Standardized crypto-loans on the Cardano blockchain

Dmytro Kondratiuk, Pablo Lamela Seijas, Alex Nemish, Simon Thompson: IOHK and Kent

WTSC'21, 5 March 2021

Overview

Finance and ACTUS

Marlowe …

… language,

… and design

ACTUS + Cardano

Executable specification

Contract generation

Assurance

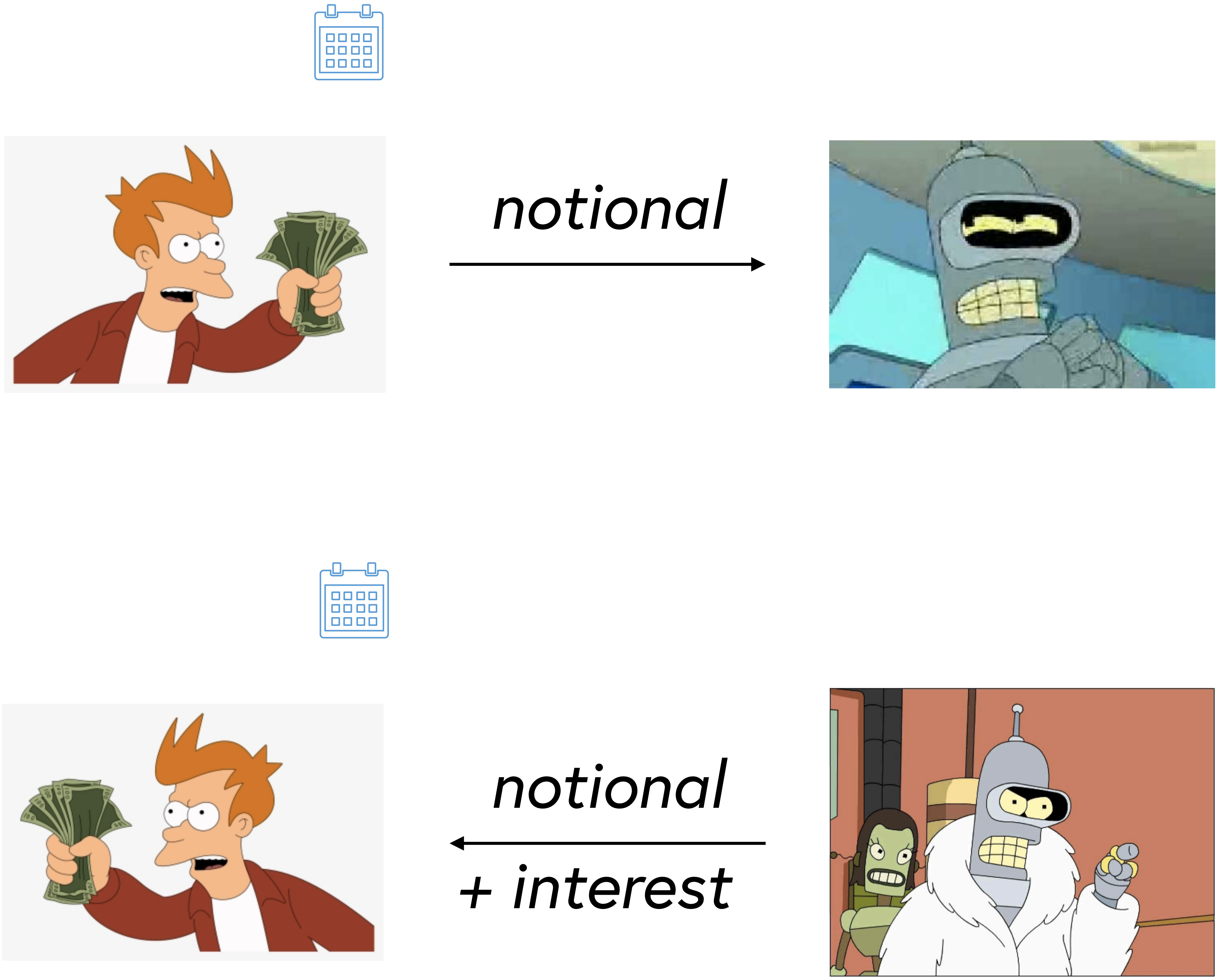
Finance

Loans

Lender advances *notional* to *borrower.*

Borrower will pay *charges* and *interest*, and *repays* on a given date.

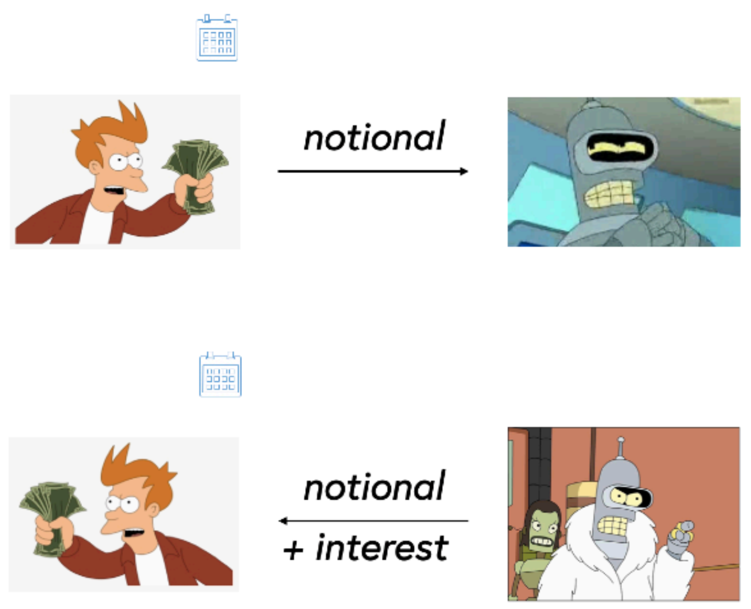
Simplest possible example, *zero coupon bond: repay* with *interest.*



notional

notional

+ interest



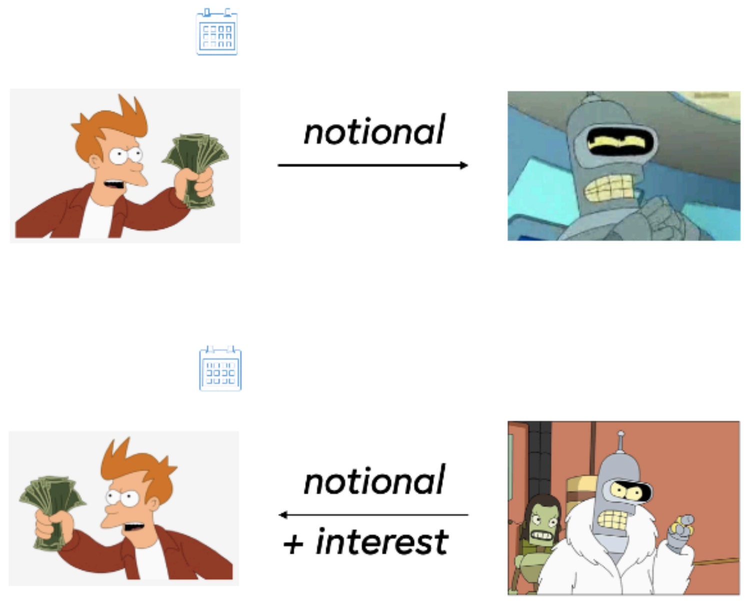
Beyond the zero coupon bond

Make *scheduled* interest payments during the life of the contract.

Interest rate can vary during the life of the contract: a *risk factor*.

The first option is entirely static …

… the second requires (re)calculation during contract execution..



Collateral

In the case of *trustless* blockchain, why should the lender ever repay?

Collateral: can be *crypto-asset* e.g. ADA used against *fiat* / *stable-coin* loan, e.g. USDT.

Borrower gains *liquidity* without selling their crypto-asset, and pays for that in *interest*.

Risk lies in the variable *valuation* of the collateral …

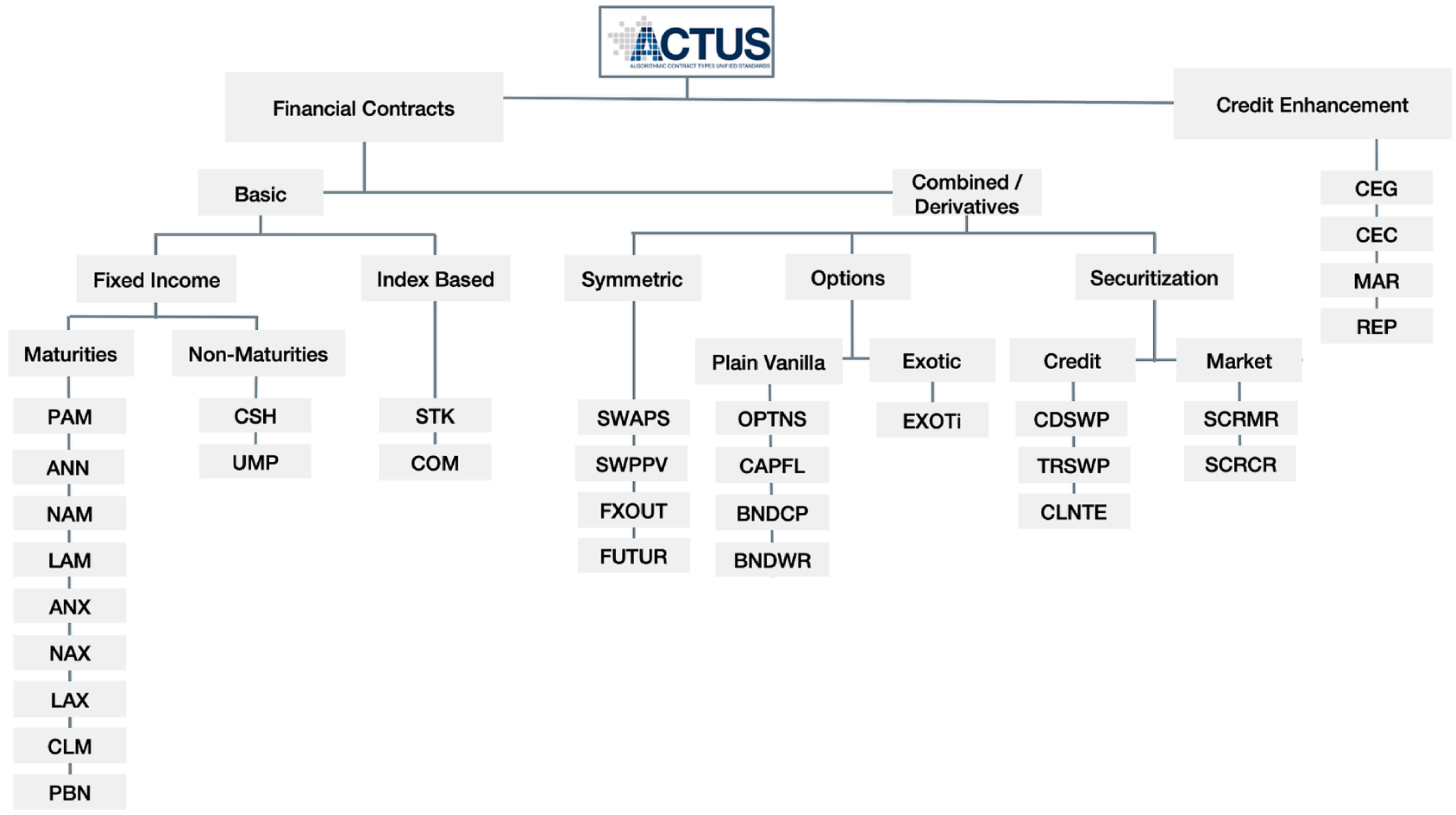
ACTUS: Algorithmic Contract Types Unified Standards

www.actusfrf.org

Different degrees of dynamism:

- Static
- Variable rates
- Off schedule payments

Tradeoff between guarantees and dynamic behaviour.



ACTUS

Financial Contracts

Credit Enhancement

Basic

Combined / Derivatives

Fixed Income

Index Based

Symmetric

Options

Securitization

Maturities

Non-Maturities

Plain Vanilla

Exotic

Credit

Market

PAM

ANN

NAM

LAM

ANX

NAX

LAX

CLM

PBN

CSH

UMP

STK

COM

SWAPS

SWPPV

FXOUT

FUTUR

OPTNS

CAPFL

BNDCP

BNDWR

EXOTi

CDSWP

TRSWP

CLNTE

SCRMR

SCRCR

CEG

CEC

MAR

REP

ACTUS state machines

Contract terms

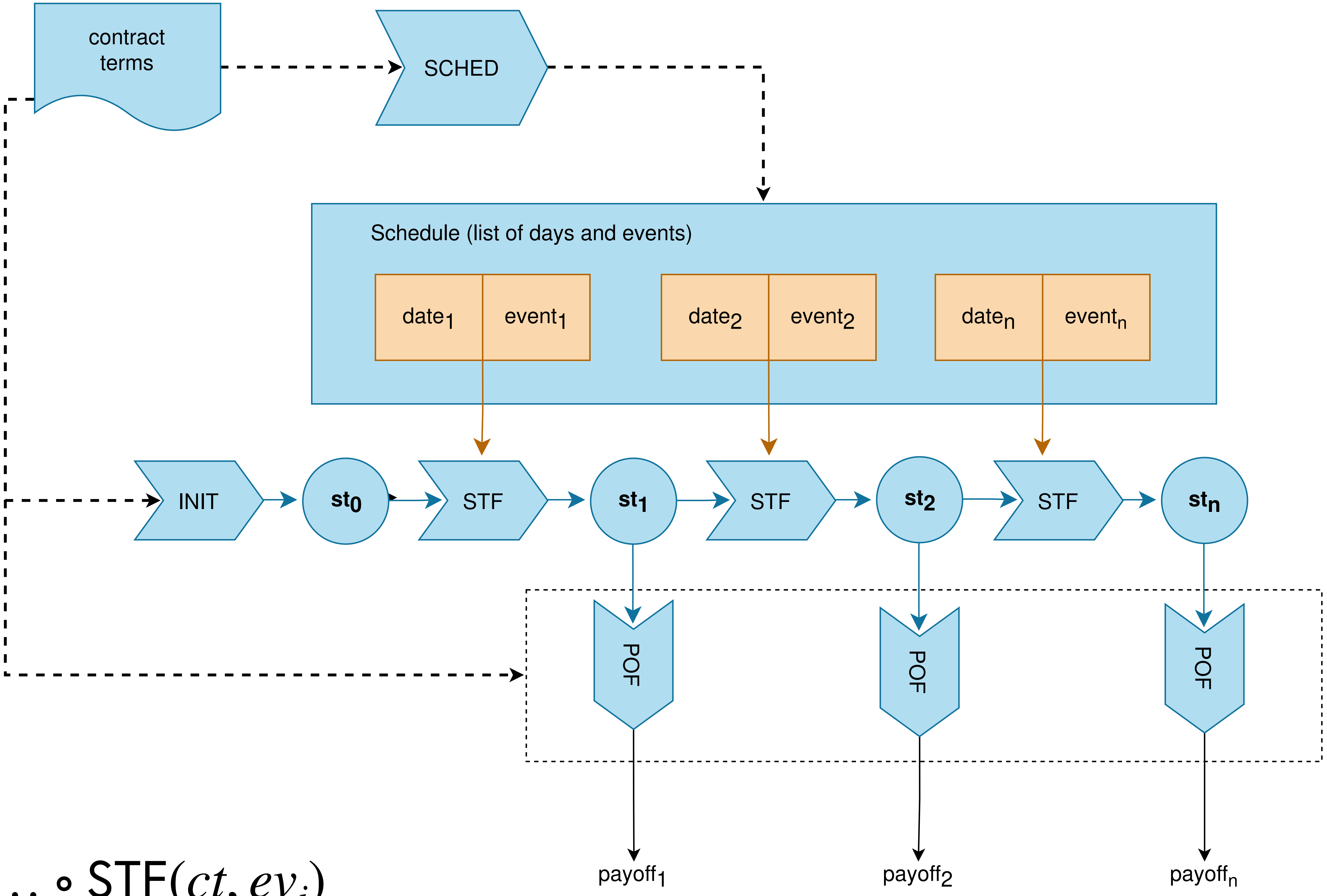
Scheduled events

State transformation function

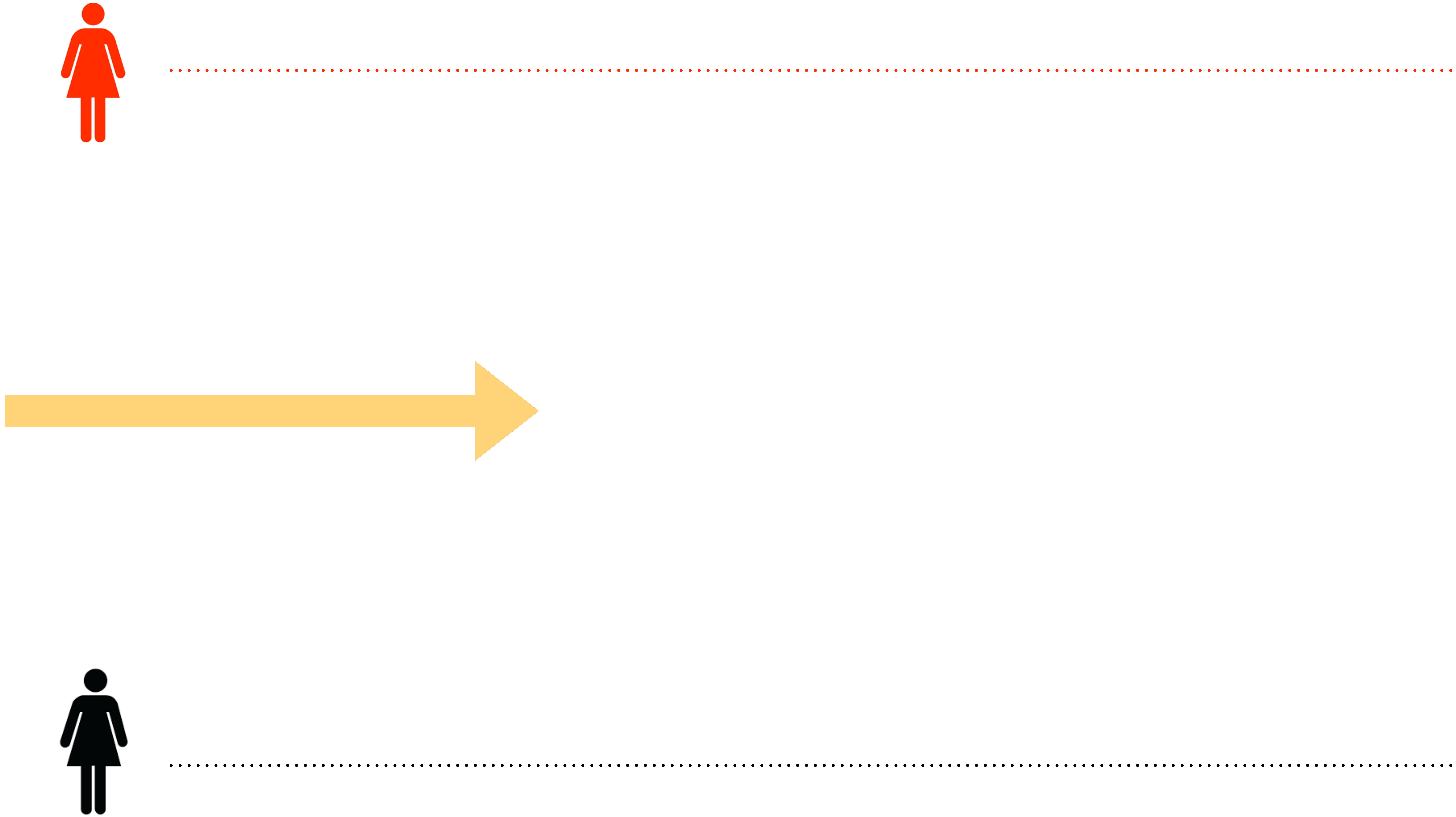
$\text{payoff}_i = \text{POF}(\text{state}_i)$

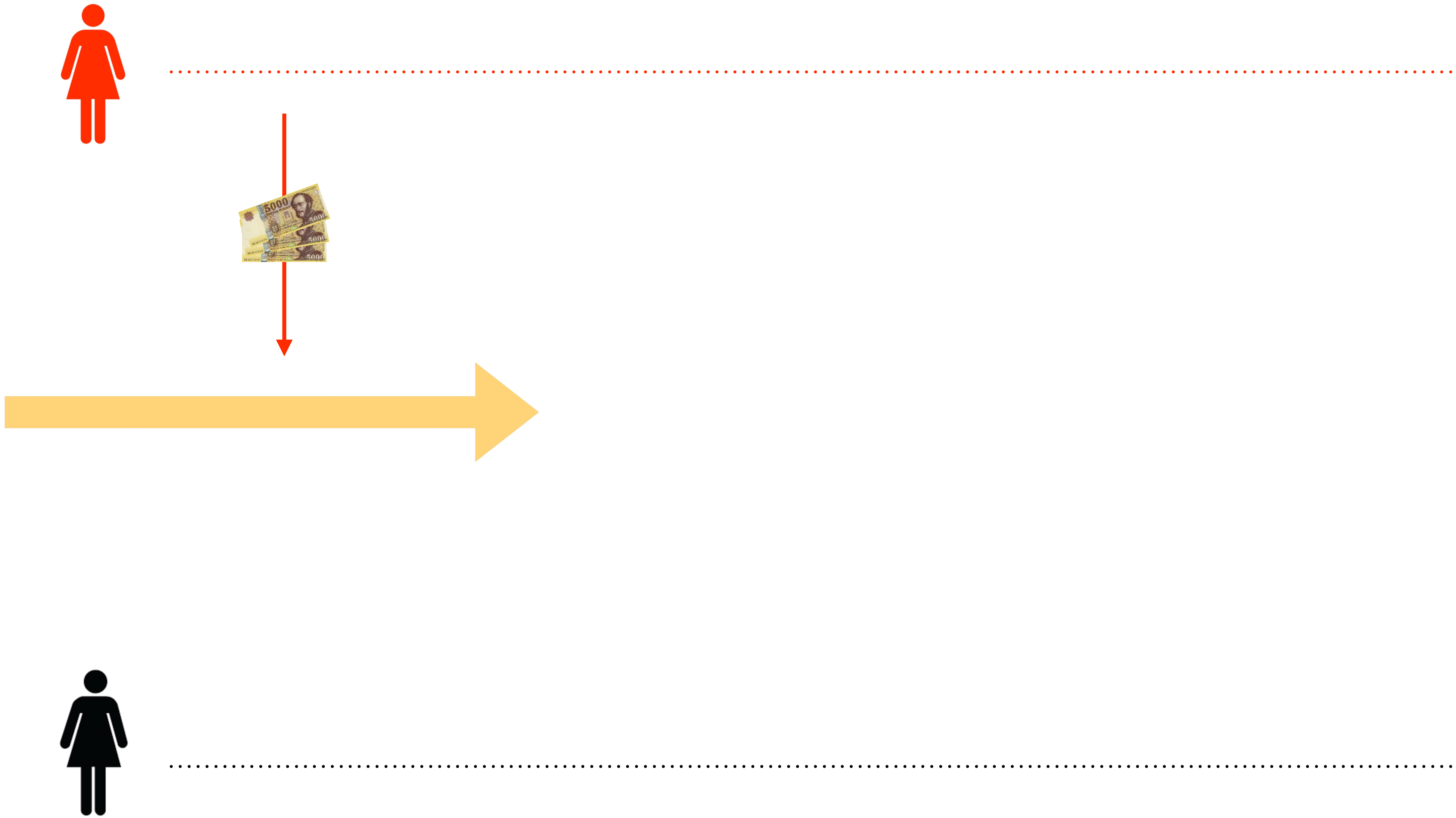
$\text{state}_i = \text{path}_i(\text{INIT}(ct))$

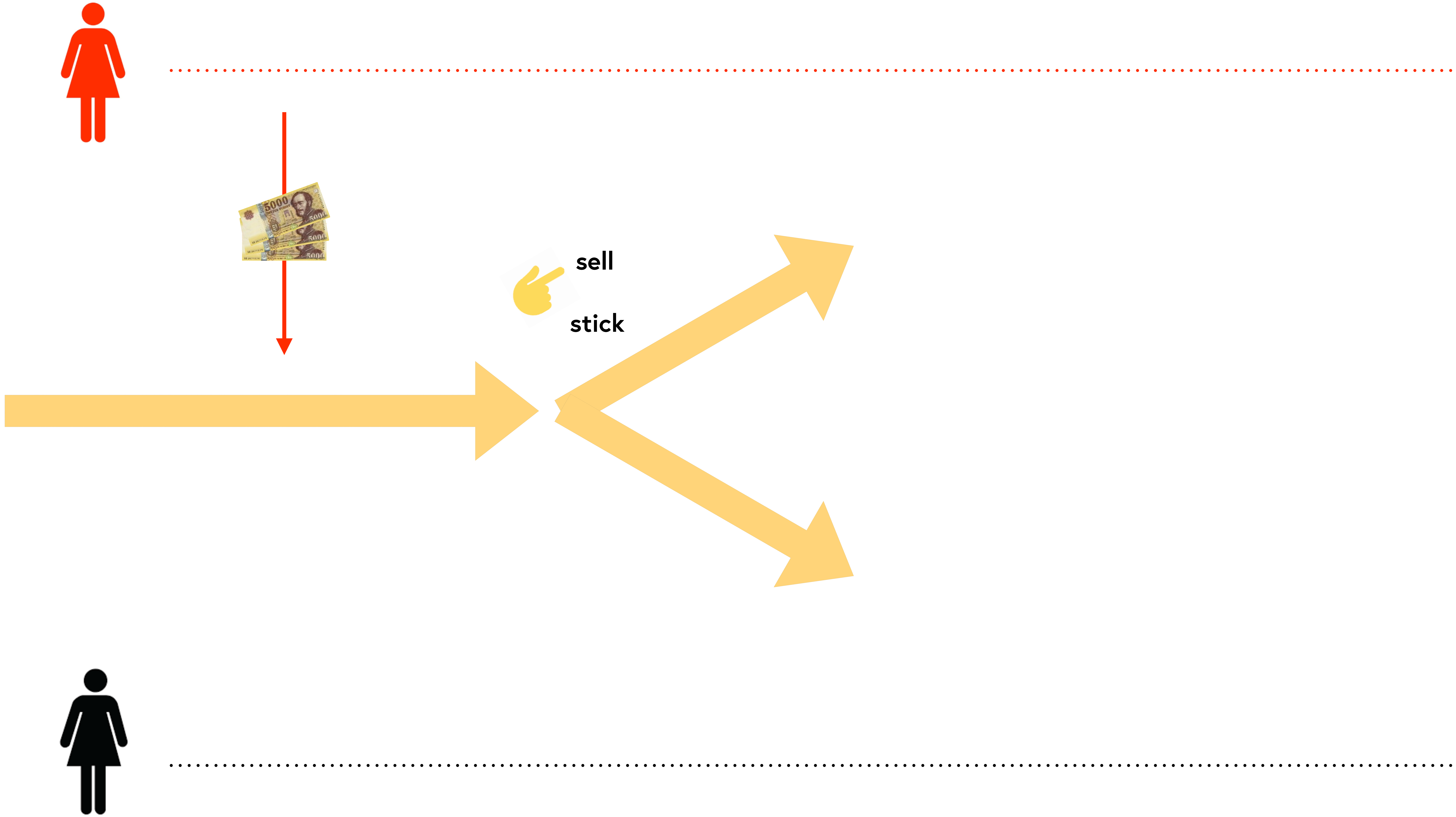
$\text{path}_i = \text{STF}(ct, ev_1) \circ \text{STF}(ct, ev_2) \circ \ldots \circ \text{STF}(ct, ev_i)$



Marlowe

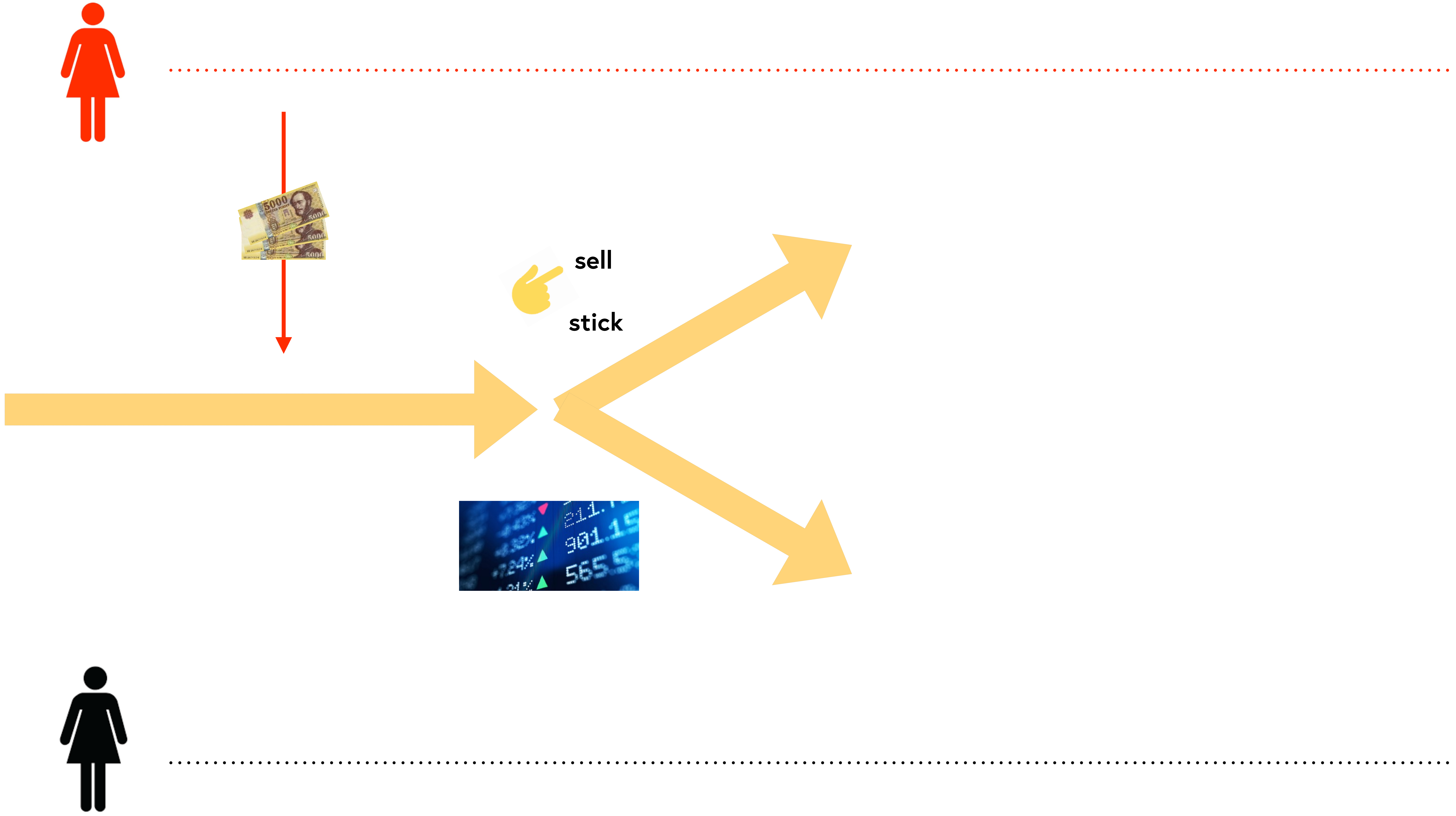



5000



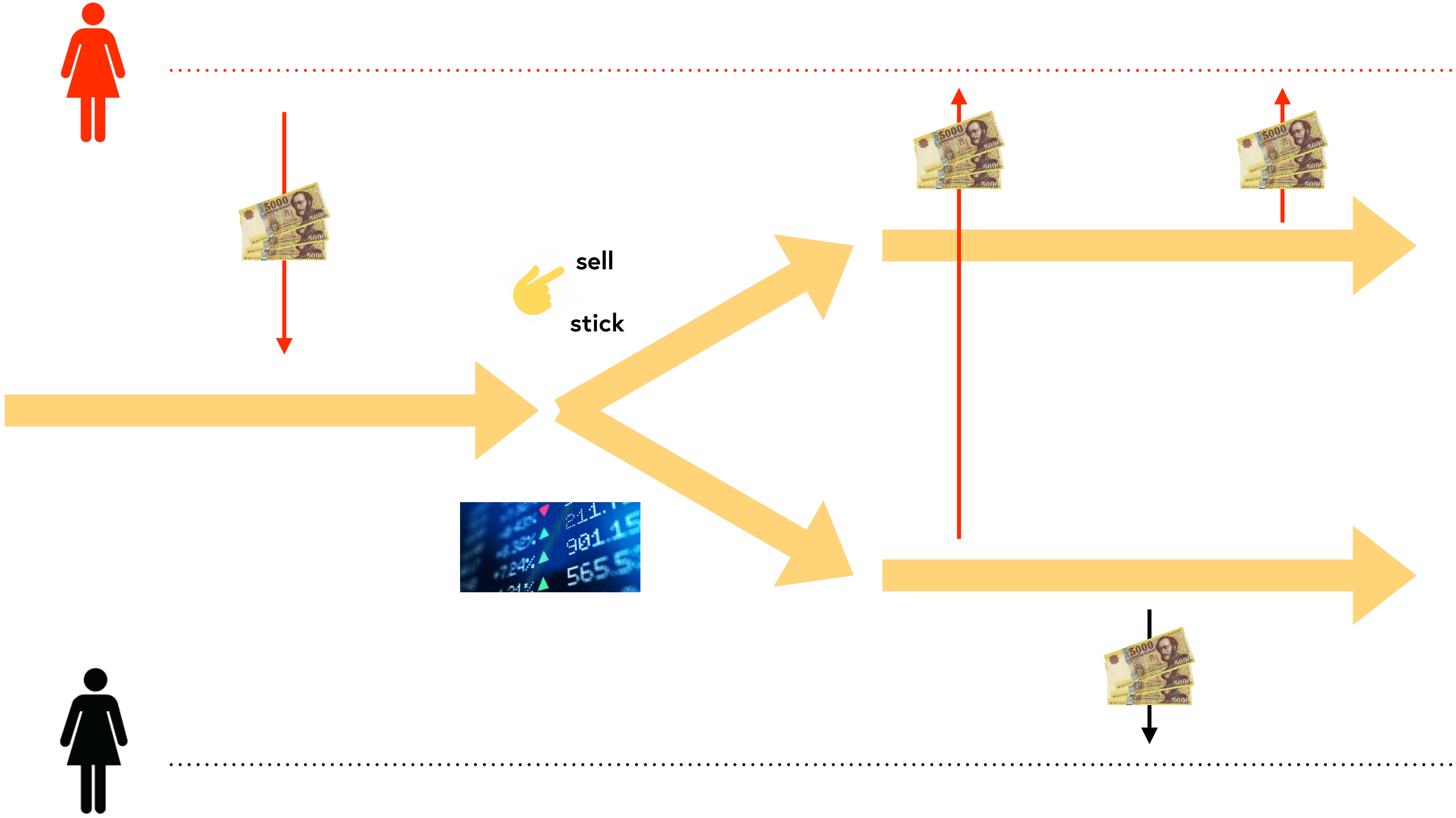
sell

stick



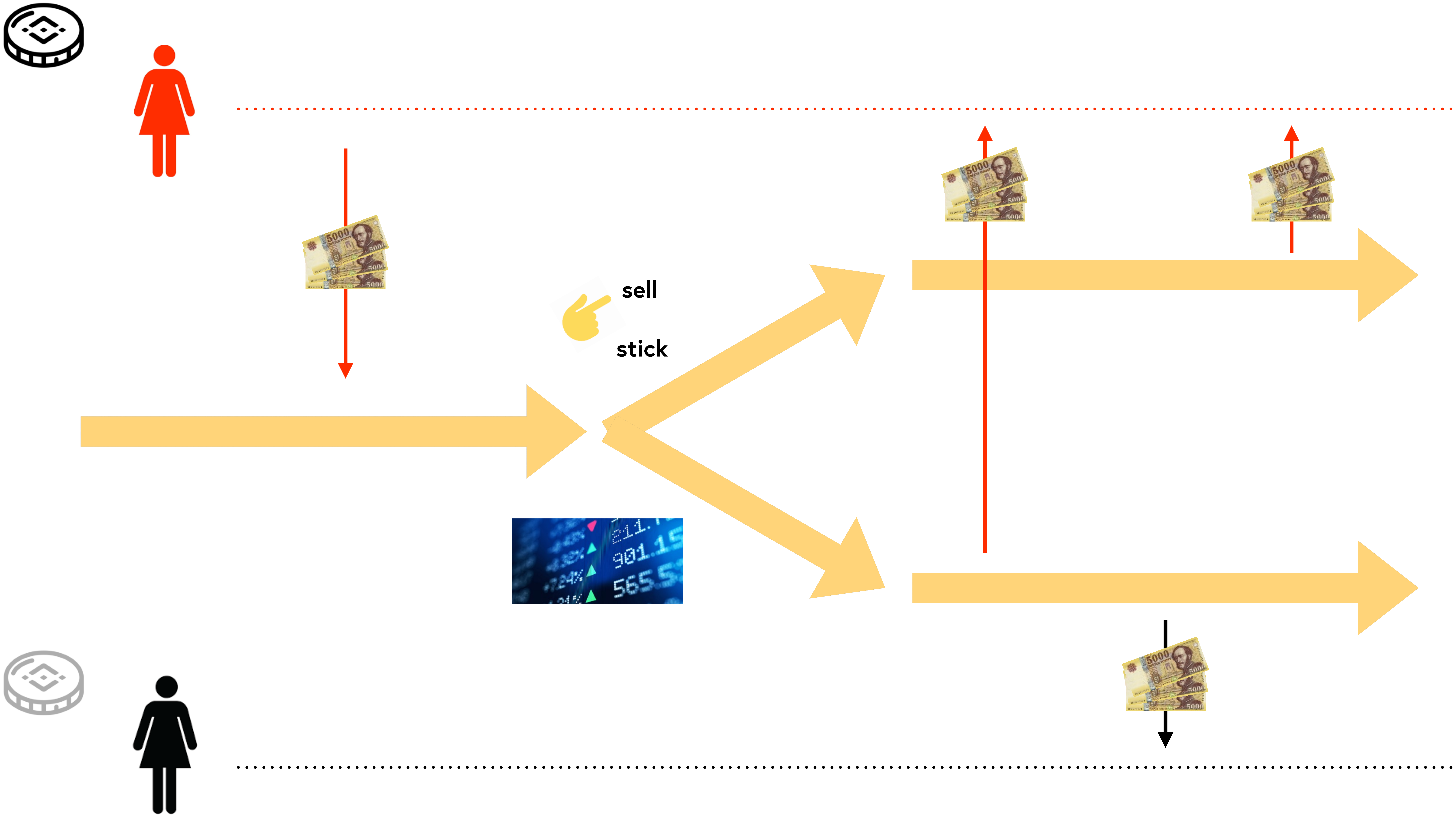
sell

stick



sell

stick



sell

stick

Design

But a contract could …

… run forever.

… wait for an input forever.

… terminate holding assets.

… "double spend" assets.

Design

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 for refund on close.
Conservation of value.	Underlying blockchain + defined constructs.

Design

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
```

Design

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
```

Design

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
```

Design

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
```

Design

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
```

Design

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
```

Design

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
```

Design

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
```

Design

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
```

Design

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
```

Design

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
```

Design

Contracts are finite.

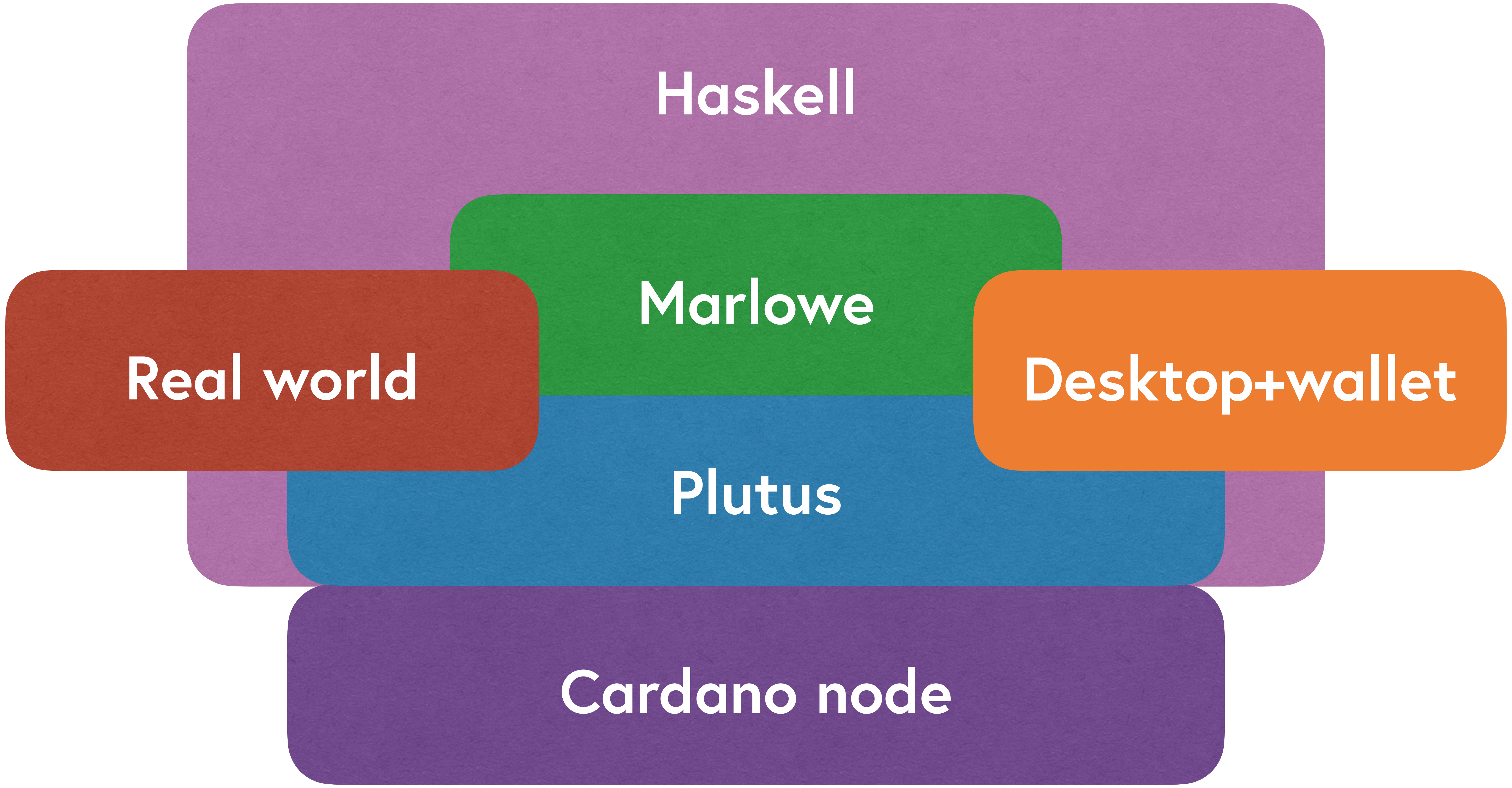
Contracts will terminate …

… 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
```



Haskell

Marlowe

Real world

Desktop+wallet

Plutus

Cardano node

Assurance

Quick Check: random-based testing of system and contract properties.

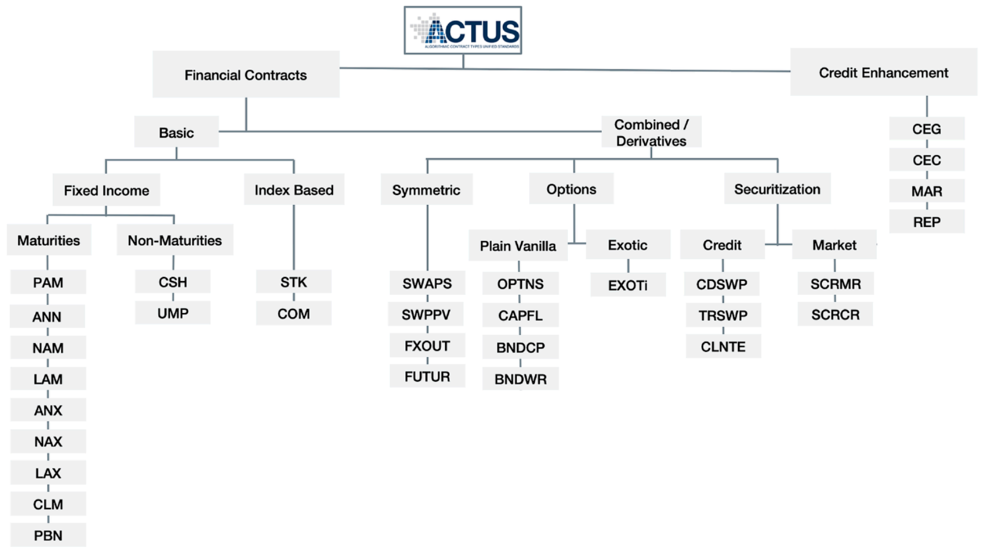
Static analysis: automatic verification of properties of individual contracts.

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

ACTUS standard: generate contracts from high-level specs, using Haskell or Agda.

ACTUS + Cardano

ACTUS in Cardano

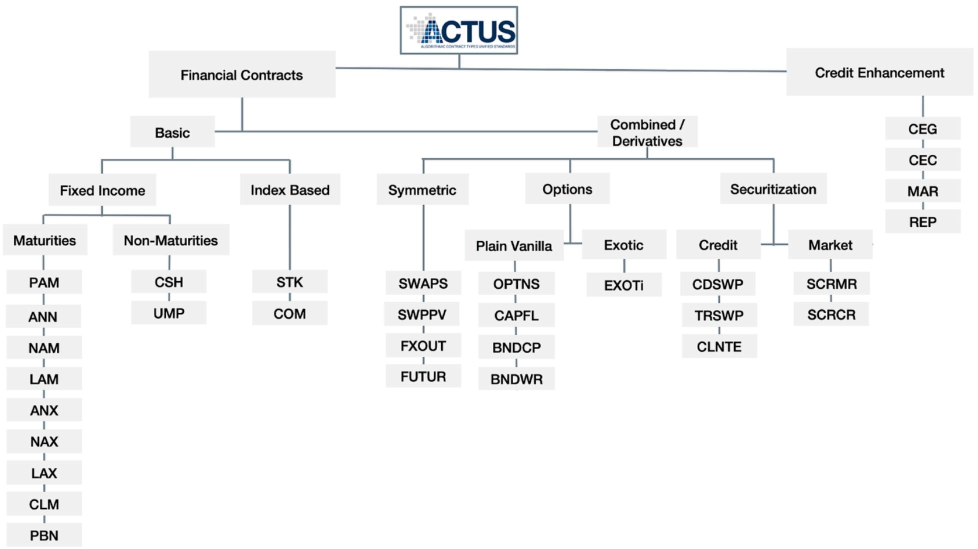


Executable specification of ACTUS in Haskell.

Generation of ACTUS contracts in Marlowe from *contact terms.*

ActusLabs interface for composing contract terms in Blockly.

ACTUS in Cardano



ACTUS

Financial Contracts

Credit Enhancement

Basic

Combined / Derivatives

Fixed Income

Index Based

Symmetric

Options

Securitization

Maturities

Non-Maturities

PAM
ANN
NAM
LAM
ANX
NAX
LAX
CLM
PBN

CSH
UMP

STK
COM

SWAPS
SWPPV
FXOUT
FUTUR

Plain Vanilla

OPTNS
CAPFL
BNDCP
BNDWR

Exotic

EXOTi

Credit

CDSWP
TRSWP
CLNTE

Market

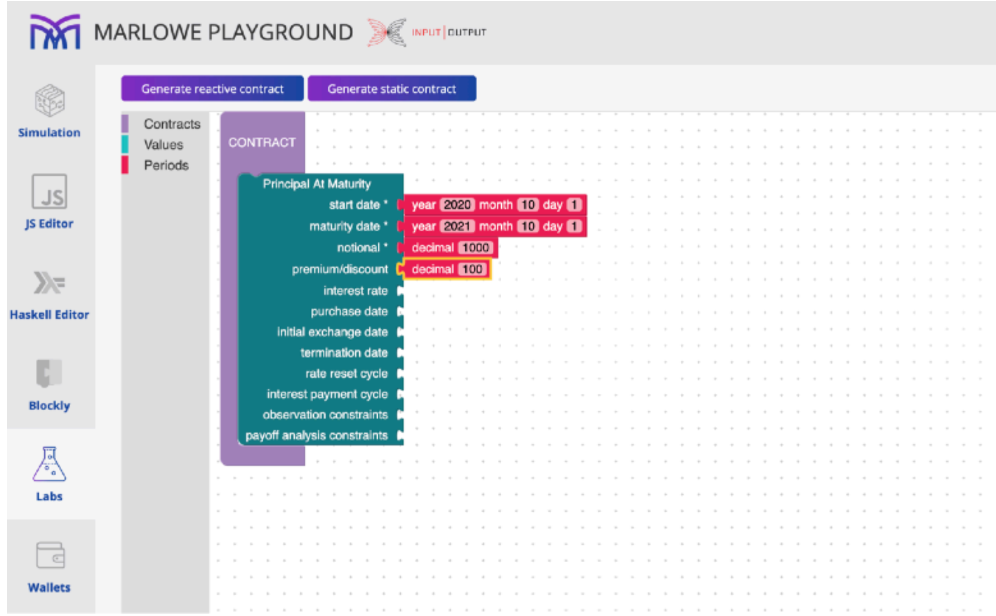
SCRMR
SCRCR

CEG
CEC
MAR
REP

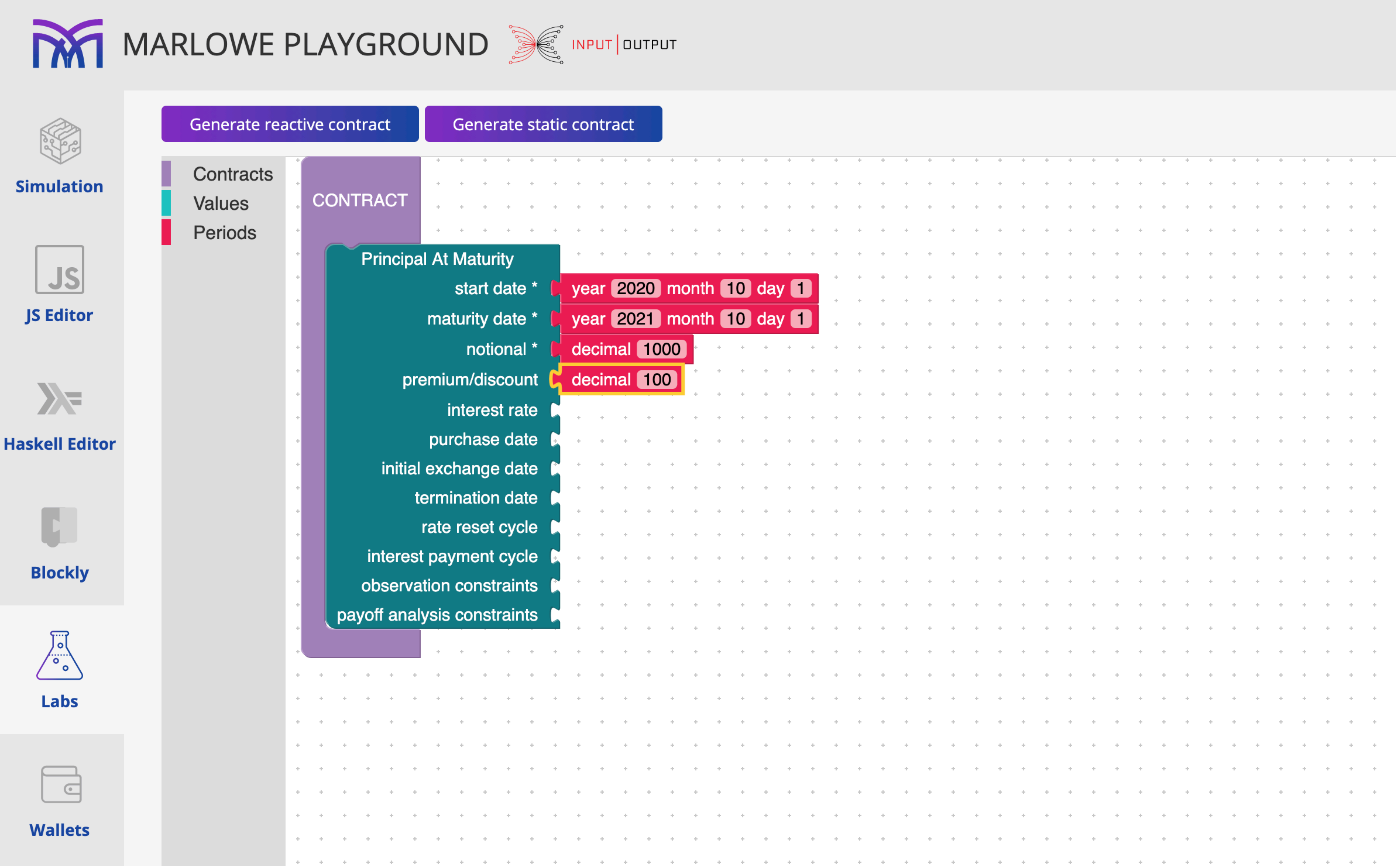
Executable specification of ACTUS in Haskell.

Generation of ACTUS contracts in Marlowe from *contact terms*.

ActusLabs interface for composing contract terms in Blockly.



Principal At Maturity
start date * year 2020 month 10 day 1
maturity date * year 2021 month 10 day 1
notional * decimal 1000
premium/discount decimal 100
interest rate
purchase date
initial exchange date
termination date
rate reset cycle
interest payment cycle
observation constraints
payoff analysis constraints



Principal At Maturity

start date * year 2020 month 10 day 1

maturity date * year 2021 month 10 day 1

notional * decimal 1000

premium/discount decimal 100

interest rate

purchase date

initial exchange date

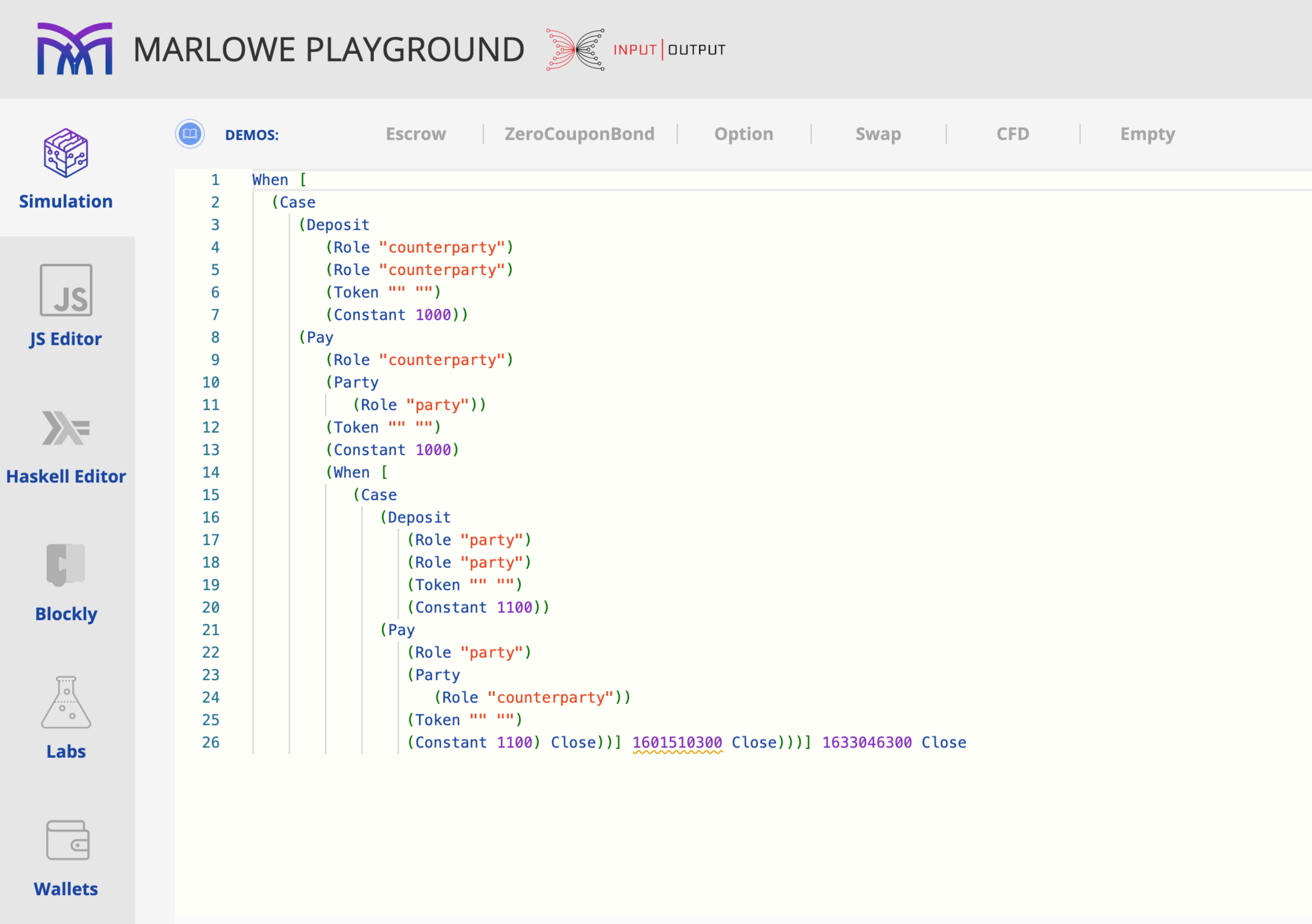
termination date

rate reset cycle

interest payment cycle

observation constraints

payoff analysis constraints



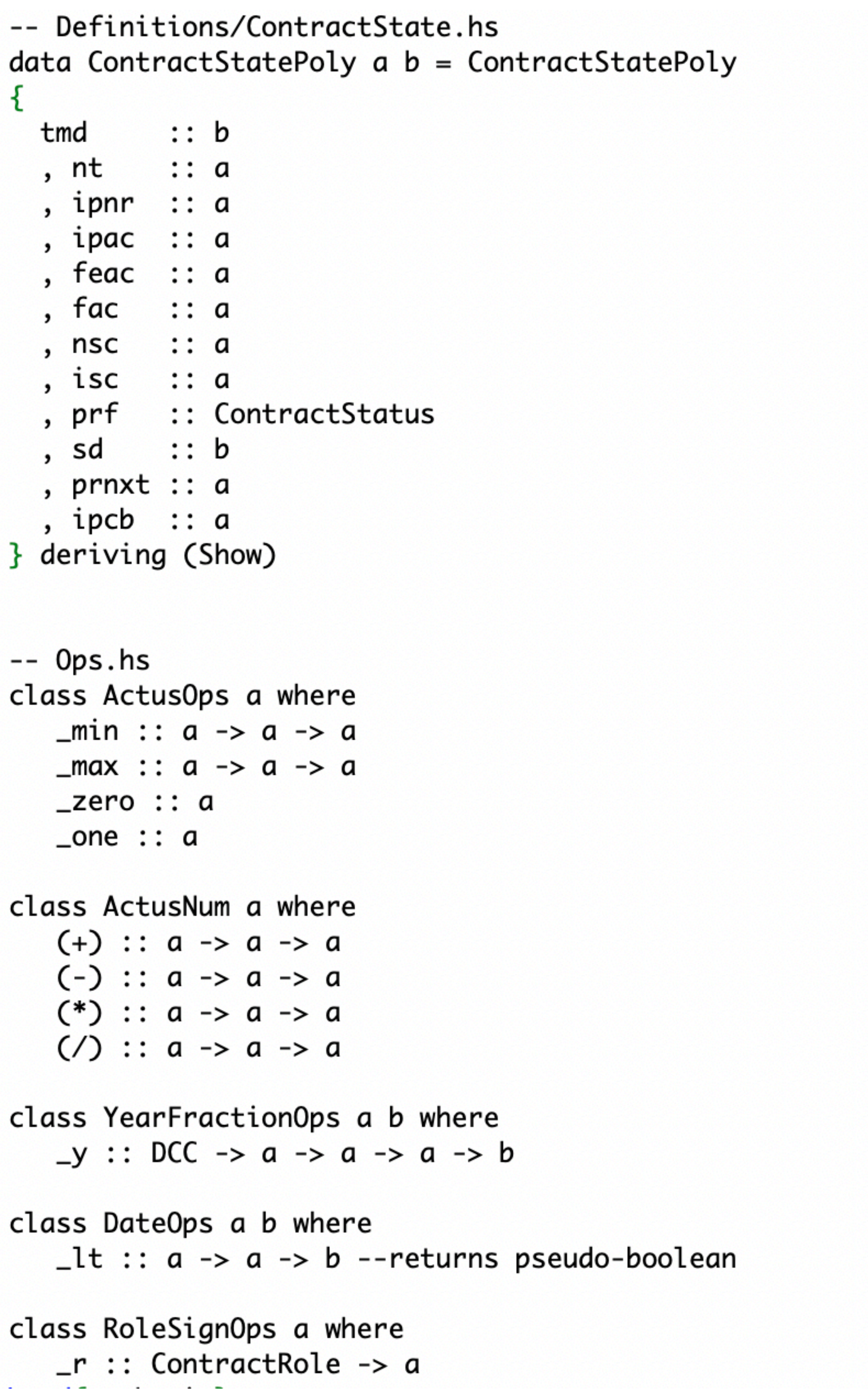
```
When [
  (Case
     (Deposit
        (Role "counterparty")
        (Role "counterparty")
        (Token "" "")
        (Constant 1000))
     (Pay
        (Role "counterparty")
        (Party
           (Role "party"))
        (Token "" "")
        (Constant 1000)
        (When [
           (Case
              (Deposit
                 (Role "party")
                 (Role "party")
                 (Token "" "")
                 (Constant 1100))
              (Pay
                 (Role "party")
                 (Party
                    (Role "counterparty"))
                 (Token "" "")
                 (Constant 1100) Close))] 1601510300 Close)))] 1633046300 Close
```

Executable specification in Haskell

Respect *naming conventions.*

Use Haskell *type classes* for overloading:
a single description gives both …
  … *cash flows* for an instrument and
  … *syntax describing* the same instrument.

Generate Marlowe or Haskell code from these descriptions …



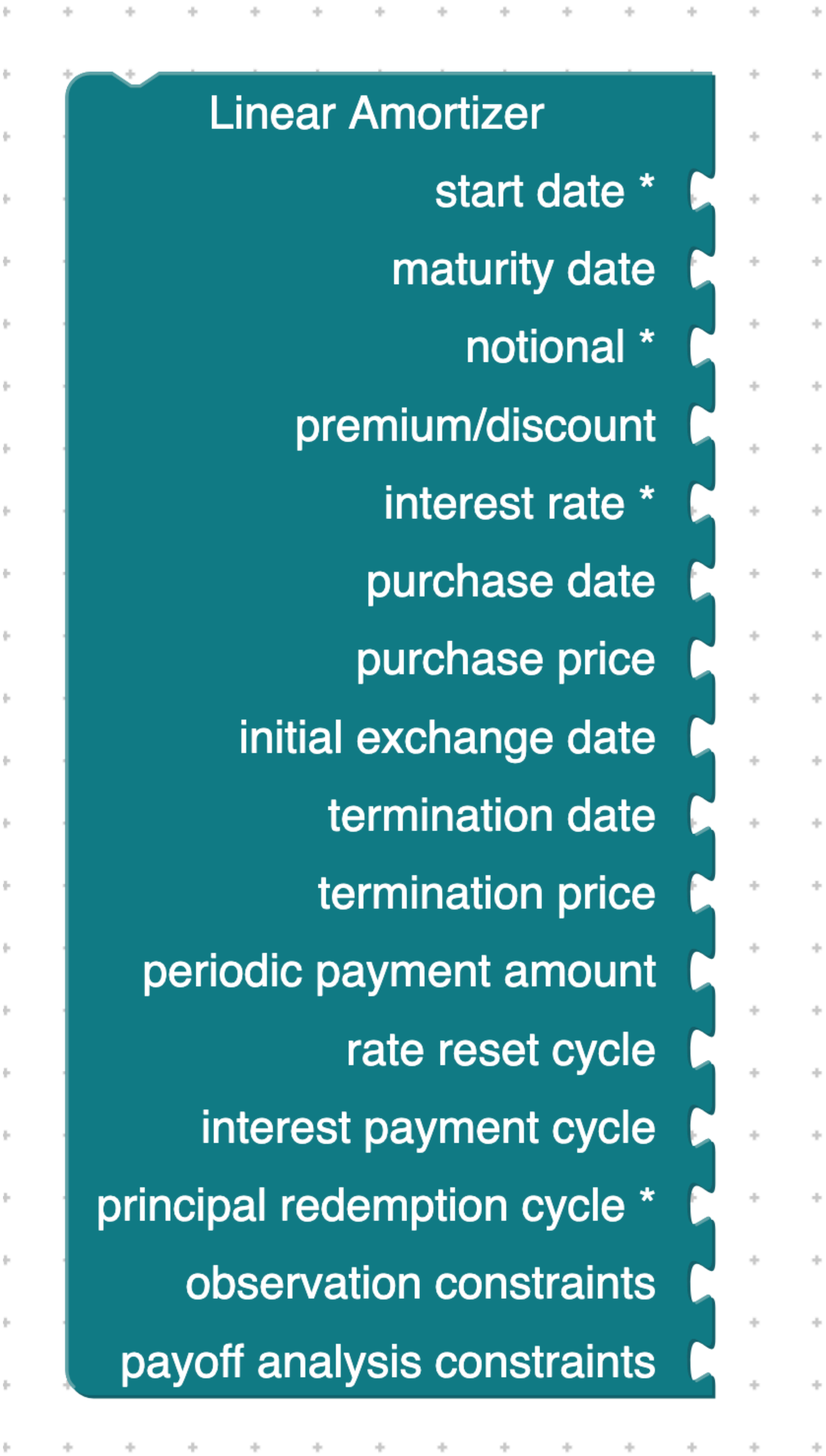
```
-- Definitions/ContractState.hs
data ContractStatePoly a b = ContractStatePoly
{
  tmd     :: b
  , nt    :: a
  , ipnr  :: a
  , ipac  :: a
  , feac  :: a
  , fac   :: a
  , nsc   :: a
  , isc   :: a
  , prf   :: ContractStatus
  , sd    :: b
  , prnxt :: a
  , ipcb  :: a
} deriving (Show)


-- Ops.hs
class ActusOps a where
   _min :: a -> a -> a
   _max :: a -> a -> a
   _zero :: a
   _one :: a

class ActusNum a where
   (+) :: a -> a -> a
   (-) :: a -> a -> a
   (*) :: a -> a -> a
   (/) :: a -> a -> a

class YearFractionOps a b where
   _y :: DCC -> a -> a -> a -> b

class DateOps a b where
   _lt :: a -> a -> b --returns pseudo-boolean

class RoleSignOps a where
   _r :: ContractRole -> a
```

Contract terms

Unified type of conditions to fit all kinds of ACTUS contracts.

Requires analysis of applicability of terms to contracts …

… and mechanism for combining the effect of multiple term instances.



Linear Amortizer

start date *

maturity date

notional *

premium/discount

interest rate *

purchase date

purchase price

initial exchange date

termination date

termination price

periodic payment amount

rate reset cycle

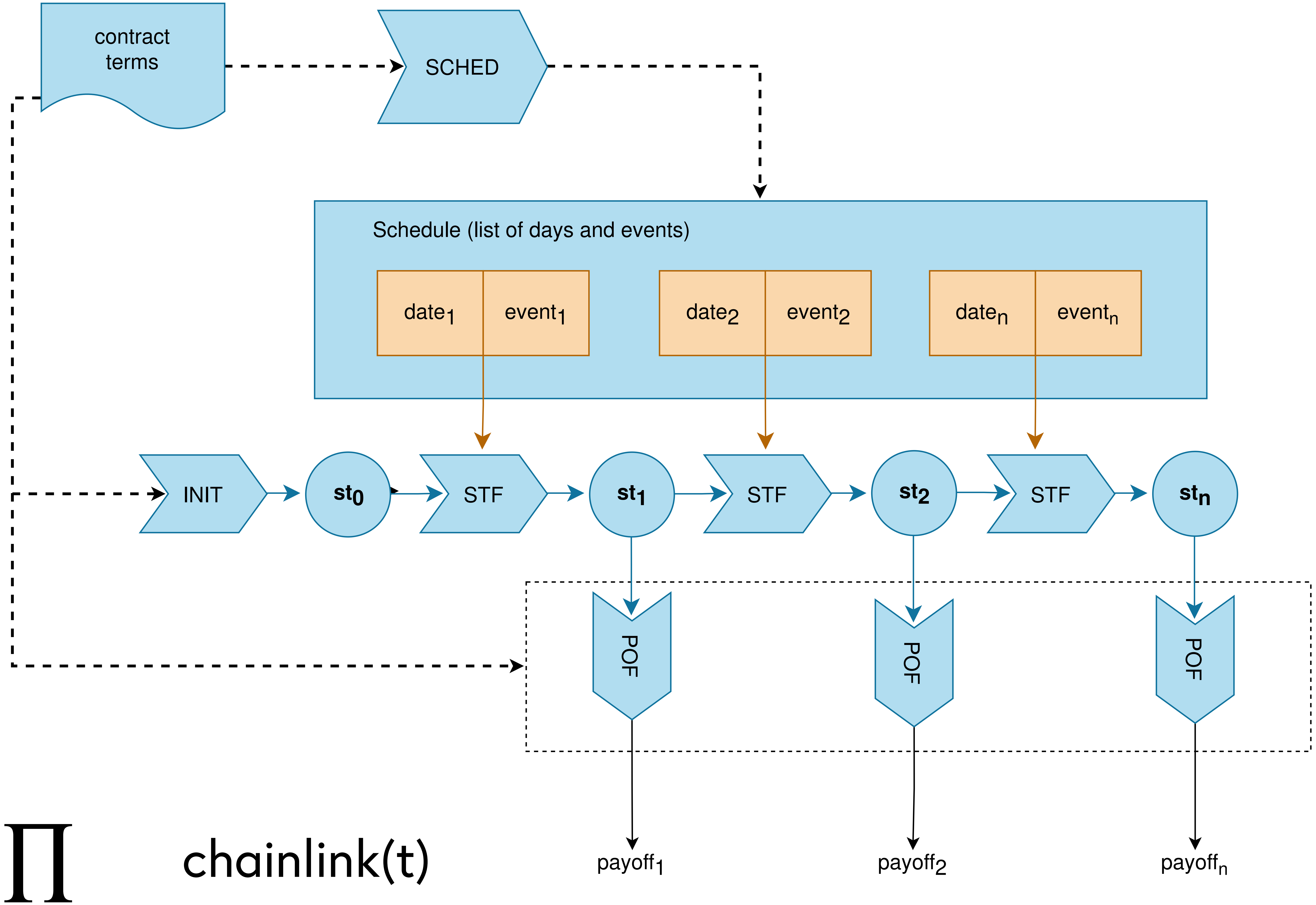
interest payment cycle

principal redemption cycle *

observation constraints

payoff analysis constraints

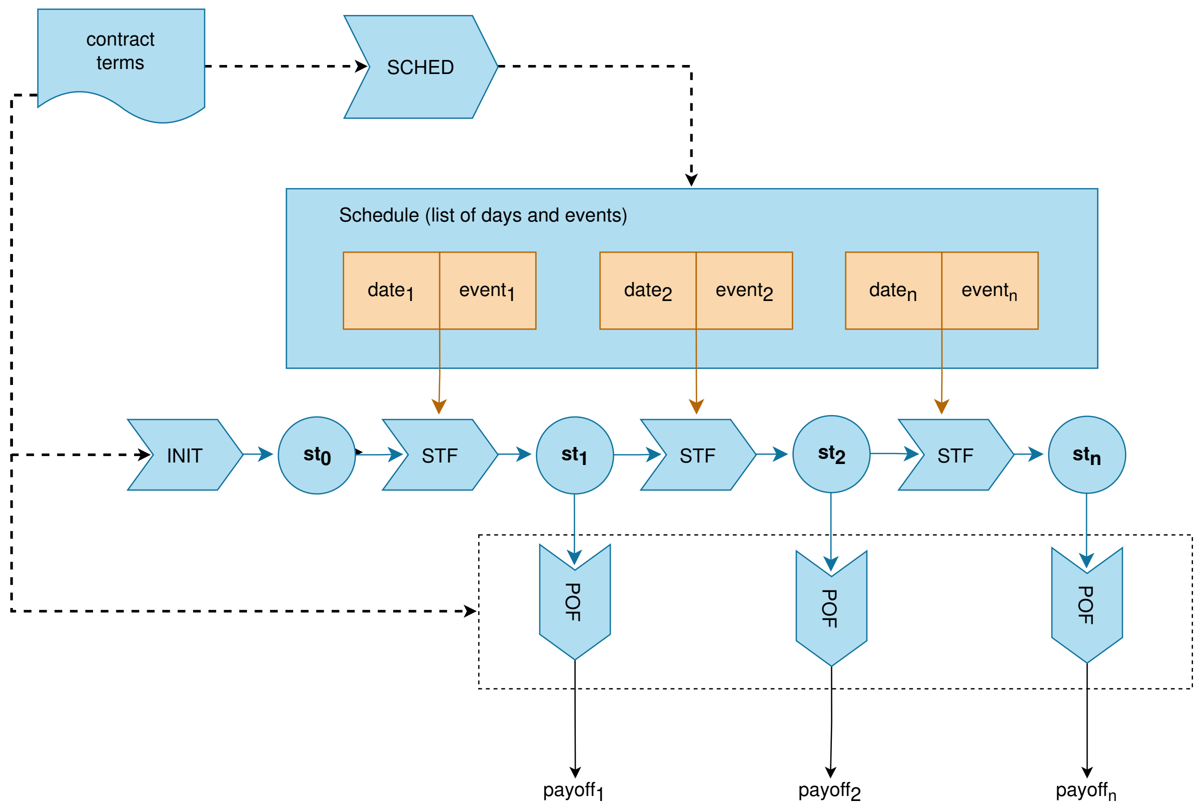
Contract generation



$$\text{contract}(ct) = \text{collaterals}(ct) \circ \text{INIT}(ct) \circ \prod_{t \in \text{SCHED(ct)}} \text{chainlink(t)}$$

$$\text{chainlink}(t) = \text{receiveData}(t) \circ \text{calculatePayoff(t)} \circ \text{processPayoff(t)}$$

Generation: under the hood



Different generation mechanisms for fixed and variable rates …
… pre-computed payments *vs* computation in the contract.

Language extension: conditional expressions.

Dealing with unbounded contracts.

Numbers: fixed-point *vs* integers.

Representing records in Marlowe.

Native tokens in Cardano

Represent ownership of roles in running contracts by *custom tokens*.

Possibility of *securitising* through multiple tokens per role.

Assurance

QuickCheck the Haskell implementation *vs* Java.

QuickCheck properties of contracts expressed via `Assert`.

SMT solving checks for potential failed payment: with c/exes.

ACTUS-specific: add a check for potential *auto refund* on `Close`.

For the future

Extend the coverage of ACTUS within ActusLabs.

ACTUS contracts onto Cardano itself: onto the Marlowe Dashboard.

Verification supported by the Isabelle Marlowe embedding.

Collisions of events, causality, hedging: all contracts have a *dual*.

https://alpha.marlowe.iohkdev.io