Smart contracts in a bare-bone UTXO model

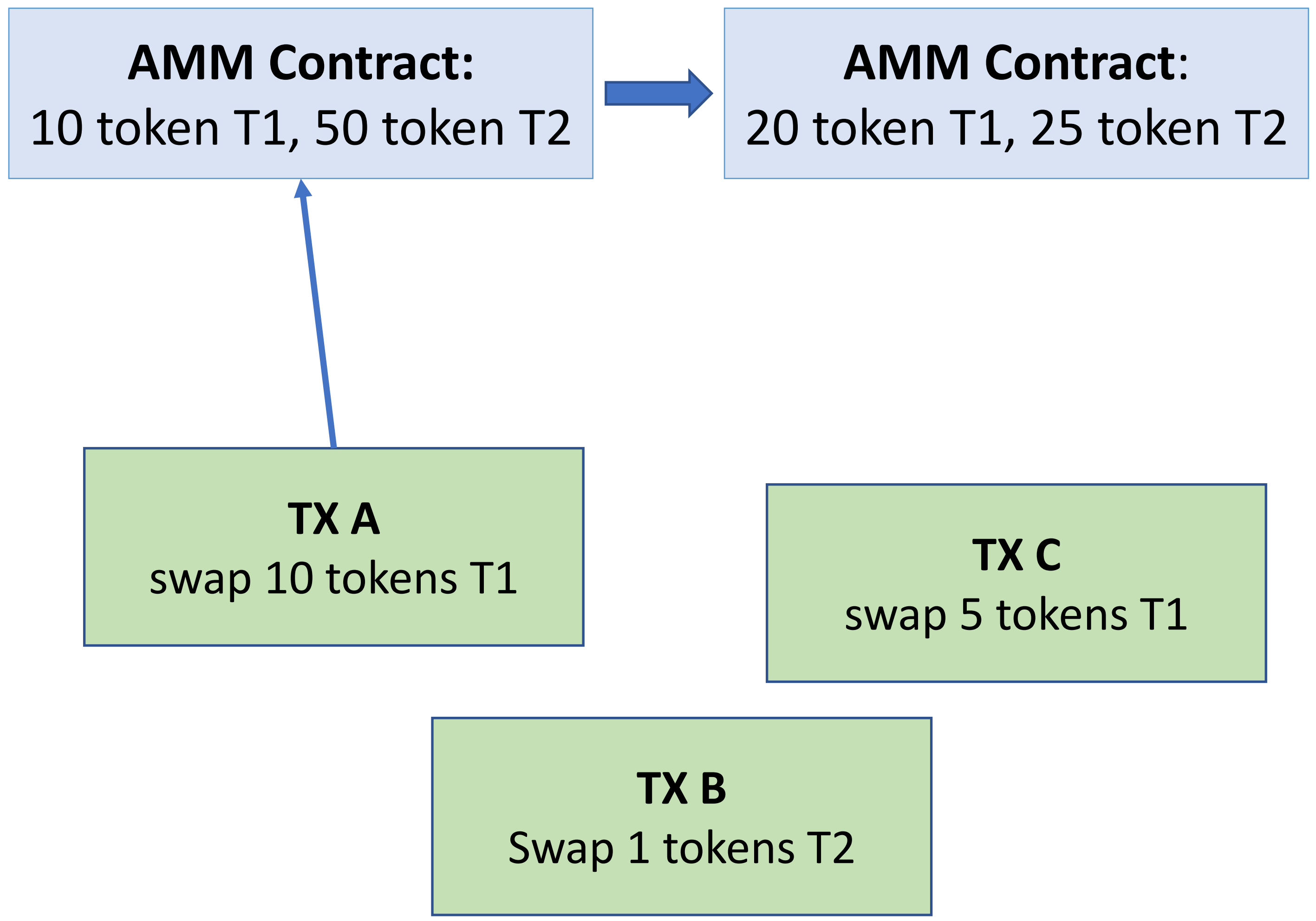
Massimo Bartoletti
Università di Cagliari

Riccardo Marchesin
Università di Trento

Roberto Zunino
Università di Trento

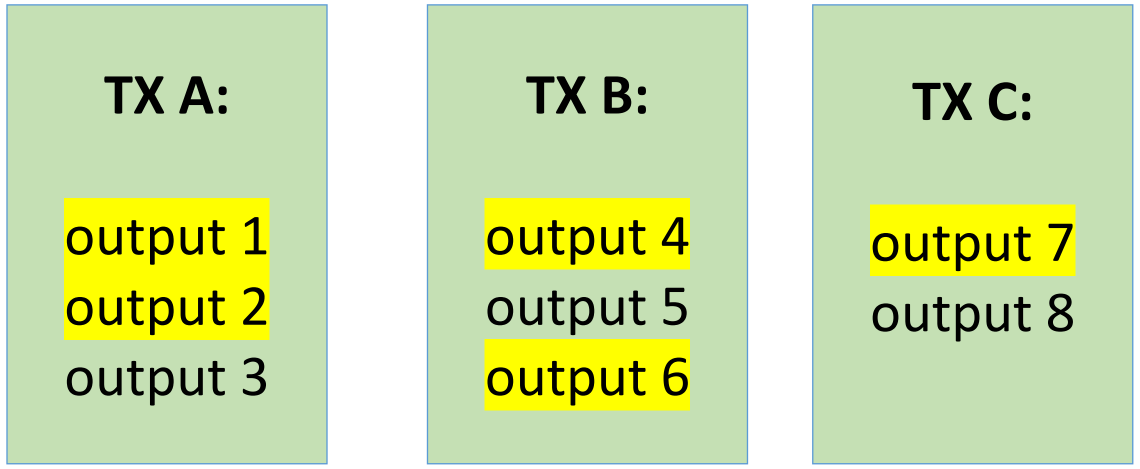
Account-based model

- E.g. Ethereum.
- Enables a familiar programming style.
- Users can’t know in which state their transaction is executed.
    - Transaction reordering attacks
    - Difficult to parallelize

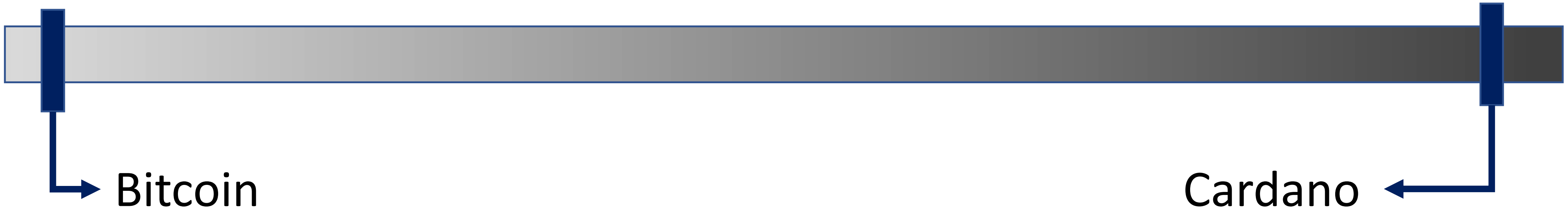


UTXO model

- E.g. Bitcoin, Cardano.
- Contract state is scattered across tx outputs.
- To execute you must specify which outputs are being redeemed -> full knowledge of the state.
    - Less susceptible to reordering attacks.
    - Easily parallelizable.



Different UTXO models



Bitcoin

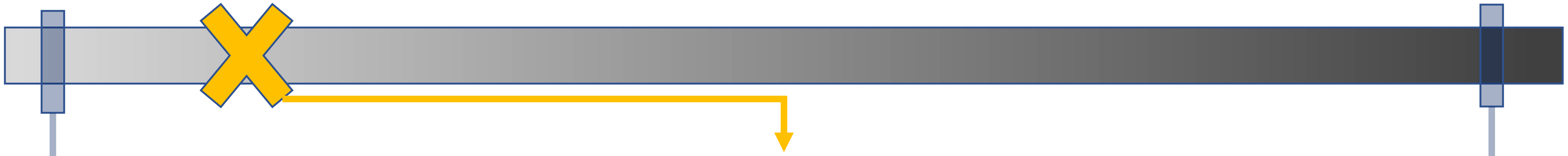
- Restricted scripting language -> limited expressiveness: contracts always terminate
- No gas mechanism.

Cardano

- Scripting language is an untyped lambda calculus –> expressive contracts.
- Gas mechanism.

The further on the left, the easier it is to implement formal verification methods

Different UTXO models



Bitcoin

- Restricted scripting language -> limited expressiveness: contracts always terminate
- No gas mechanism

Our model

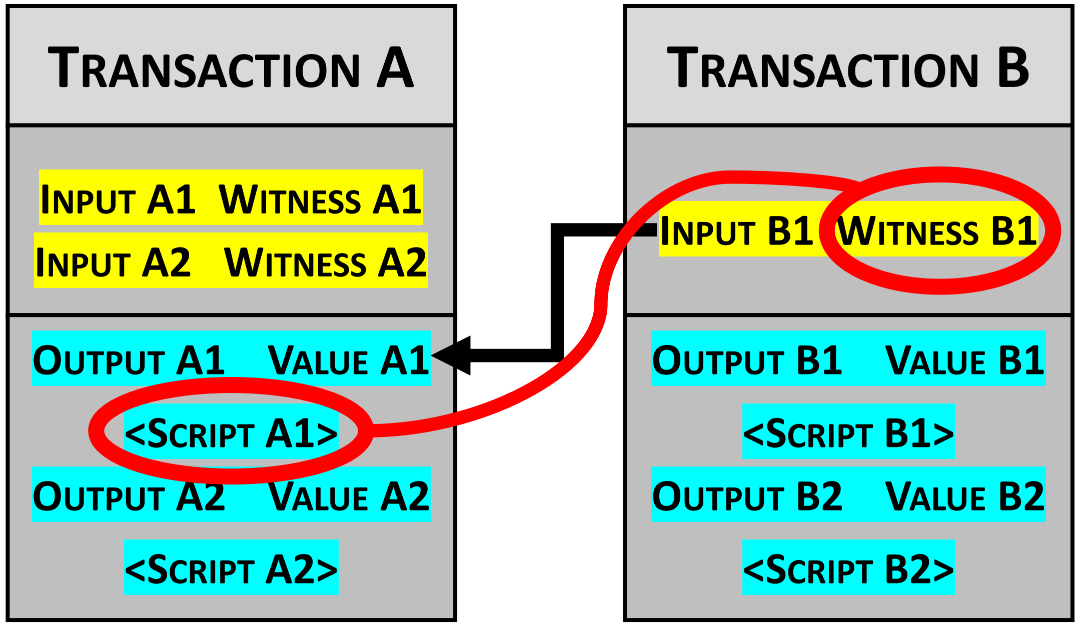
- Bitcoin-like scripting language extended with covenants.

Cardano

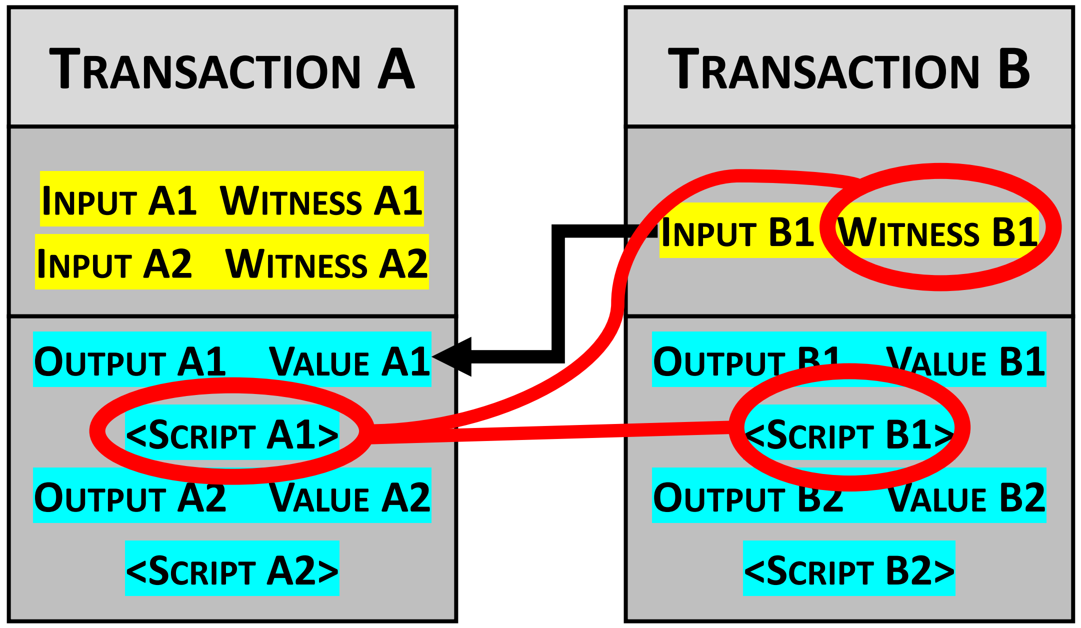
- Scripting language is an untyped lambda calculus -> expressive contracts
- Gas mechanism.

Covenants

Covenants are a set of primitives that allow a transaction script to "look into the future" and access the output field of the redeeming transaction

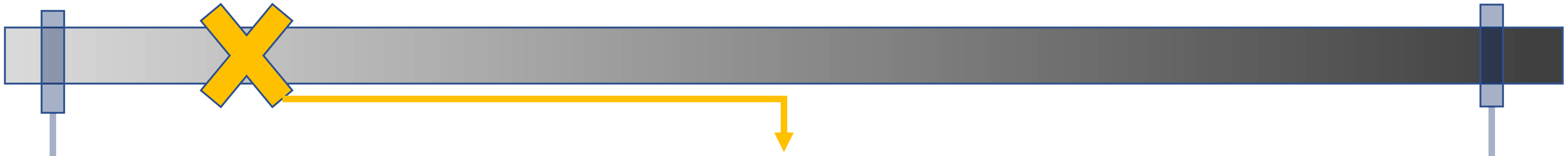


Bitcoin



Bitcoin + covenants

Different UTXO models



Bitcoin

- Restricted scripting language.
- Limited expressiveness: contracts always terminate

Our model

- Bitcoin-like scripting language extended with covenants.
- Scripting language is not Turing complete, but contracts are.
- No gas mechanism

Cardano

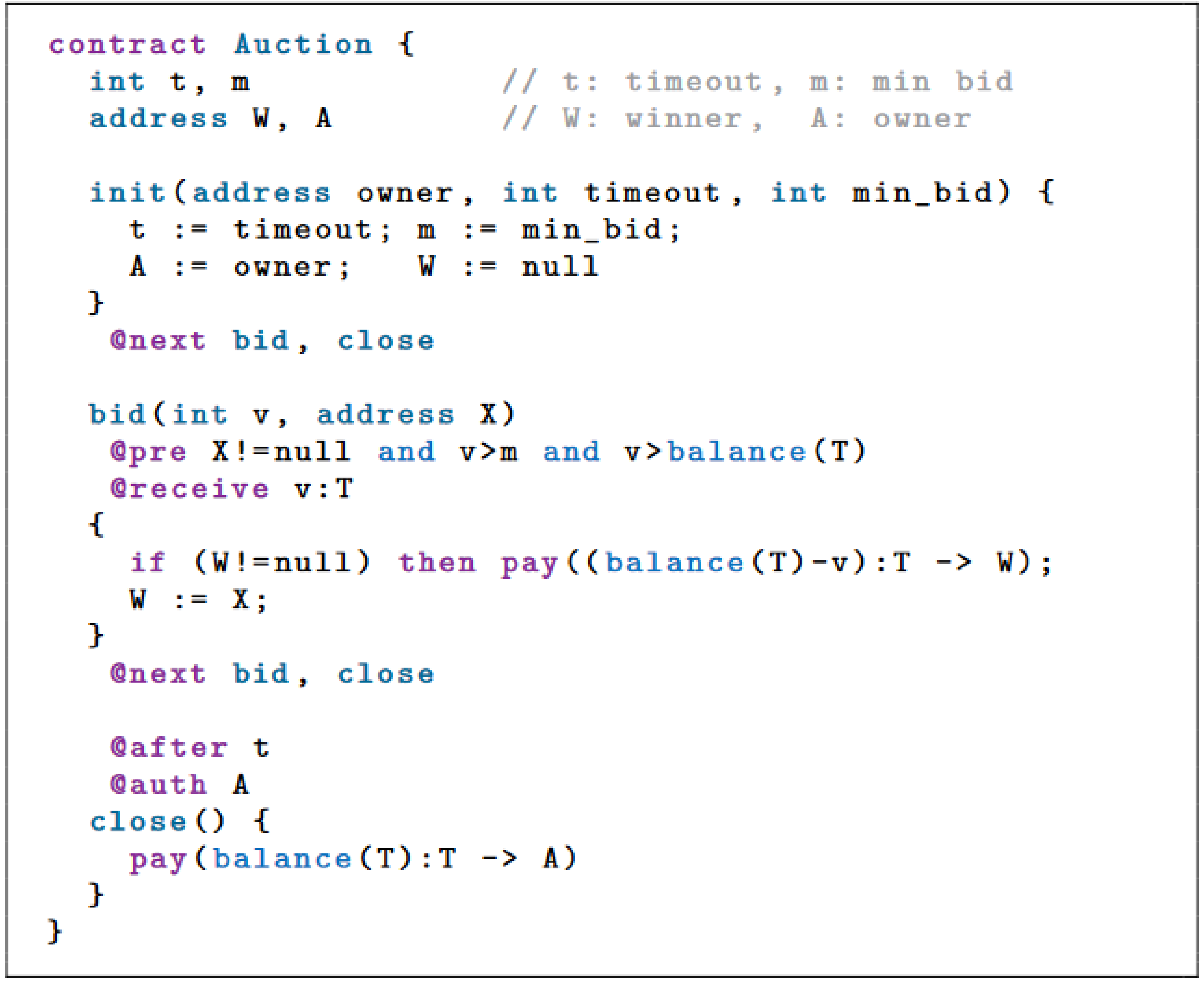
- The scripting language is an untyped lambda calculus
- Expressive contracts (Turing complete)

Our contract language

Solidity-like imperative language that compiles to UTXO.

Compilation exploits covenants to preserve contract script.

More complex examples: AMM, …



```
contract Auction {
  int t, m              // t: timeout, m: min bid
  address W, A          // W: winner,  A: owner

  init(address owner, int timeout, int min_bid) {
    t := timeout; m := min_bid;
    A := owner;    W := null
  }
   @next bid, close

  bid(int v, address X)
   @pre X!=null and v>m and v>balance(T)
   @receive v:T
  {
    if (W!=null) then pay((balance(T)-v):T -> W);
    W := X;
  }
   @next bid, close

   @after t
   @auth A
  close() {
    pay(balance(T):T -> A)
  }
}
```

Security of the compiler

Two levels of abstraction:

- **Symbolic** level: Formal contracts semantics.
- **Computational** level: UTXO blockchain with covenants.

Symbolic to computational compiler.

Computational soundness: symbolic security implies computational security.

Full paper

Secure compilation of rich smart contracts on poor UTXO blockchains: https://arxiv.org/abs/2305.09545