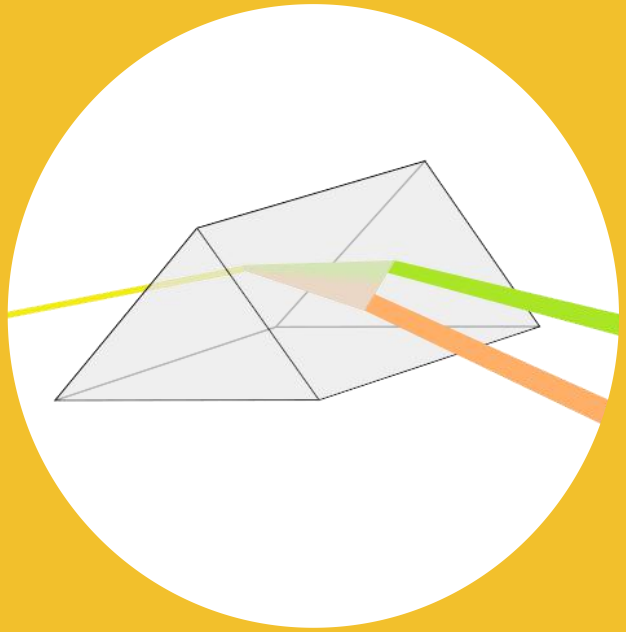
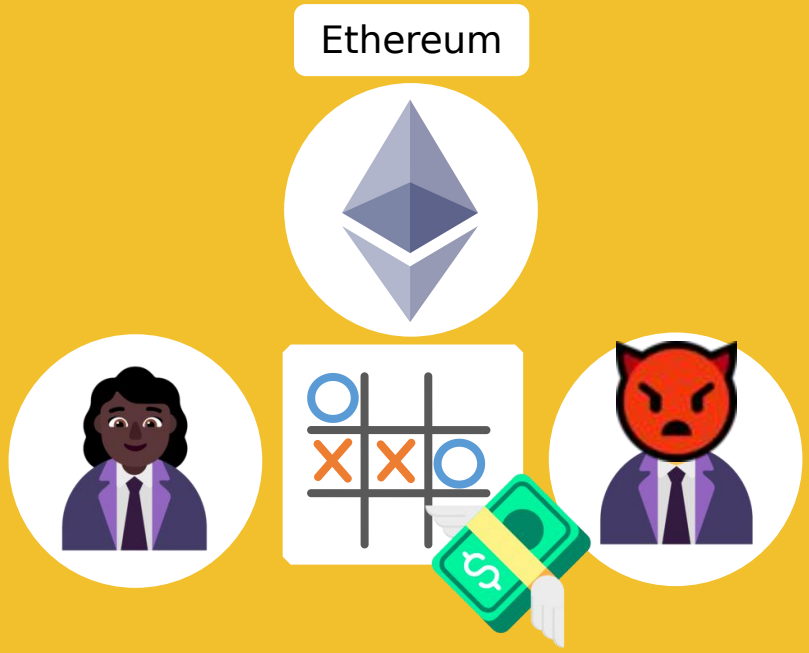
Prisma



A tierless language for
enforcing client-contract protocols
in **decentralized applications**

Richter, Kretzler, Weisenburger,
Salvaneschi, Faust, Mezini

dApps

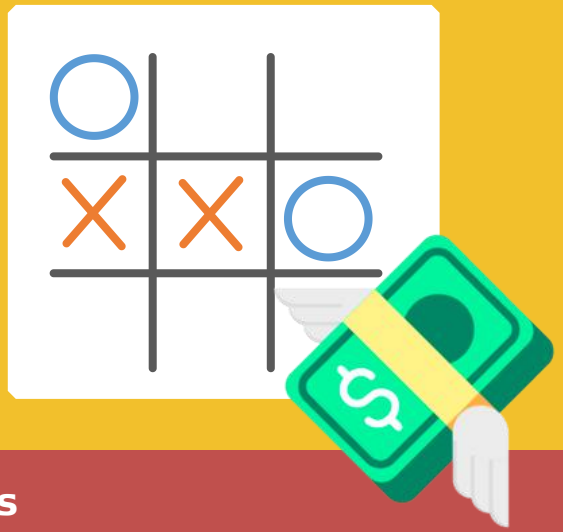


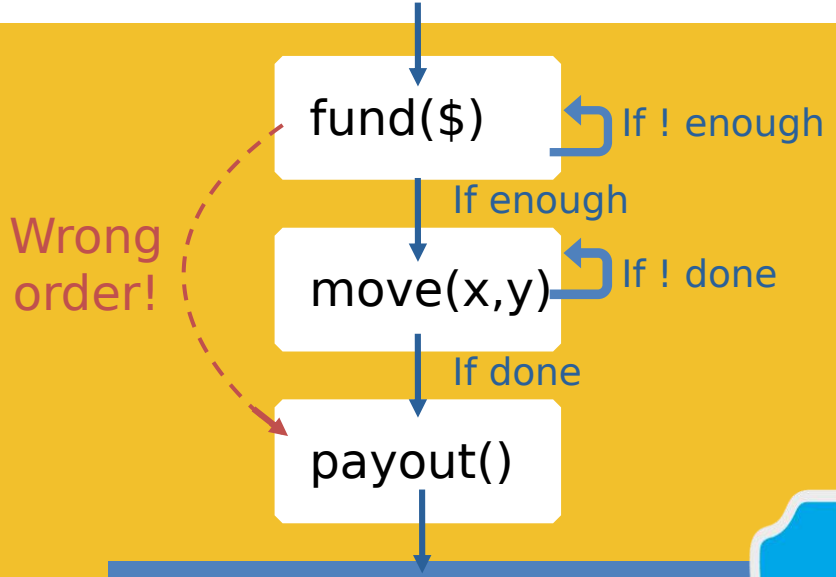
(Smart) Contracts:

– Passive Entity
– React to Messages
– Send/Receive Money

dApp = Contract + Client

Contract





Exploits

– DAO exploits 50 M $

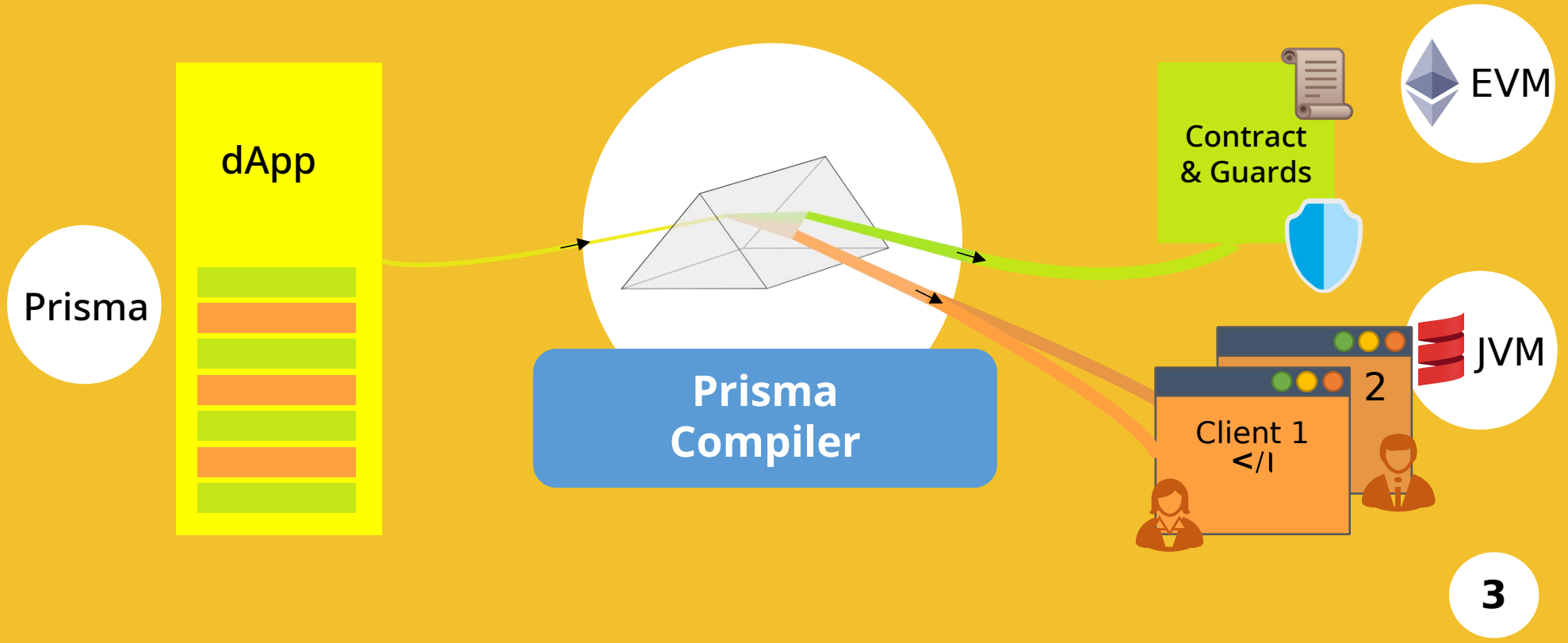
– Parity exploits 30+150 M $

Problem

– concurrency by-default

– need to explicitly forbid wrong interactions

Overview of our Tierless Approach



Enforcing **Control Flow Integrity** with Prisma

Programming Model

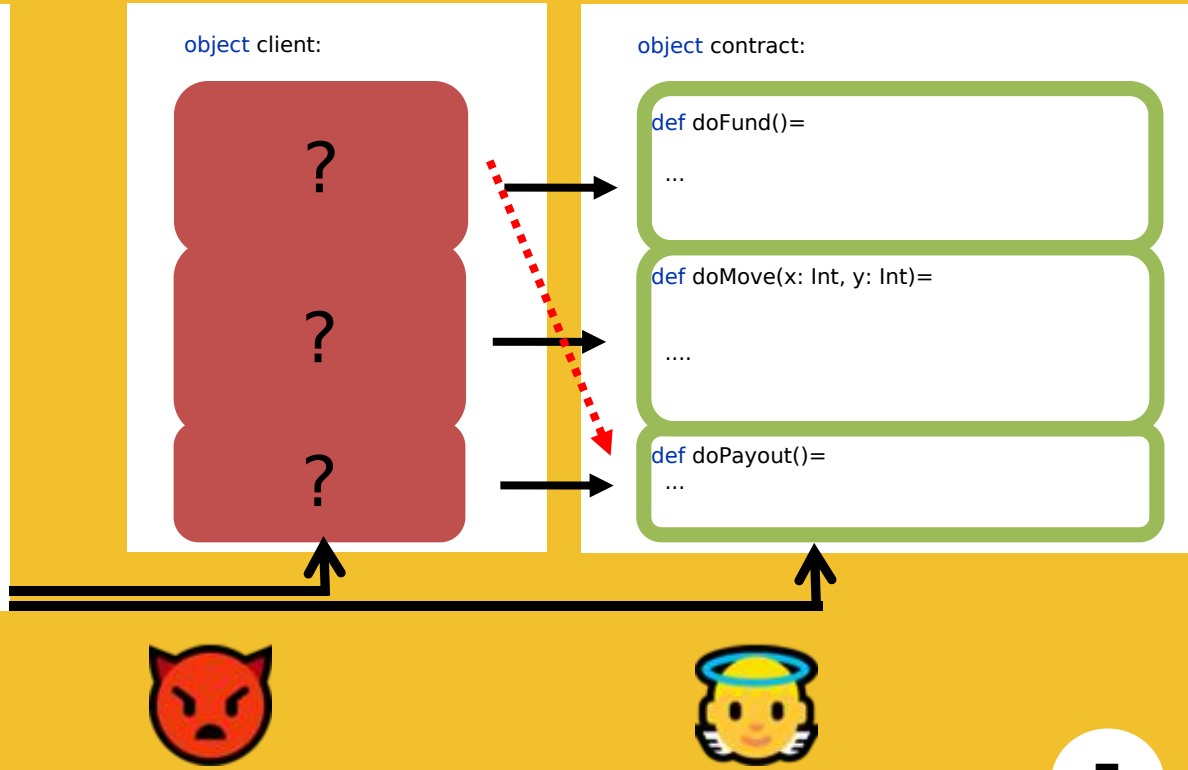
```
@co val init: Unit =
  while balance() < FUNDING_GOAL
  do    client(_ => true){ decideFunds() }

  while moves < 9 && winner == NONE
  do    val pair: UU =
          client(a => a == players(moves % 2))
            { decideMove() }
        move(pair.x, pair.y)

  client(a => true){ ready() }
  if    winner != NONE
  then players(winner).transfer(balance())
  else players(0     ).transfer(balance() / 2)
       players(1     ).transfer(balance())
```

```
object client:
  def decideFunds()=
    ...
  def decideMove()=
    ...
  def ready()=
    ...
```

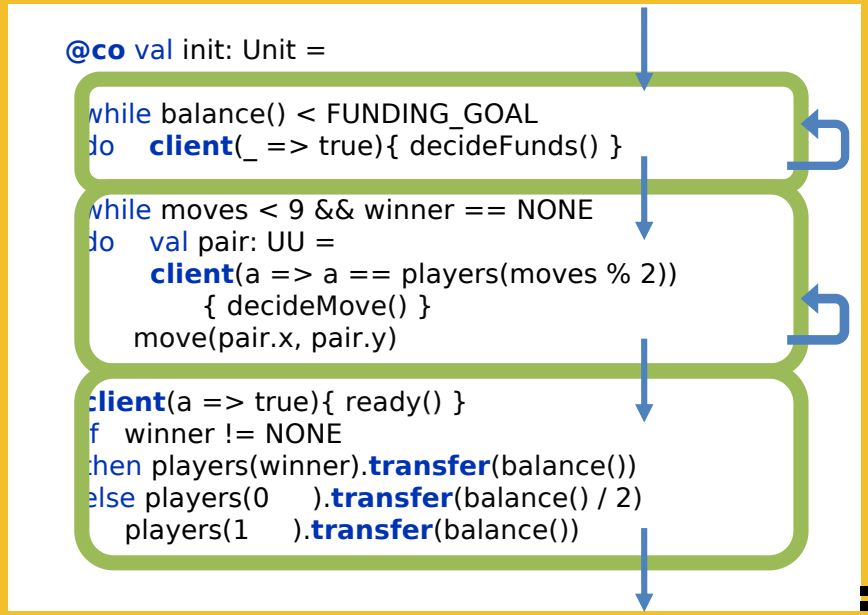
```
object contract:
  def doFund()=
    ...
  def doMove(x: Int, y: Int)=
    ...
  def doPayout()=
    ...
```

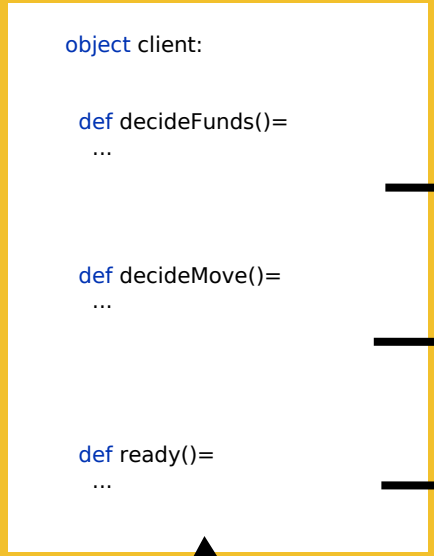
Control Flow Integrity

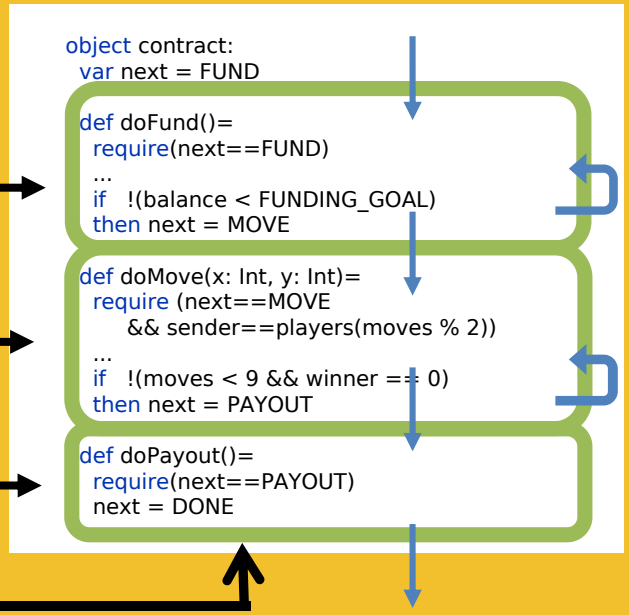
```
@co val init: Unit =
while balance() < FUNDING_GOAL
do    client(_ => true){ dec ?
while moves < 9 && winner == NONE
do    val pair: UU =
        client(a => a == players(moves % 2))
          { decid ?
      move(pair.x, pair.y)
client(a => true){ r ?
f   winner != NONE
hen players(winner).transfer(balance())
lse players(0     ).transfer(balance() / 2)
    players(1     ).transfer(balance())
```



Control Flow Integrity







Formalisation

- **Compilation**
  Source → Target
  Translate every source program into a target program.

- **Secure Compilation**
  not Source → not Target
  Despite the splitting, anything you cannot do with the source, you cannot do with the target either.

Control Flow Integrity (Contract)

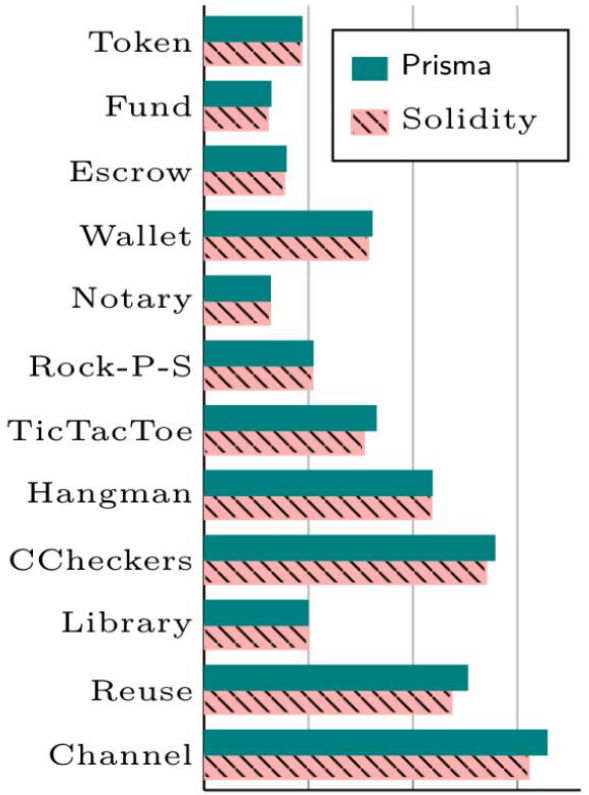
Theorem (Secure Compilation)

$$\forall P. \quad \{\ init_A(comp'(mnf'((P)))) \ \} \approx ... \approx \{\ init_A(P) \ \}$$

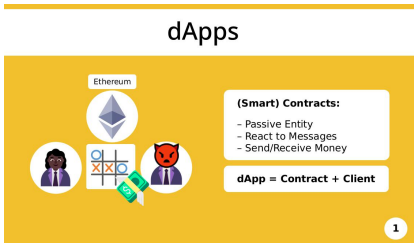
More details in the paper
- syntax & semantics of source & target language
- compilation process
- proof of secure compilation

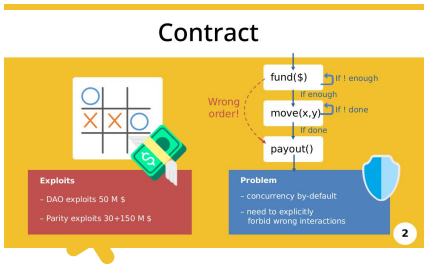
Evaluation

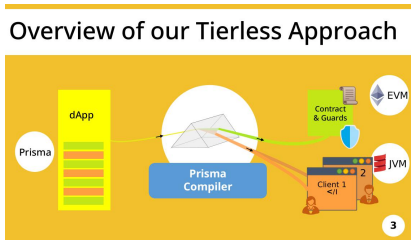
- smart contracts are expensive
- 12 case studies covering common dApp scenarios
- Prisma-generated vs. Solidity-handwritten contracts
- deployment and runtime costs
- overhead is small
- **More details in the paper**

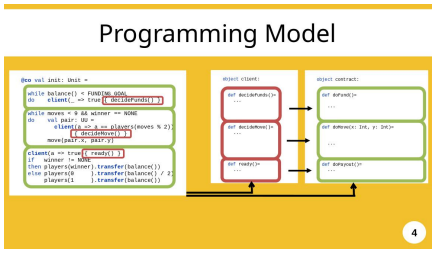


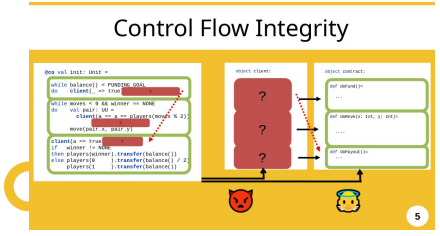
Deployment cost [kGas]

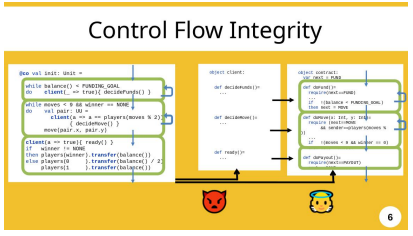


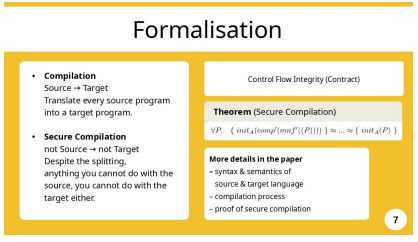


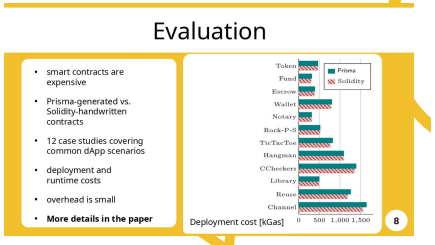












Thanks

- Prisma, a tierless language for dApps
- Compilation & Secure Compilation
- Case Studies & Empirical Evaluation
- Comparision of Prisma with Behavioral Types for Smart Contracts

https://github.com/stg-tud/prisma

NomosR

$$\frac{\Psi; \Gamma, (y{:}A) \vdash P :: (c : B)}{\Psi; \Gamma \vdash (y \leftarrow \mathsf{recv}\ c;\ P) :: (c : A \multimap B)}$$

Prisma

$$\frac{\Gamma \vdash p : Addr \to Bool \qquad \Gamma \vdash b : Ether \times T}{\Gamma \vdash \mathsf{awaitCl}(p)\{\ b\ \} : T}$$