avoid *pitfalls* (very technical)

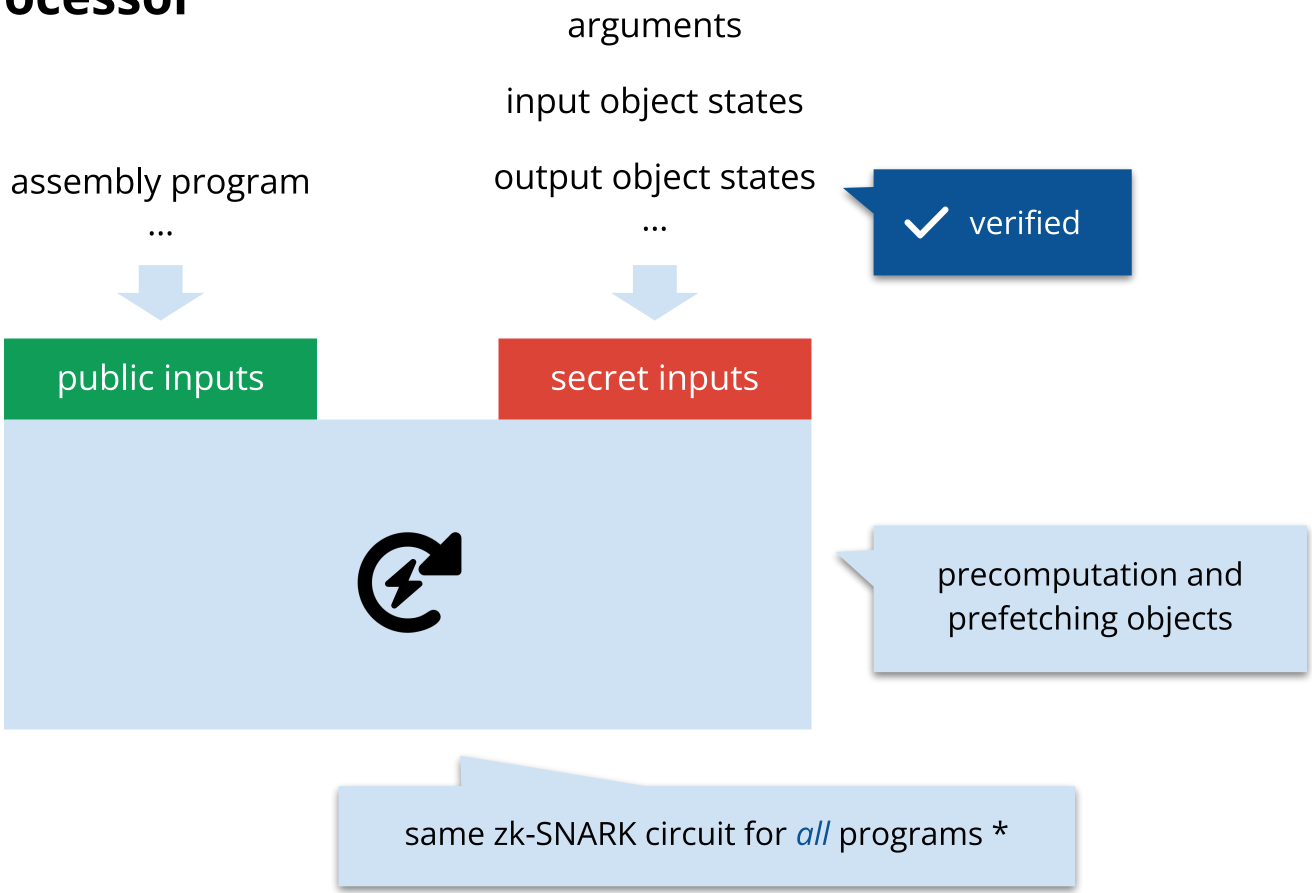
`Coin#2022.transfer(Bob)`

zk-SNARK

I correctly…

- *accessed* the required object states
- *invalidated* these states
- *updated* the states according to the called function
- *encrypted* the new states

Zero-knowledge Processor



assembly program

...

public inputs

arguments

input object states

output object states

...

secret inputs

✓ verified

precomputation and prefetching objects

same zk-SNARK circuit for *all* programs *

Security Properties

simulation-based indistinguishability proof

Data and Identity Privacy

Object accesses, data, sender, and arguments *hidden*

Correctness

Cannot violate class logic

Integrity

Cannot *tamper with* or *replay* transactions

Availability

Cannot *block* valid transactions

identified and fixed
2 attacks on ZEXE

Evaluation

Available on : eth-sri/zapper

on idealized ledger

Expressiveness

Coin

Decentralized Exchange

Private Auction

Double-blind Peer-review

...

Efficiency

On commodity desktop

< **0.01 s** compilation

≈ **22 s** tx generation

99.9 % proof generation

< **0.03 s** tx verification

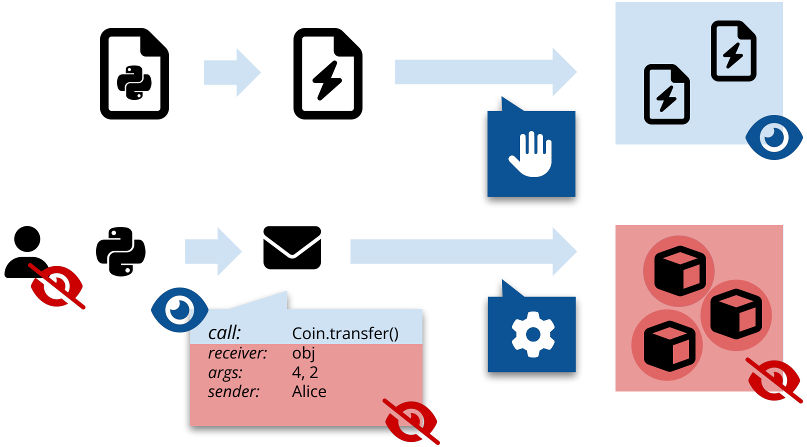
w/o consensus

Summary

eth-sri/zapper

Zapper

```
class Coin(Contract):
  val: Uint
  # owner: Address

  def transfer(self, to: Address):
      require(self.owner == self.me)
      self.owner = to
```



call:	Coin.transfer()
receiver:	obj
args:	4, 2
sender:	Alice