Marlowe

Smart Contracts

Marlowe

A SPECIAL-PURPOSE LANGUAGE FOR FINANCIAL CONTRACTS

Designed for users, as well as developers.

Designed for maximum assurance.

Assurance

CONTRACTS DO WHAT THEY SHOULD ...
... AND NOT WHAT THEY SHOULDN'T

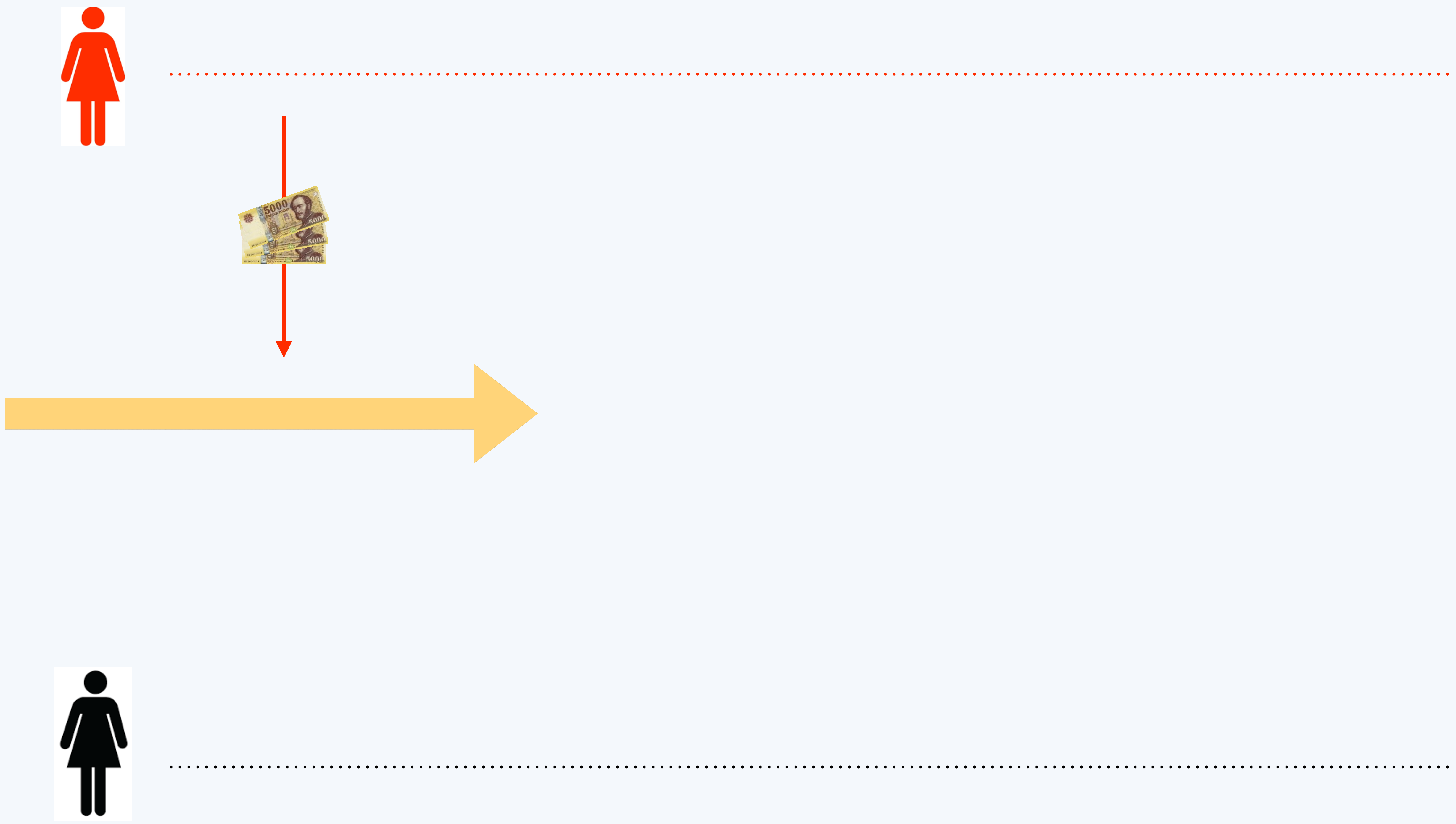
Language as *simple* as it can be.

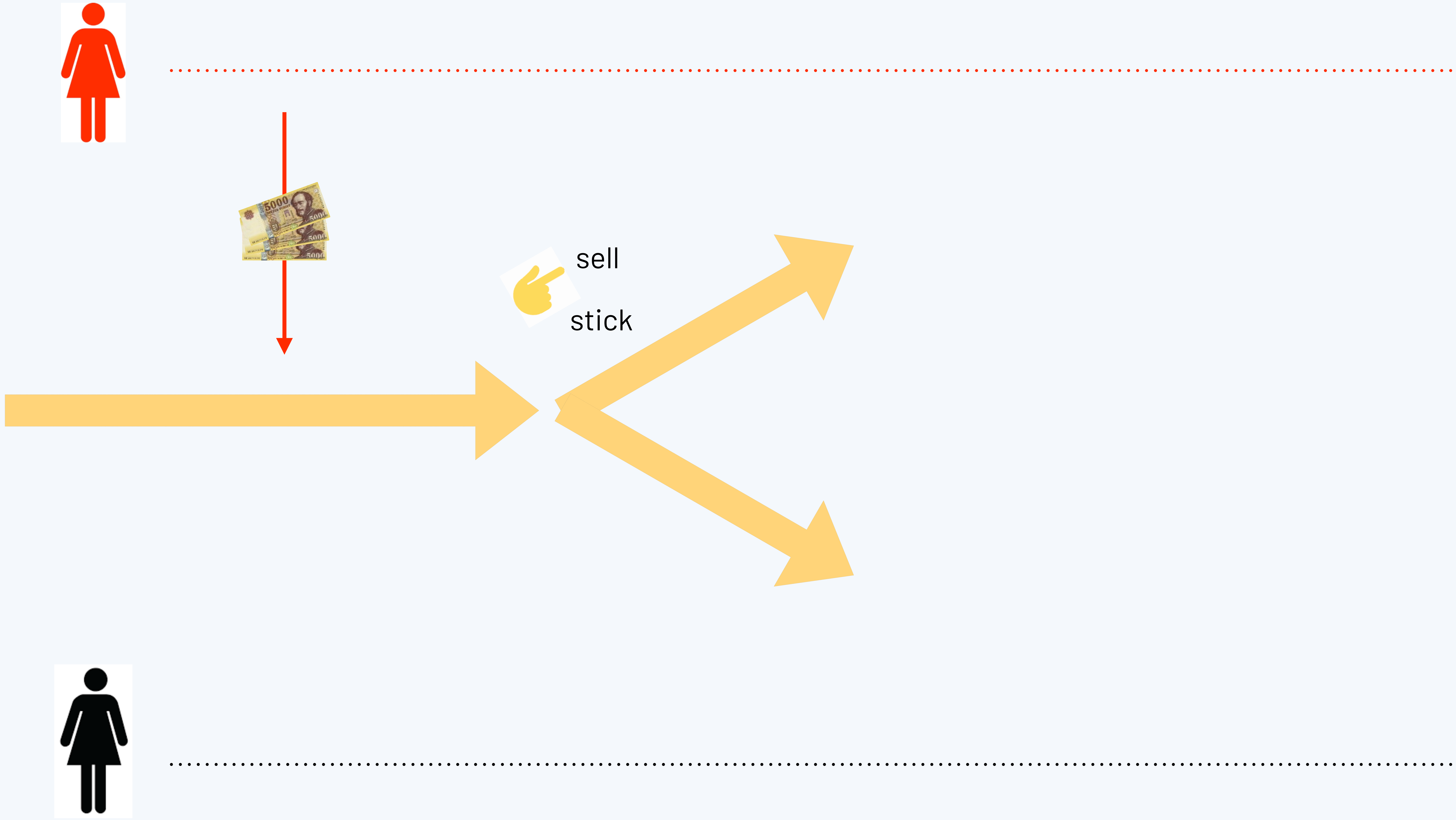
Contracts can be *read* and *simulated*.

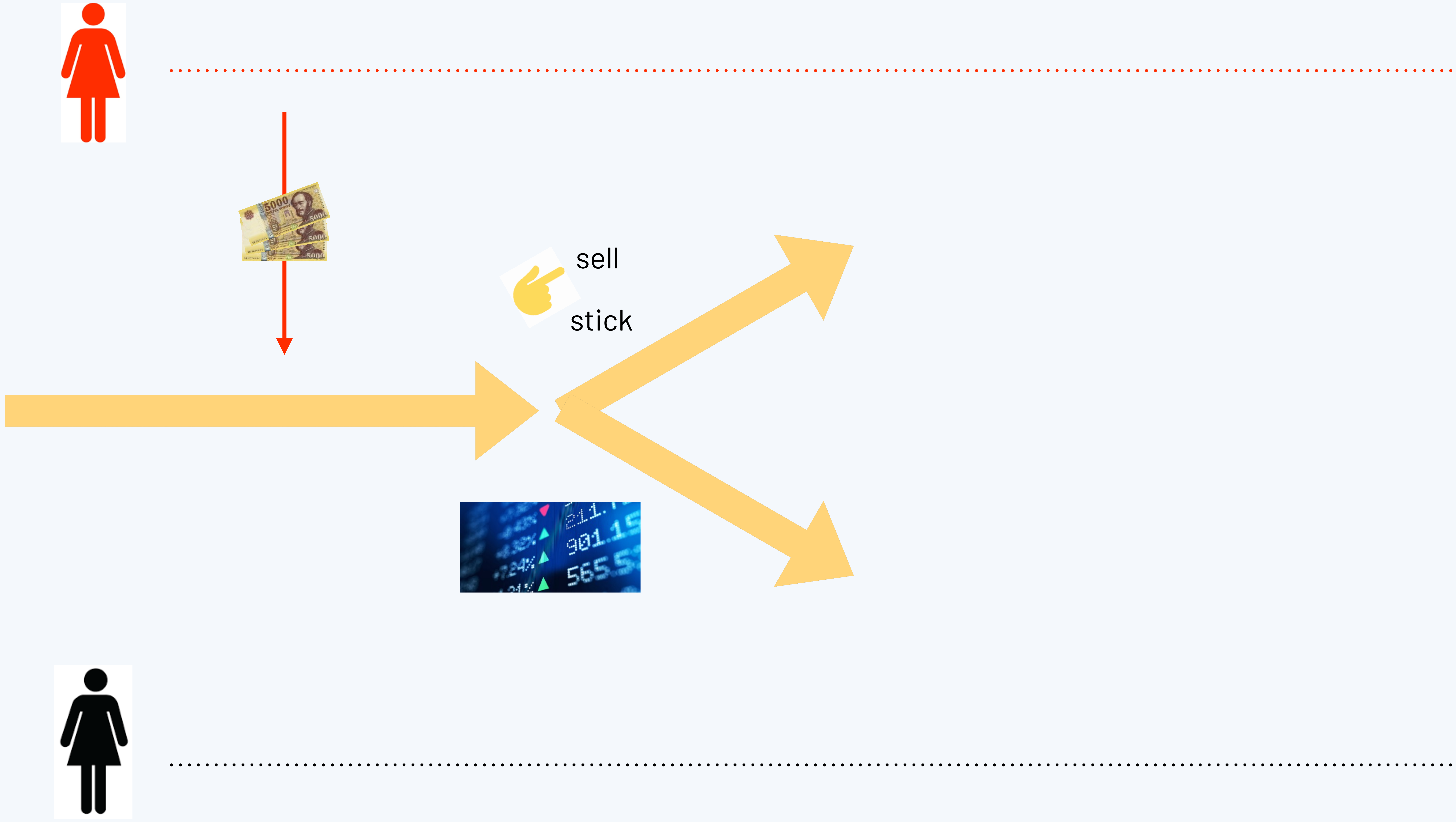
Before running, can explore *all behaviour.*

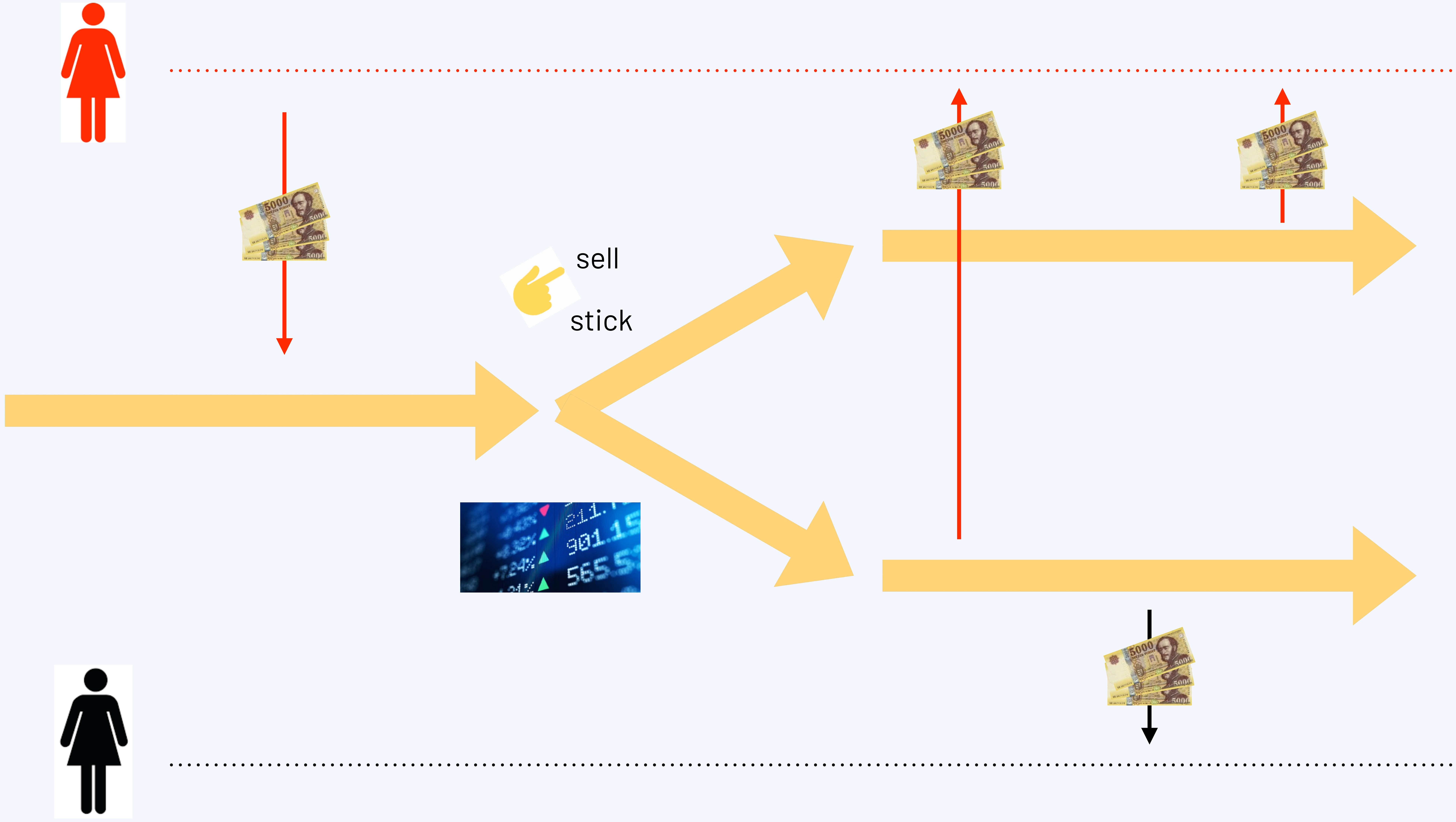
System can be *proved safe* in various ways.

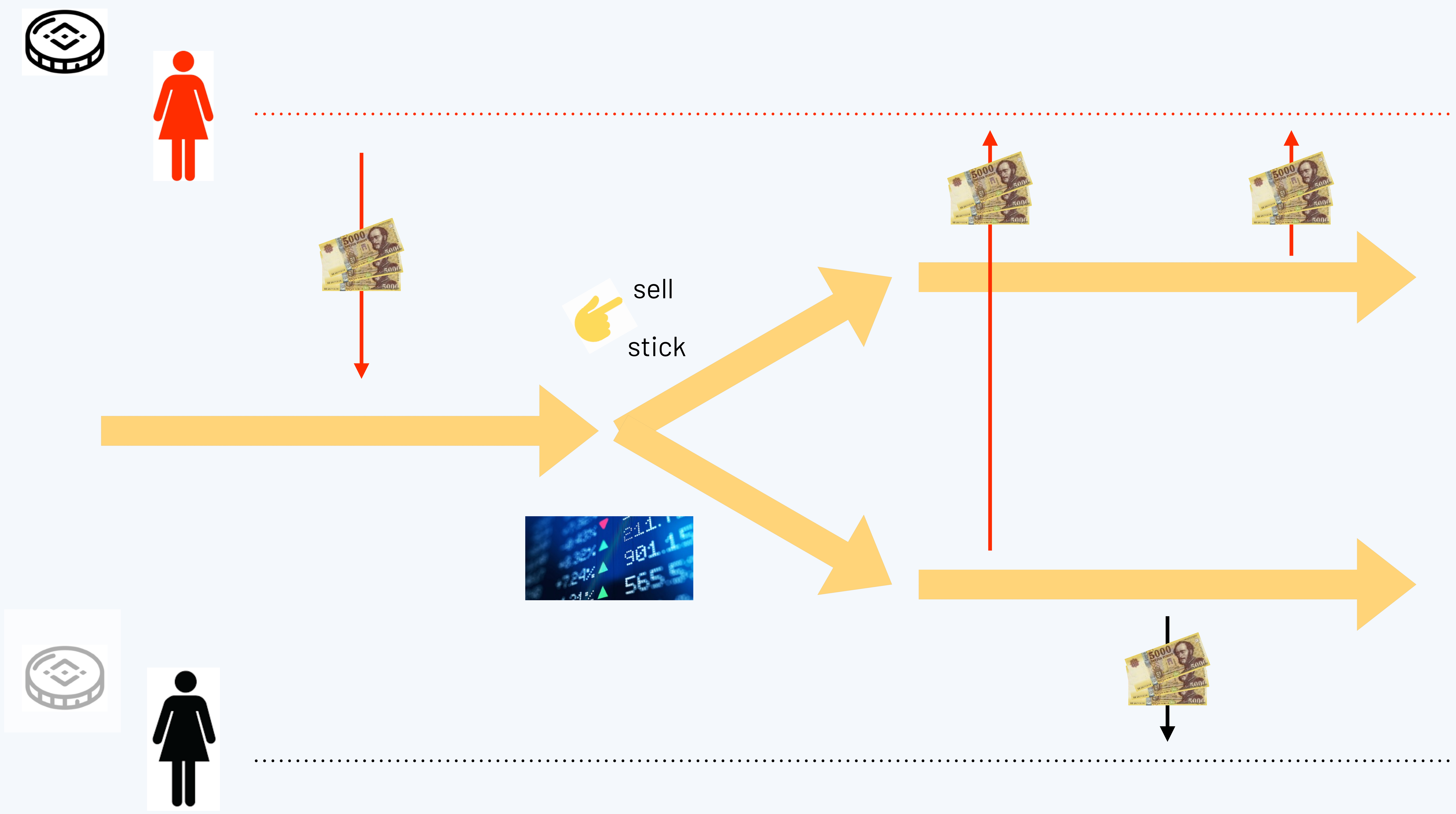
What does a financial contract do?


5000


sell
stick


sell
stick


sell
stick
211.1
901.15
565.5


sell
stick
211.
901.15
565.5

Design

A contract could …

A CONTRACT IS JUST A PROGRAM RUNNING ON A BLOCKCHAIN

… run forever.

… wait for an input forever.

… terminate holding assets.

… “double spend” assets.

Designed for safety

Contracts are finite.

No recursion or loops (in Marlowe).

Designed for safety

Contracts are finite.

No recursion or loops (in Marlowe).

Contracts will terminate …

Timeouts on actions: choice, deposit, …

Designed for safety

Contracts are finite.

No recursion or loops (in Marlowe).

Contracts will terminate ...

Timeouts on actions: choice, deposit, ...

... with a defined lifetime.

Read off from timeouts.

Designed for safety

Contracts are finite.	No recursion or loops (in Marlowe).
Contracts will terminate ...	Timeouts on actions: choice, deposit, ...
... with a defined lifetime.	Read off from timeouts.
No assets retained on close.	(Local) accounts refunded on close.

Designed for safety

Contracts are finite.	No recursion or loops (in Marlowe).
Contracts will terminate …	Timeouts on actions: choice, deposit, …
… with a defined lifetime.	Read off from timeouts.
No assets retained on close.	(Local) accounts refunded on close.
Conservation of value.	Underlying blockchain

Designed for safety

Contracts are finite.	No recursion or loops (in Marlowe).
Contracts will terminate ...	Timeouts on actions: choice, deposit, ...
... with a defined lifetime.	Read off from timeouts.
No assets retained on close.	(Local) accounts refunded on close.
Conservation of value.	Underlying blockchain

The Marlowe language

Contracts are finite.

Contracts will terminate ...

... with a defined lifetime.

No assets retained on close.

Conservation of value.

```
data Contract = Close
              | Pay Party Payee Value Contract
              | If Observation Contract Contract
              | When [Case Action Contract]
                     Timeout Contract
              | Let ValueId Value Contract
              | Assert Observation Contract
```

The Marlowe language

Contracts are finite.

Contracts will terminate ...

... with a defined lifetime.

No assets retained on close.

Conservation of value.

```
data Contract = Clo
              | Pay Party Payee Value Contract
              | If Observation Contract Contract
              | When [Case Action Contract]
                     Timeout Contract
              | Let ValueId Value Contract
              | Assert Observation Contract
```

The Marlowe language

Contracts are finite.

Contracts will terminate ...

... with a defined lifetime.

No assets retained on close.

Conservation of value.

```
data Contract = Close
              | Pay Party Payee Value Contract
              | If Observation Contract Contract
              | When [Case Action Contract]
                     Timeout Contract
              | Let ValueId Value Contract
              | Assert Observation Contract
```

The Marlowe language

Contracts are finite.

Contracts will terminate ...

... with a defined lifetime.

No assets retained on close.

Conservation of value.

```
data Contract = Close
              | Pay Party Payee Value Contract
              | If Observation Contract Contract
              | When [Case Action Contract]
                     Timeout Contract
              | Let ValueId Value Contract
              | Assert Observation Contract
```

The Marlowe language

Contracts are finite.

Contracts will terminate …

… with a defined lifetime.

No assets retained on close.

Conservation of value.

```
data Contract = Close
              | Pay Party Payee Value Contract
              | If Observation Contract Contract
              | When [Case Action Contract]
                     Timeout Contract
              | Let ValueId Value Contract
              | Assert Observation Contract
```

The Marlowe language

Contracts are finite.

Contracts will terminate …

… with a defined lifetime.

No assets retained on close.

Conservation of value.

```
data Contract = Close
              | Pay Party Payee Value Contract
              | If Observation Contract Contract
              | When [Case Action Contract]
                     Timeout Contract
              | Let ValueId Value Contract
              | Assert Observation Contract
```

The Marlowe language

Contracts are finite.

Contracts will terminate …

… with a defined lifetime.

No assets retained on close.

Conservation of value.

```
data Contract = Close
              | Pay Party Payee Value Contract
              | If Observation Contract Contract
              | When [Case Action Contract]
                     Timeout Contract
              | Let ValueId Value Contract
              | Assert Observation Contract
```

The Marlowe language

Contracts are finite.

Contracts will terminate ...

... with a defined lifetime.

No assets retained on close.

Conservation of value.

```
data Contract = Close
              | Pay Party Payee Value Contract
              | If Observation Contract Contract
              | When [Case Action Contract]
                     Timeout Contract
              | Let ValueId Value Contract
              | Assert Observation Contract
```

The Marlowe language

Contracts are finite.

Contracts will terminate …

… with a defined lifetime.

No assets retained on close.

Conservation of value.

```
data Contract = Close
              | Pay Party Payee Value Contract
              | If Observation Contract Contract
              | When [Case Action Contract]
                     Timeout Contract
              | Let ValueId Value Contract
              | Assert Observation Contract
```

The Marlowe language

Contracts are finite.

Contracts will terminate ...

... with a defined lifetime.

No assets retained on close.

Conservation of value.

```
data Contract = Close
              | Pay Party Payee Value Contract
              | If Observation Contract Contract
              | When [Case Action Contract]
                     Timeout Contract
              | Let ValueId Value Contract
              | Assert Observation Contract
```

The Marlowe language

Contracts are finite.

Contracts will terminate …

… with a defined lifetime.

No assets retained on close.

Conservation of value.

```
data Contract = Close
              | Pay Party Payee Value Contract
              | If Observation Contract Contract
              | When [Case Action Contract]
                     Timeout Contract
              | Let ValueId Value Contract
              | Assert Observation Contract
```

The Marlowe language

Contracts are finite.

Contracts will terminate ...

... with a defined lifetime.

No assets retained on close.

Conservation of value.

```
data Contract = Close
              | Pay Party Payee Value Contract
              | If Observation Contract Contract
              | When [Case Action Contract]
                     Timeout Contract
              | Let ValueId Value Contract
              | Assert Observation Contract
```

Product


Marlowe Suite
marlowe-finance.io
Run
Market
Play
Build

End users:
obtain and
run contracts
distributed

End users: obtain and run contracts distributed

Contracts up and down loaded, with assurances

Marlowe Suite

marlowe-finance.io

Run

Market

Play

Build

Run: End users: obtain and run contracts distributed

Market: Contracts up and down loaded, with assurances

Play: Contracts can be simulated interactively

Run	Market	Play	Build
End users: obtain and run contracts distributed	Contracts up and down loaded, with assurances	Contracts can be simulated interactively	Contracts built in code, visually, and embedded

Currently combined in the Marlowe Playground

End users: obtain and run contracts distributed

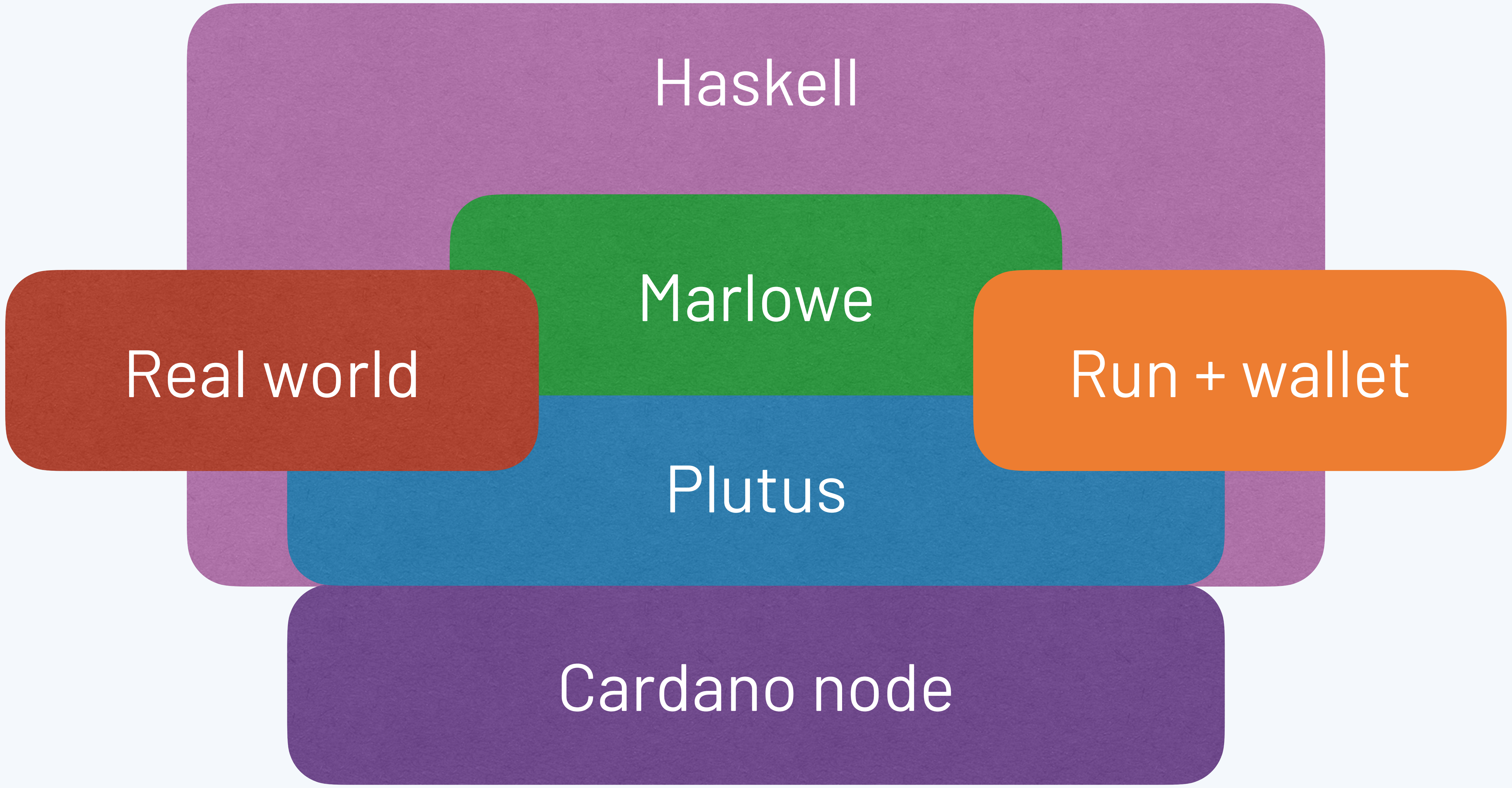
Contracts up and down loaded, with assurances

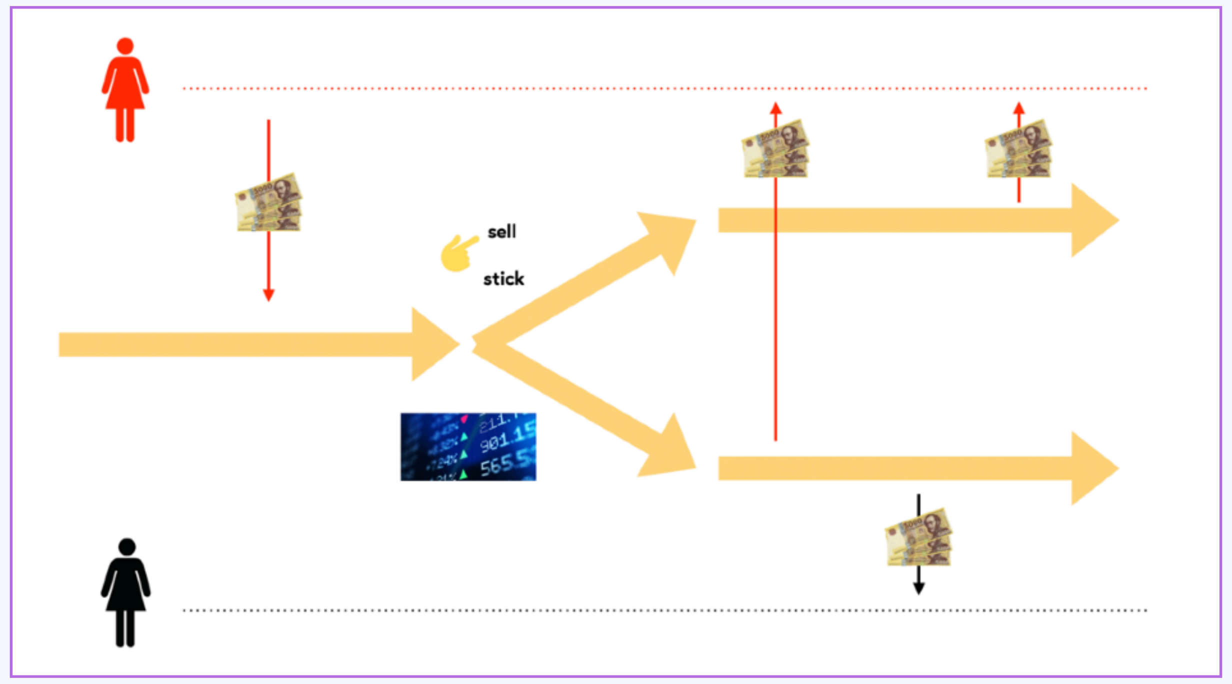
Contracts can be simulated interactively

Contracts built in code, visually, and embedded

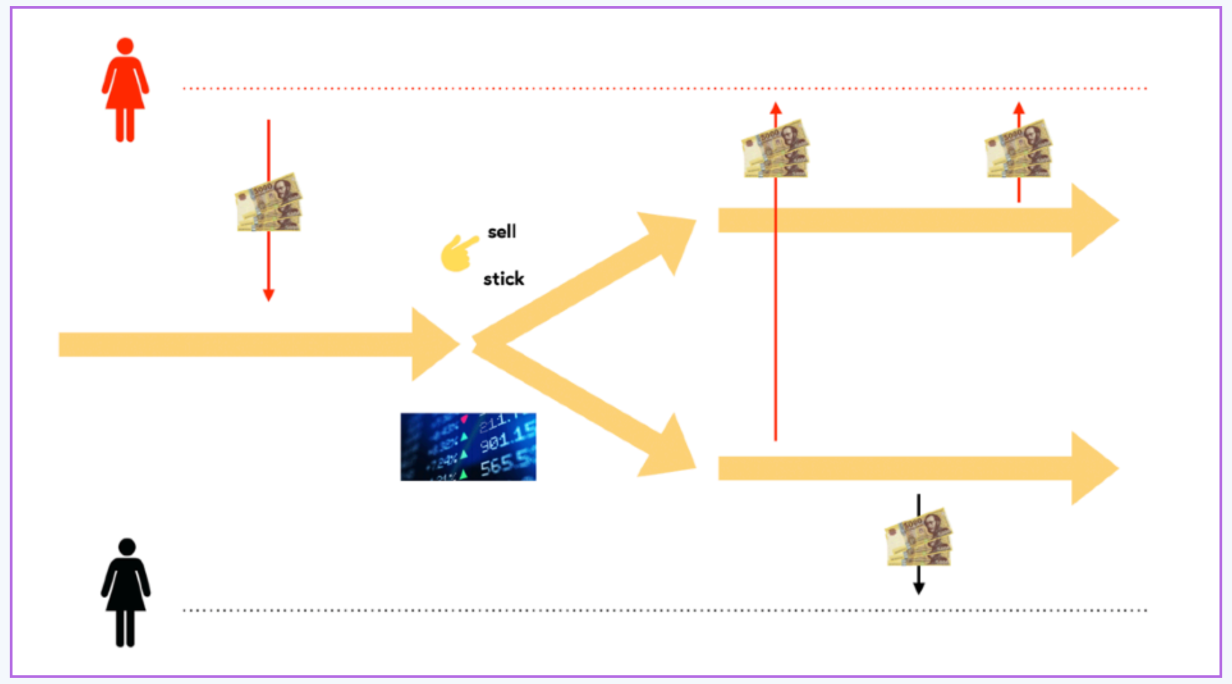
Currently combined in the Marlowe Playground

Engineering

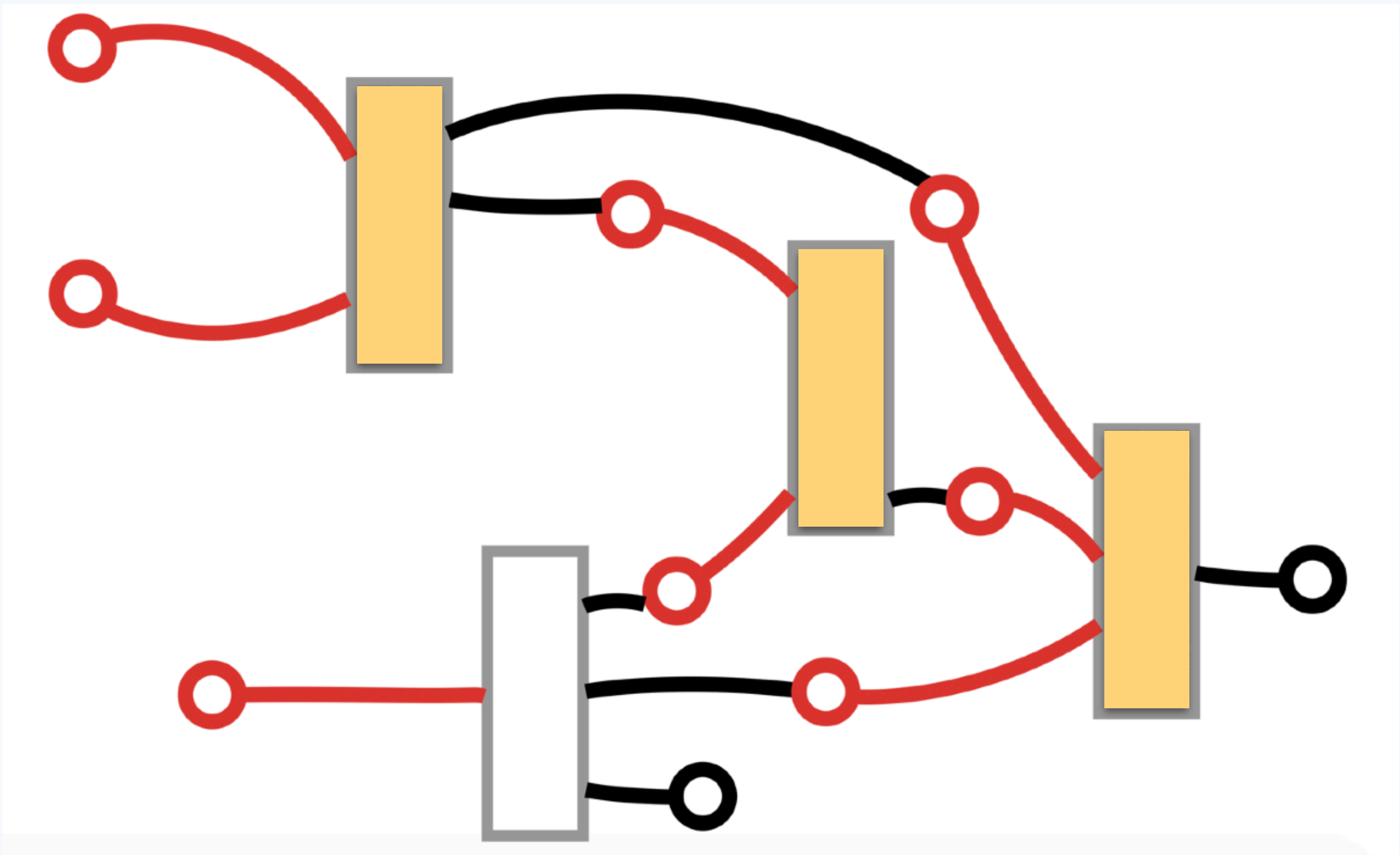

Haskell
Marlowe
Real world
Run + wallet
Plutus
Cardano node

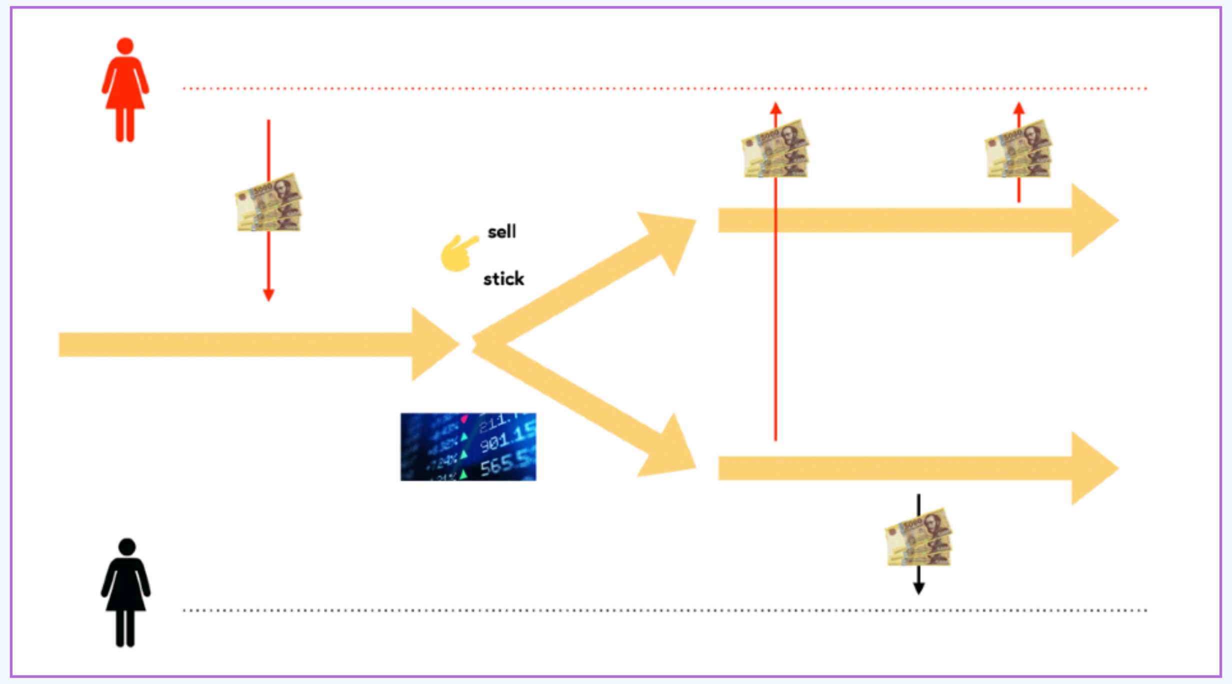


sell
stick



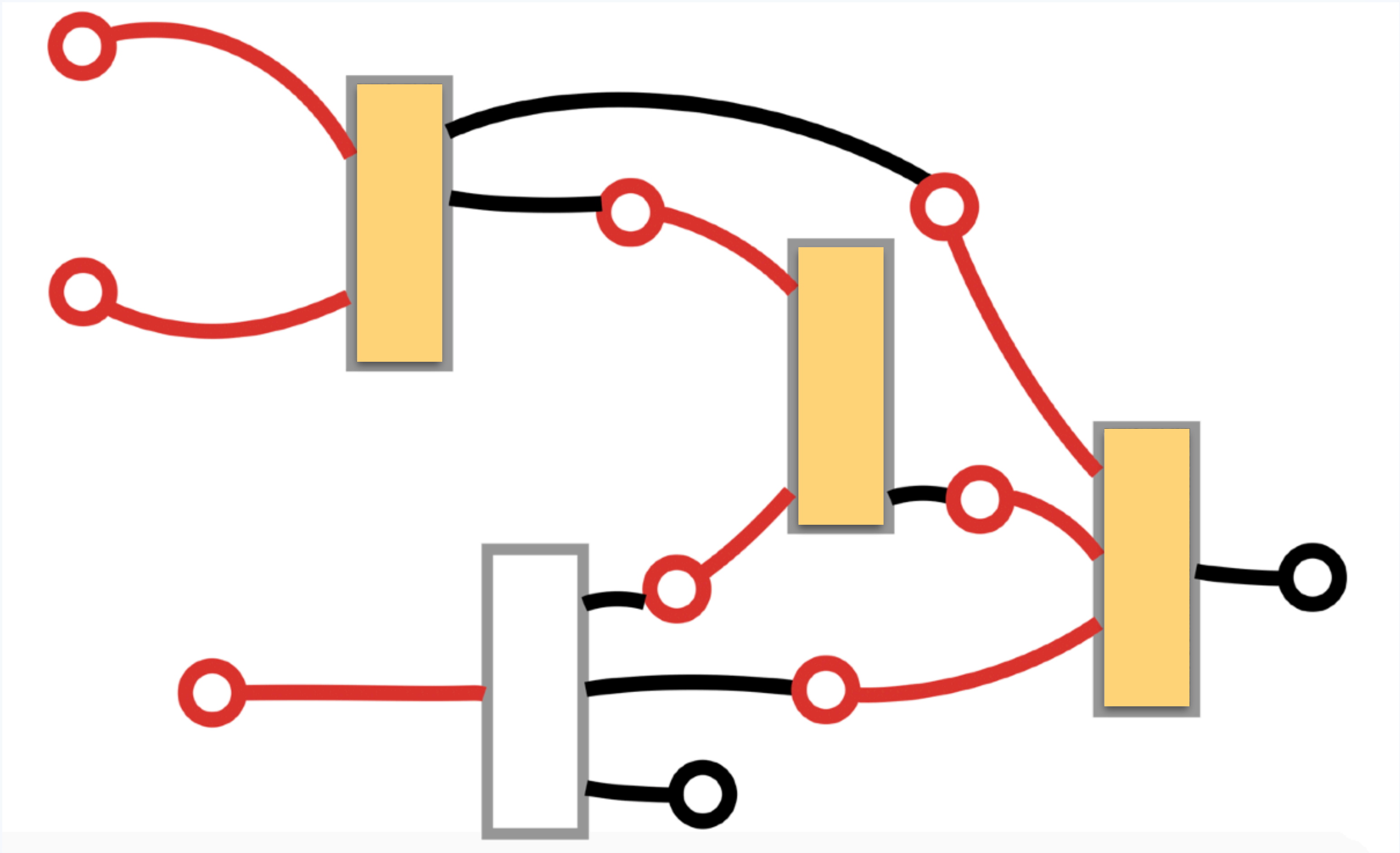
Validation is through
the Marlowe interpreter,
i.e. a Plutus contract.

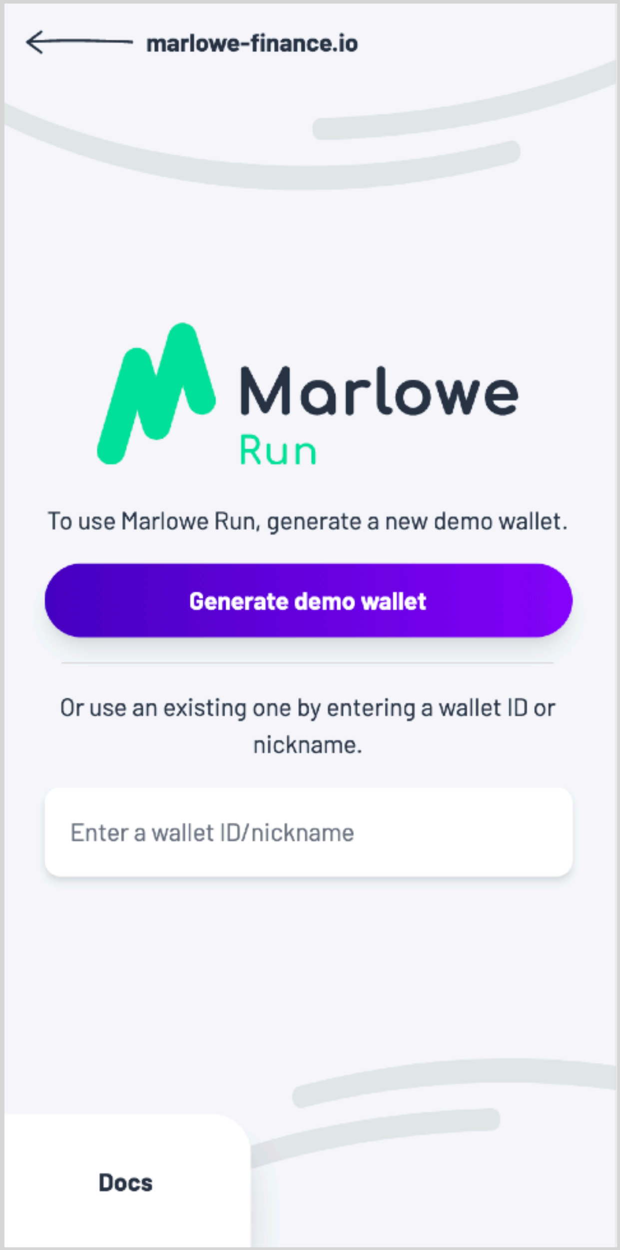


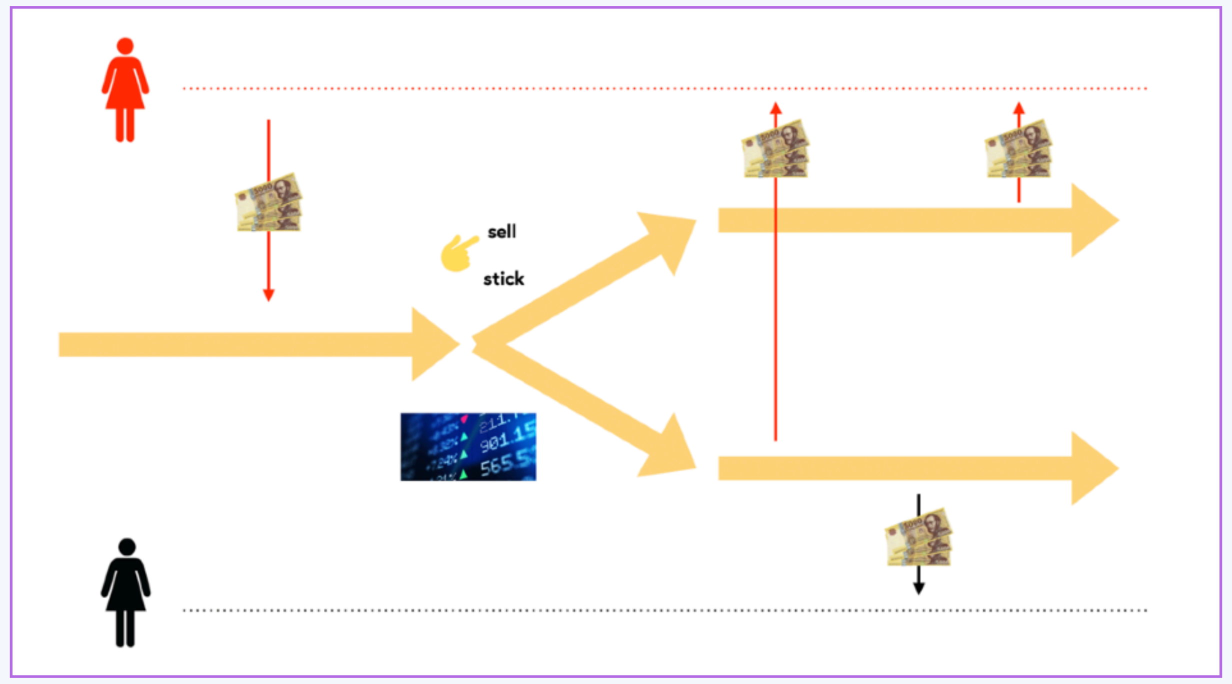


Validation is through the Marlowe interpreter, *i.e.* a Plutus contract.

Transactions built by Marlowe Run + wallet

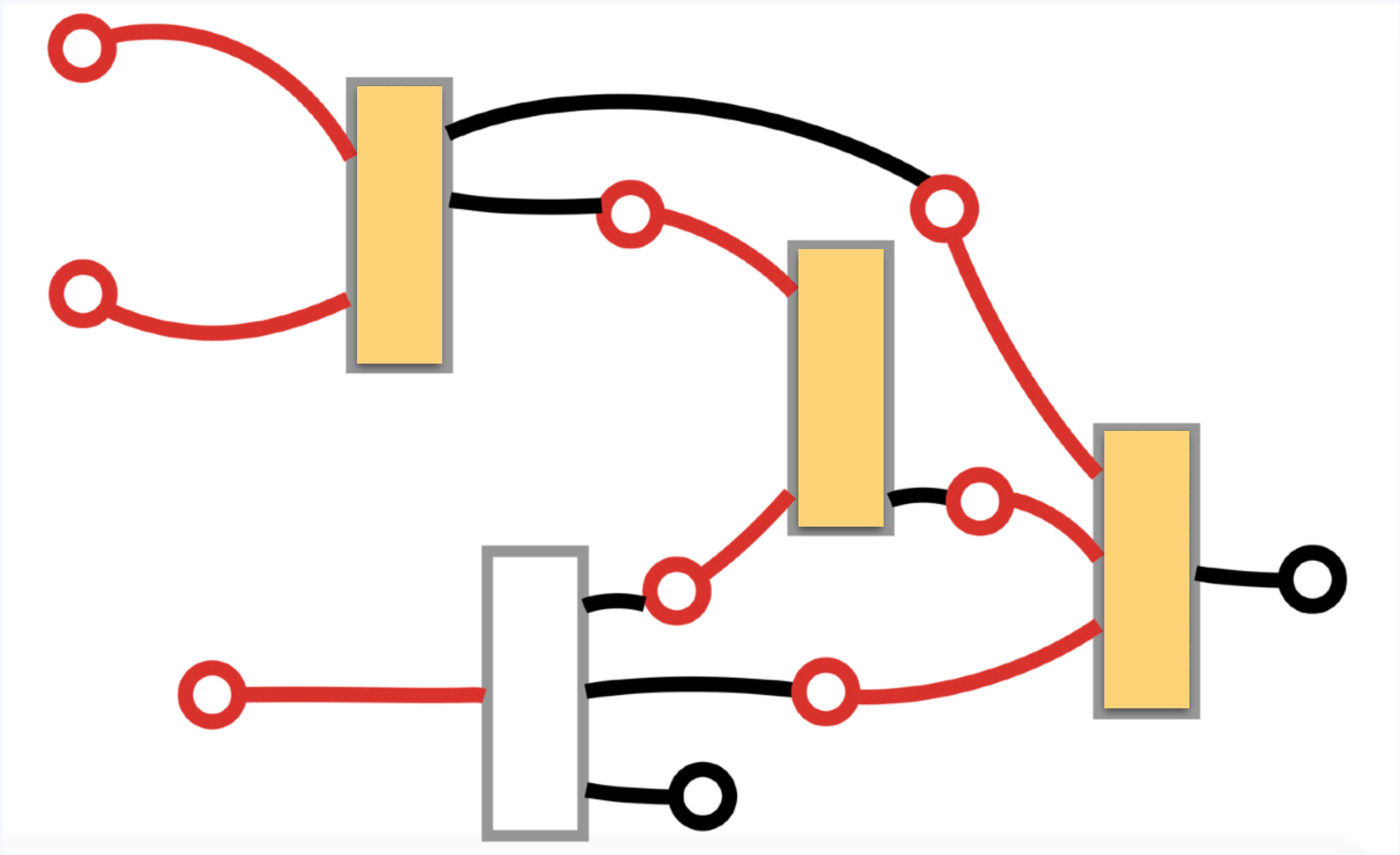


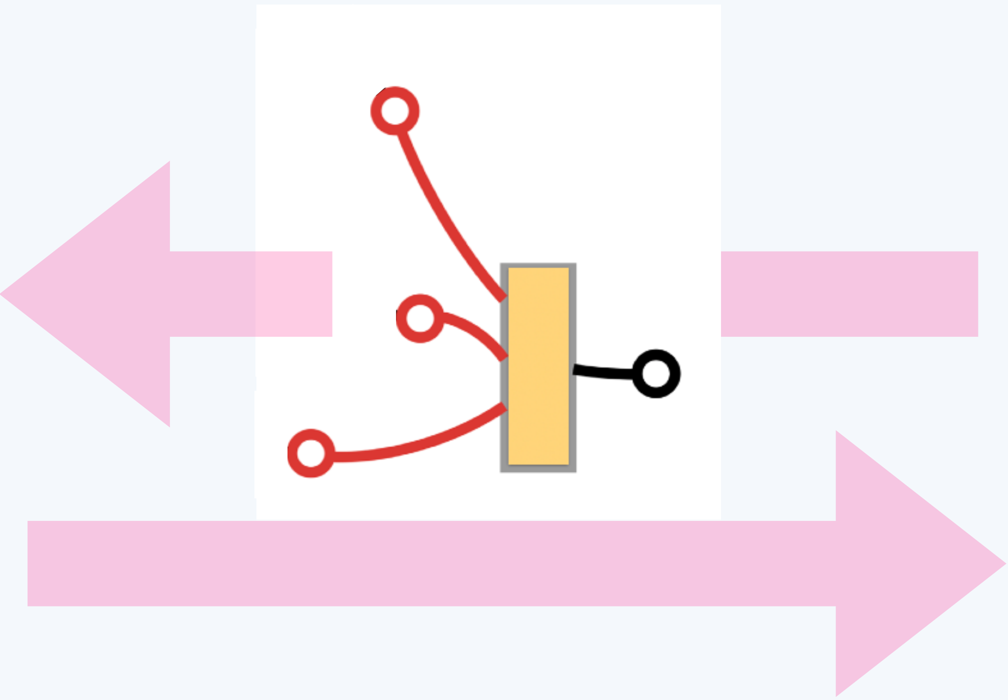


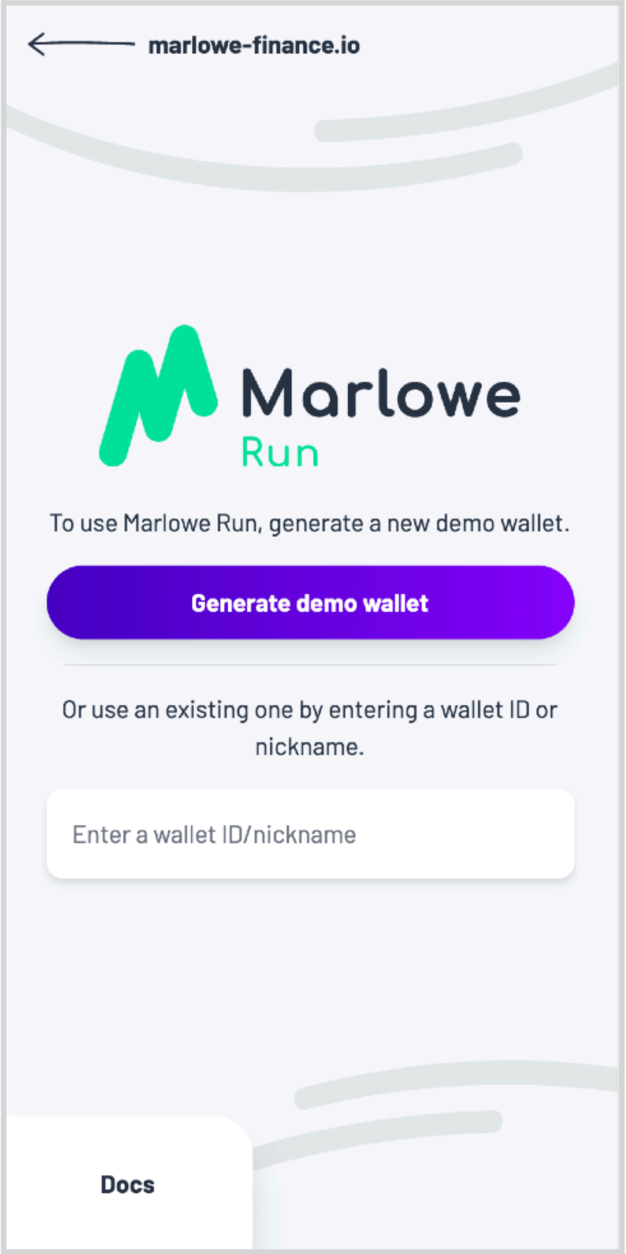


Validation is through the Marlowe interpreter, *i.e.* a Plutus contract.

Transactions built by Marlowe Run + wallet







DAEDALUS

System design

```haskell
-- | Carry a step of the contract with no inputs
reduceContractStep :: Environment -> State -> Contract -> ReduceStepResult
reduceContractStep env state contract = case contract of

    Close -> case refundOne (accounts state) of
        Just ((party, money), newAccounts) -> let
            newState = state { accounts = newAccounts }
            in Reduced ReduceNoWarning (ReduceWithPayment (Payment party money)) newState Close
        Nothing -> NotReduced

    Pay accId payee val cont -> let
        amountToPay = evalValue env state val
        in  if amountToPay <= 0
            then Reduced (ReduceNonPositivePay accId payee amountToPay) ReduceNoPayment state cont
            else let
                balance    = moneyInAccount accId (accounts state) -- always positive
                moneyToPay = Lovelace amountToPay -- always positive
                paidMoney  = min balance moneyToPay -- always positive
                newBalance = balance - paidMoney -- always positive
                newAccs    = updateMoneyInAccount accId newBalance (accounts state)
                warning = if paidMoney < moneyToPay
                          then ReducePartialPay accId payee paidMoney moneyToPay
                          else ReduceNoWarning
                (payment, finalAccs) = giveMoney payee paidMoney newAccs
                in Reduced warning payment (state { accounts = finalAccs }) cont
```

Semantics = executable specification in Haskell

Denotational semantics

Definitional interpreter

Semantics = executable specification in Haskell

Denotational semantics	Definitional interpreter
Completeness	Must cover *all* cases

Semantics = executable specification in Haskell

Denotational semantics	Definitional interpreter
Completeness	Must cover *all* cases
Engagement	Can *run* the semantics

Repurpose the semantics

In Isabelle	For reasoning and proof
In Plutus (≃ Haskell)	For implementation on blockchain
In PureScript	For browser-based simulation

Aside: how to verify that these versions are the same?

Extract Haskell code from the Isabelle version.

Test this against the original Haskell version on random contracts.

Eventually use a Haskell in JS implementation to replace the PureScript.

Usable

Usable

CONTRACT WRITING AND UNDERSTANDING

Marlowe contracts can be *authored* in various different ways.

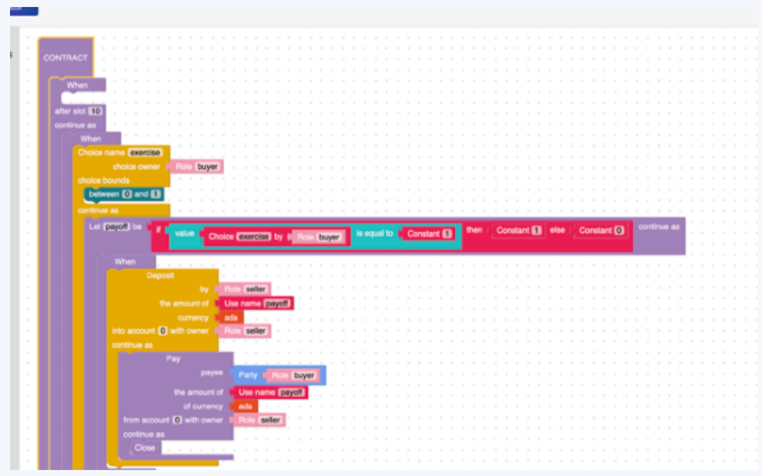
Marlowe contracts can be explored before they are run in a *simulation*.

Usable

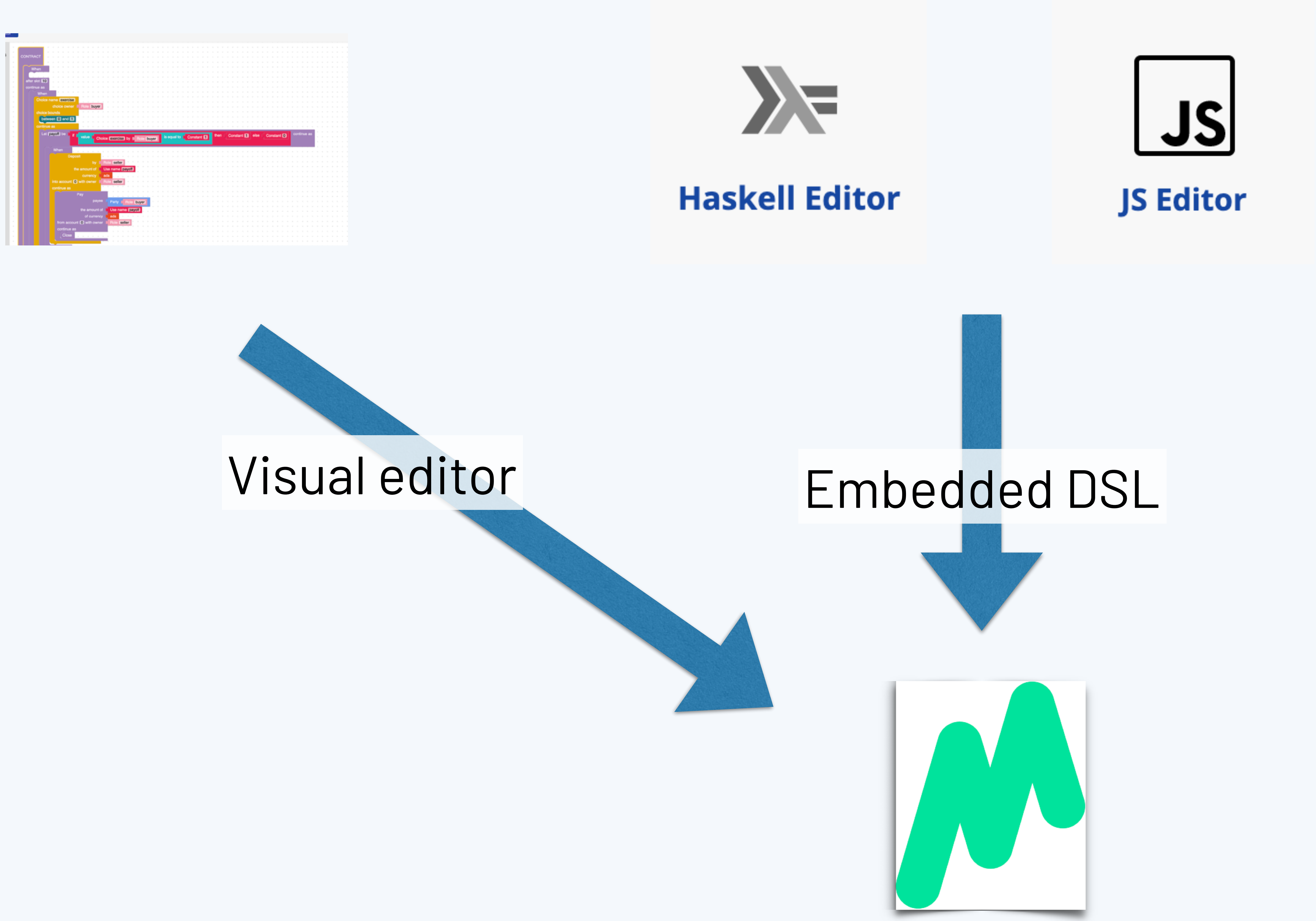
CONTRACT WRITING AND UNDERSTANDING

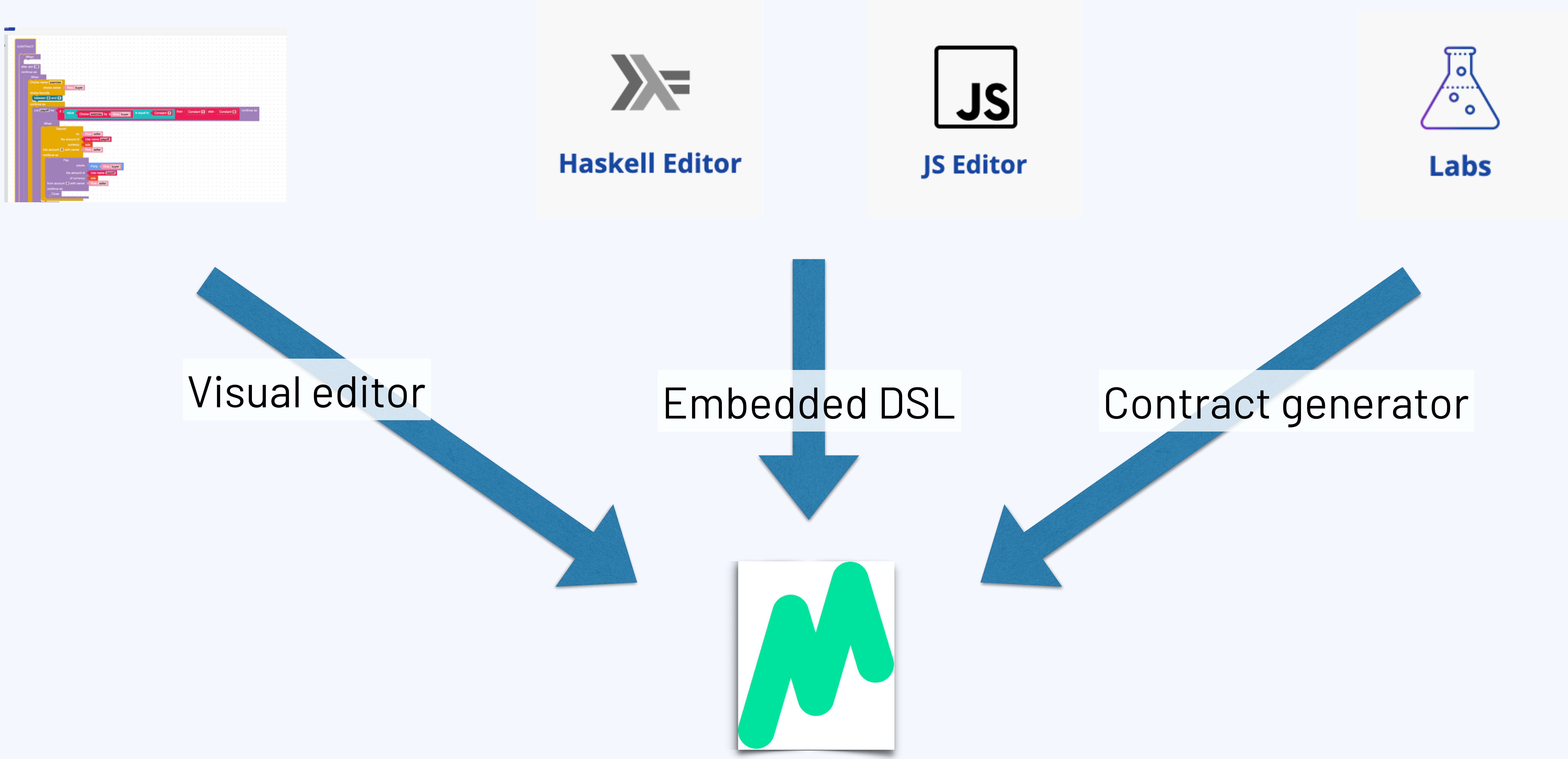
Marlowe contracts can be *authored* in various different ways.

Marlowe contracts can be explored before they are run in a *simulation*.



Visual editor


Haskell Editor
JS Editor
Visual editor
Embedded DSL


Haskell Editor
JS Editor
Labs
Visual editor
Embedded DSL
Contract generator

MARLOWE PLAYGROUND

Escrow with collateral *

Tutorials   Actus Labs

New Project   Open   Open Example   Rename   Save   Save As...   Edit source

```
When
    [Case
        (Deposit
            (Role "Seller")
            (Role "Buyer")
            (Token "" "")
            (Constant 100000000)
        )
        (When
            [Case
                (Choice
                    (ChoiceId
                        "Everything is alright"
                        (Role "Buyer")
                    )
                    [Bound 0 0]
                )
                Close , Case
                (Choice
                    (ChoiceId
                        "Report problem"
                        (Role "Buyer")
                    )
                    [Bound 1 1]
                )
                (Pay
                    (Role "Seller")
                    (Account (Role "Buyer"))
                    (Token "" "")
                    (Constant 100000000)
                    (When
                        [Case
                            (Choice
                                (ChoiceId
                                    "Confirm problem"
                                    (Role "Seller")
                                )
                                [Bound 1 1]
                            )
                            Close , Case
                            (Choice
                                (ChoiceId
```

Current State

current slot: 0   **expiration slot:** 17

ACTIONS

*Participant **Buyer***   "The party that pays for the item on sale.,

Deposit **100,000,000** units of **ADA** into account of **Seller** as **Buyer**   +

Other Actions

Move to slot 10   +

Undo   Reset

TRANSACTION LOG

Action	Slot
Deposit **1,000,000** units of **ADA** into account of **Seller** as **Seller**	0
Deposit **1,000,000** units of **ADA** into account of **Buyer** as **Buyer**	0

cardano.org | iohk.io

Telegram | Twitter

Escrow with collateral

ESCROW

Tasks | Balances

Step 1 Completed

S Seller

Seller made a deposit of ₳ 1.000000 into their account on 31 May 2021 between 08:44 and 08:45

Tasks | Balances

Step 2 Completed

B Buyer

You made a deposit of ₳ 1.000000 into your account on 31 May 2021 between 08:45 and 08:46

Tasks | Balances

Step 3 Completed

B Buyer

You made a deposit of ₳ 1,000.000000 into Seller's account on 31 May 2021 between 08:45 and 08:47

Tasks | Balances

Step 4 Completed

B Buyer

You chose 0 for "Everything is alright" on 31 May 2021 between 08:45 and 08:49

Tasks | Balances

Step 5 Contract closed

This contract is now closed

There are no tasks to complete

Next

Assurance

Assurance

USING THE POWER OF LOGIC

Static analysis: automatic verification of properties of individual contracts.

Verification: machine-supported proof of system and contract properties.

Static analysis

Can check *all* execution paths through a Marlowe contract.

All choices, *all* choices of slots for transaction submission.

Example: is it possible there may not be enough to fulfil a *Pay* construct?

Constructive: if it is, then here's a counter-example.

Static analysis

Can check *all* execution paths through a Marlowe contract.

All choices, *all* choices of slots for transaction submission.

Example: is it possible there may not be enough to fulfil a *Pay* construct?

Constructive: if it is, then here's a counter-example.

```
            (Role "alice")
            (Role "alice")
            (Token "" "")
            (Constant 450)
        )
        (When
            [Case
                (Choice
                    (ChoiceId
                        "choice"
                        (Role "alice")
                    )
                    [Bound 0 1]
                )
                (When
                    [Case
                        (Choice
                            (ChoiceId
```

Current State | **Static Analysis** | Warnings | Errors | Logs

Warning Analysis Result: Pass

Static analysis could not find any execution that results in any warning.

Analyse for warnings | Analyse reachability

*Participant **alice***

Deposit **450** units of **A**
Account **(Role "alice")**
"alice")

Other Actions

Move to slot 10

Undo

Modelling co
Marlowe

Marlowe is designed to
execution of financial c
blockchain, and specifi
Cardano. Contracts are
together a small numb
in combination can be
many different kinds o

```
            (Role "alice")
            (Role "alice")
            (Token "" "")
            (Constant 40)
        )
        (When
            [Case
                (Choice
                    (ChoiceId
                        "choice"
                        (Role "alice")
                    )
                    [Bound 0 1]
                )
                (When
                    [Case
                        (Choice
                            (ChoiceId
```

Participant ***alice***

Deposit **40** units of **AD**
(Role "alice") as **(Role**

Other Actions

Move to slot 10

Undo

Modelling co
Marlowe

Marlowe is designed to
execution of financial c
blockchain, and specifi
Cardano. Contracts are
together a small numb
in combination can be
many different kinds o

Current State | **Static Analysis** | Warnings (4) | Errors | Logs

Warning Analysis Result: Warnings Found

Static analysis found the following counterexample:

- Warnings issued:
  1. **TransactionPartialPay** - The contract is supposed to make a payment of **450** units of **ADA** from account of **(Role "alice")** to **party (Role "bob")** but there is only **40**.
- Initial slot: **0**
- Offending transaction list:
  1. **Transaction** with slot interval **0 to 3** and inputs:
     a. **IDeposit** - Party **(Role "alice")** deposits **40** units of **ADA** into account of **(Role "alice")**.
  2. **Transaction** with slot interval **1 to 2** and inputs:
     a. **IChoice** - Party **(Role "alice")** chooses number **0** for choice **"choice"**.
  3. **Transaction** with slot interval **1 to 1** and inputs:
     a. **IChoice** - Party **(Role "bob")** chooses number **0** for choice **"choice"**.

Analyse for warnings | Analyse reachability

The system is safe

Prove properties of the Marlowe system once and for all.

Theorem: Accounts are never -ve.

Theorem: Money preservation:

money_in = money_in_accounts + money_out

Theorem: Close produces no warnings.

Theorem: Static analysis is sound and complete.

And we can do the same for individual contracts and templates too.

The system is safe

Prove properties of the Marlowe system once and for all.

Theorem: Accounts are never -ve.

Theorem: Money preservation:

$$money\_in = money\_in\_accounts + money\_out$$

Theorem: Close produces no warnings.

Theorem: Static analysis is sound and complete.

And we can do the same for individual contracts and templates too.

More information about Marlowe

The marlowe and plutus github repositories.

The IOHK research library: search for “Marlowe”.

Online tutorial in the Marlowe Playground.

Alex's presentation coming up next.

Marlowe

A SPECIAL-PURPOSE LANGUAGE FOR FINANCIAL CONTRACTS

Designed for users, as well as developers.

Designed for maximum assurance.

Assurance

CONTRACTS DO WHAT THEY SHOULD ...
... AND NOT WHAT THEY SHOULDN'T

Language as *simple* as it can be.

Contracts can be *read* and *simulated*.

Before running, can explore *all behaviour.*

System can be *proved safe* in various ways.

https://play.marlowe-finance.io/