Tensile

Stretching Liquidity To Its Fullest Strength

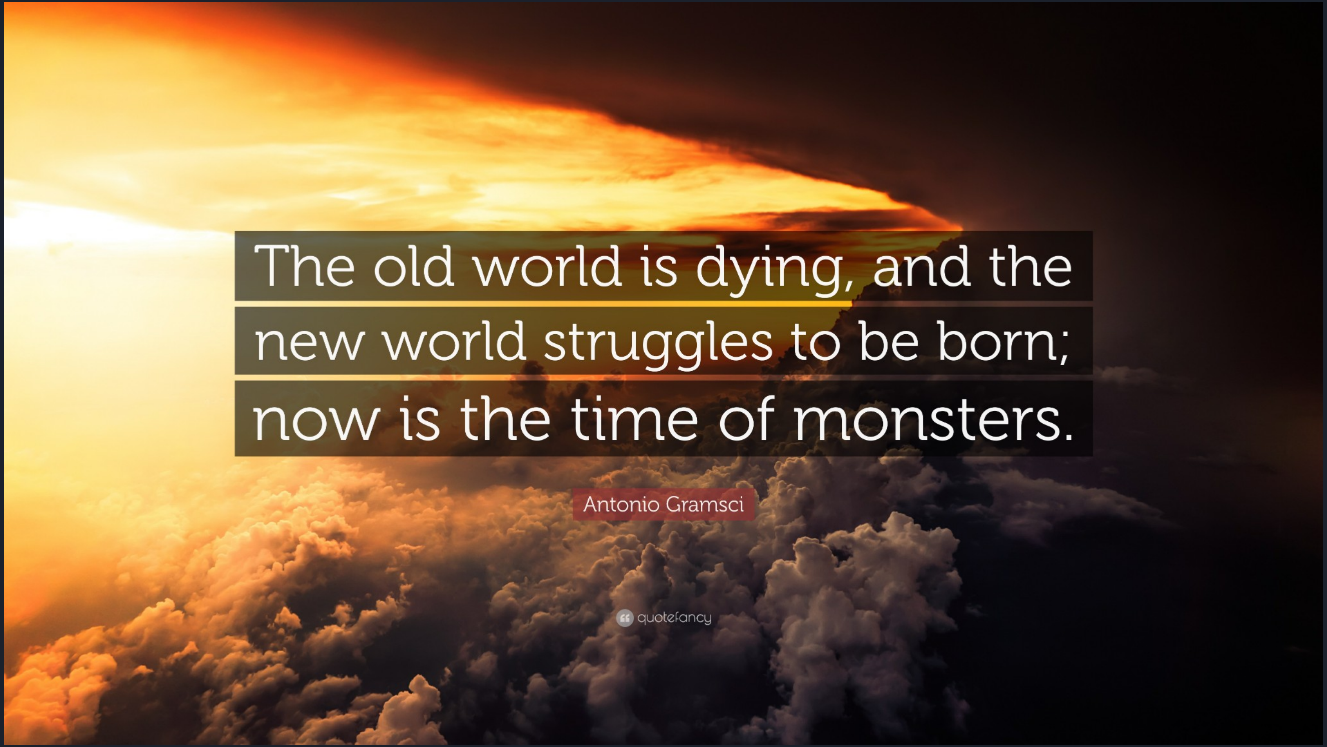
Decentralized Derivatives on Ergo

DeCo’s Ergoscript Course final presentation by
ahrnsetido and zuli

Presentation Outline

1. **Introduction**
2. **Competition**
3. **Tensile Platform**
4. **Future Contract**
5. **Future Code**

Introduction



> The old world is dying, and the new world struggles to be born; now is the time of monsters.

Antonio Gramsci

quotefancy

Introduction

Requirements for fair trading:

- neutral marketplace
- unstoppable market
- irreversible trade
- verifiable and transparent

Competition

Disadvantages of CEXs:

- centralized
- KYC
- limited OTC trading possibilities
- higher fees
- custodial/withdrawal difficulties

Advantages of CEXs:

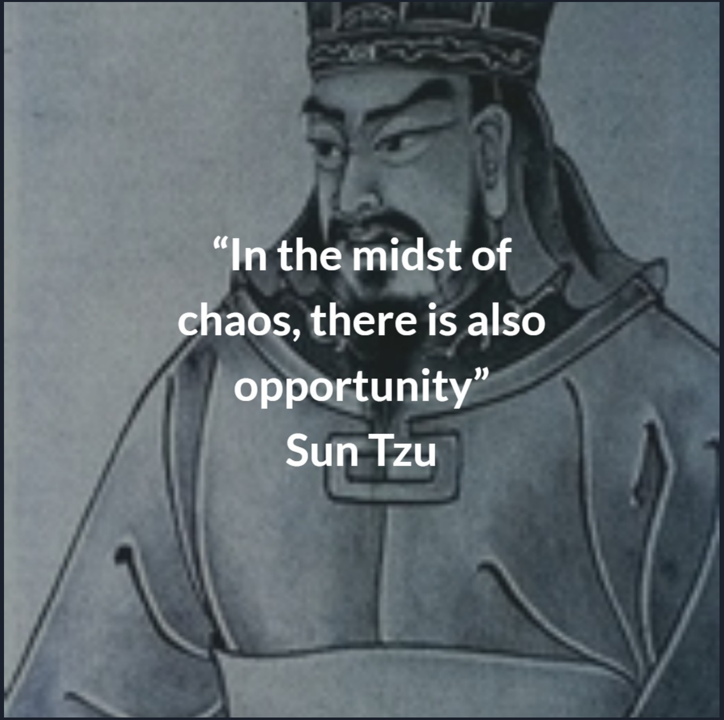
- more liquidity
- customer support
- easier access
- exotic derivatives
- low latency

Does it have to be like that?

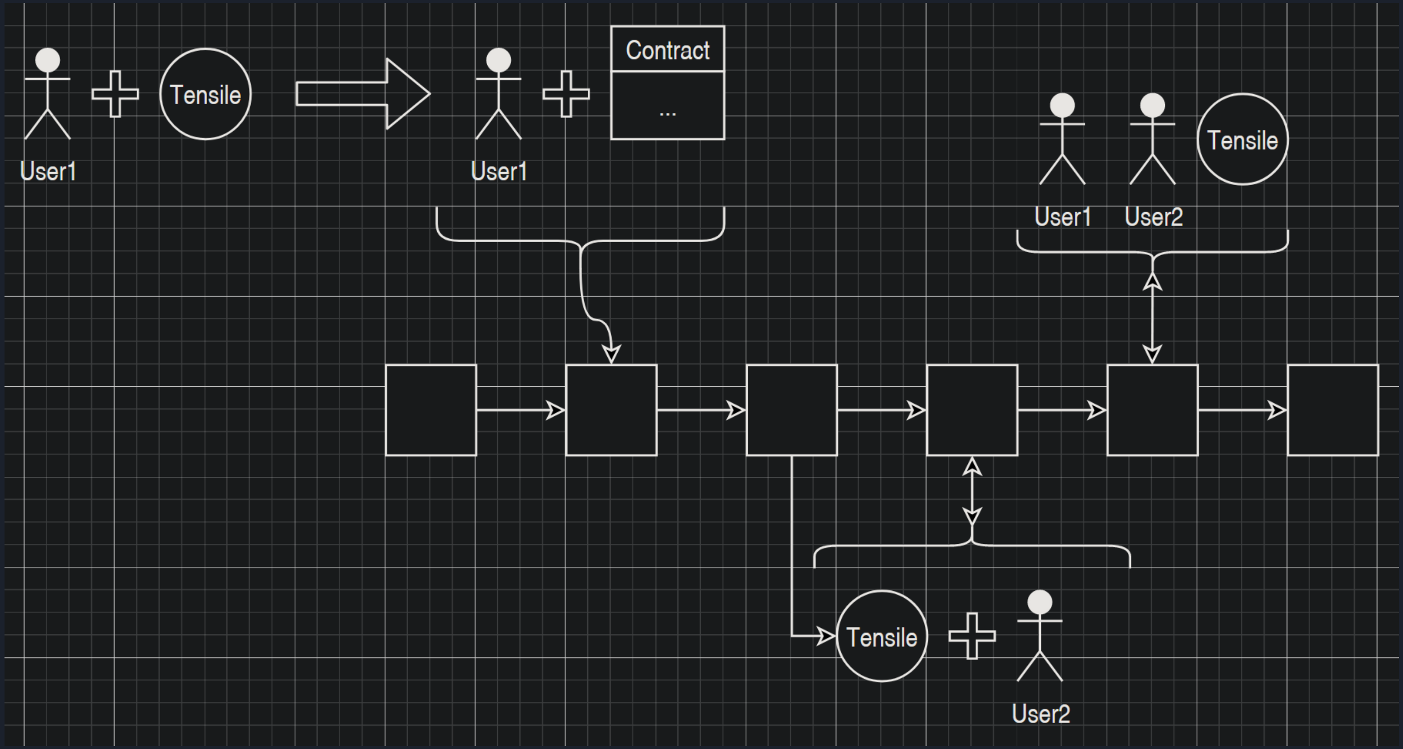
Tensile Platform

Why Tensile?

- decentralized
- non-custodial
- lower fees
- OTC and via LP
- irreversible and immutable trading



Tensile Platform

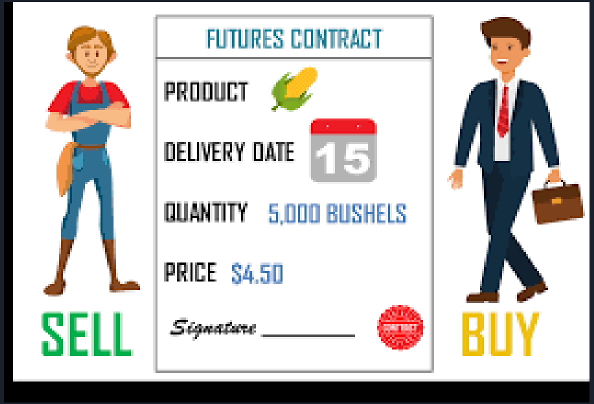


Applicability of derivatives?

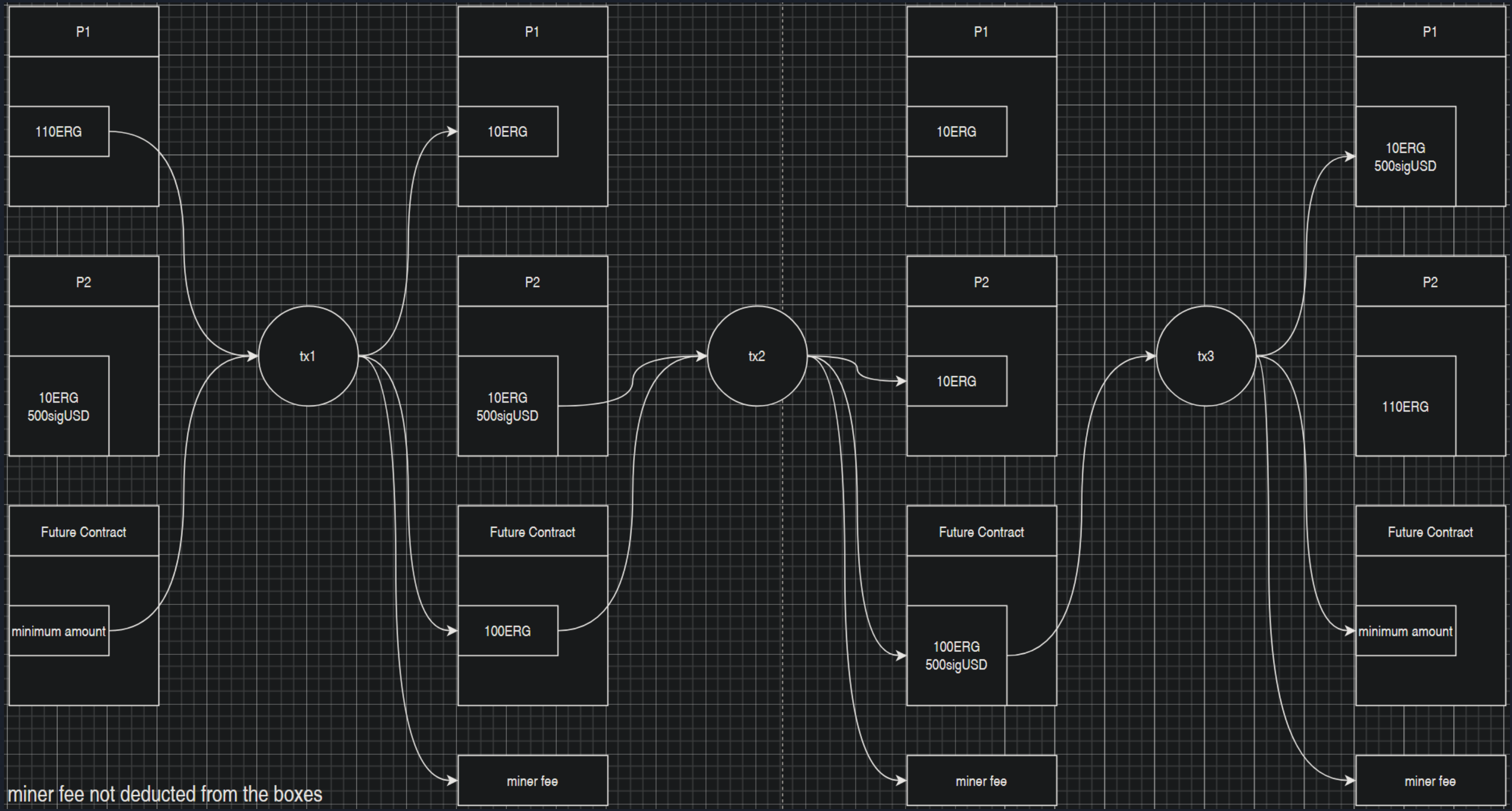
Future Contract

Explanation:

- financial contract
- price and date are predetermined
- traded on exchanges or OTC
- risk hedging or speculative use with leverage

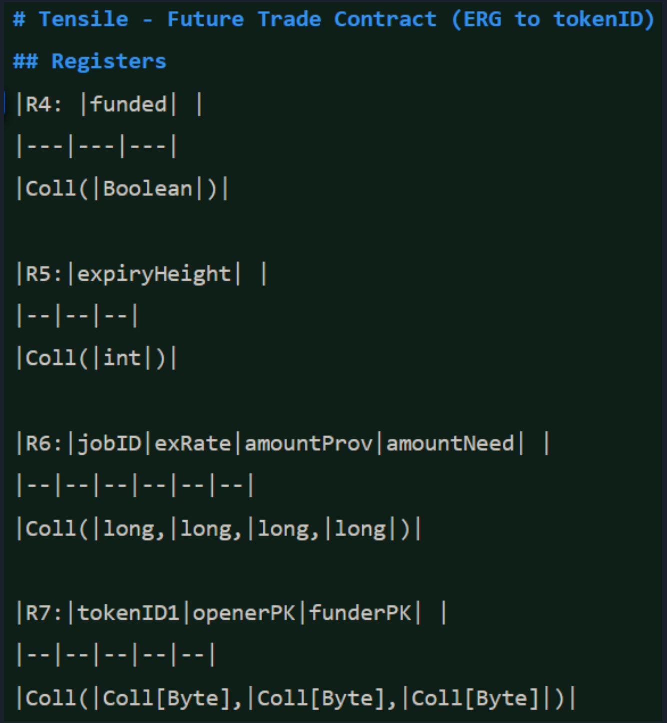


Future Contract

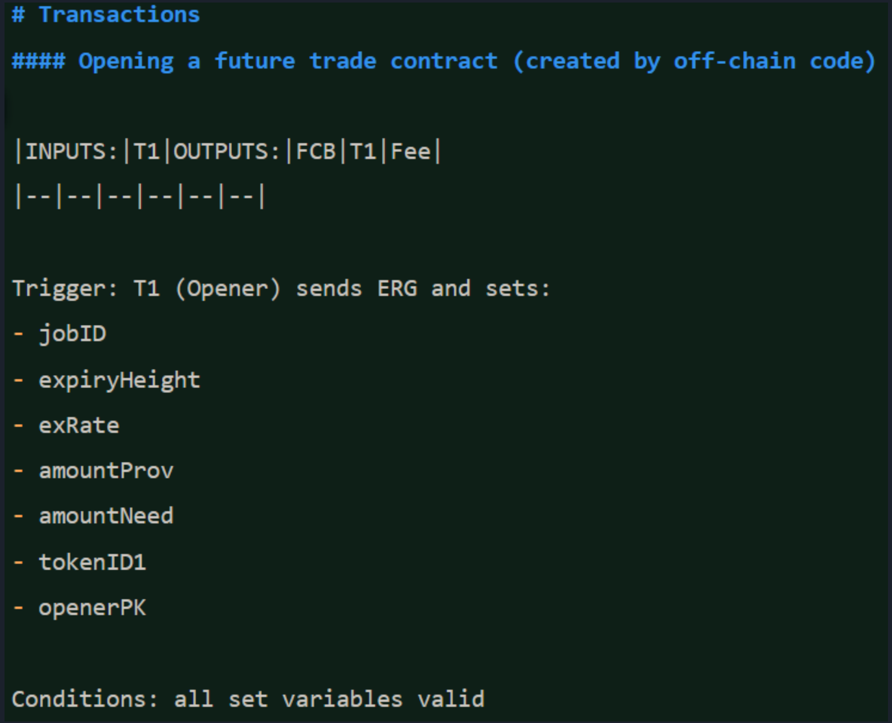


miner fee not deducted from the boxes

Future Code

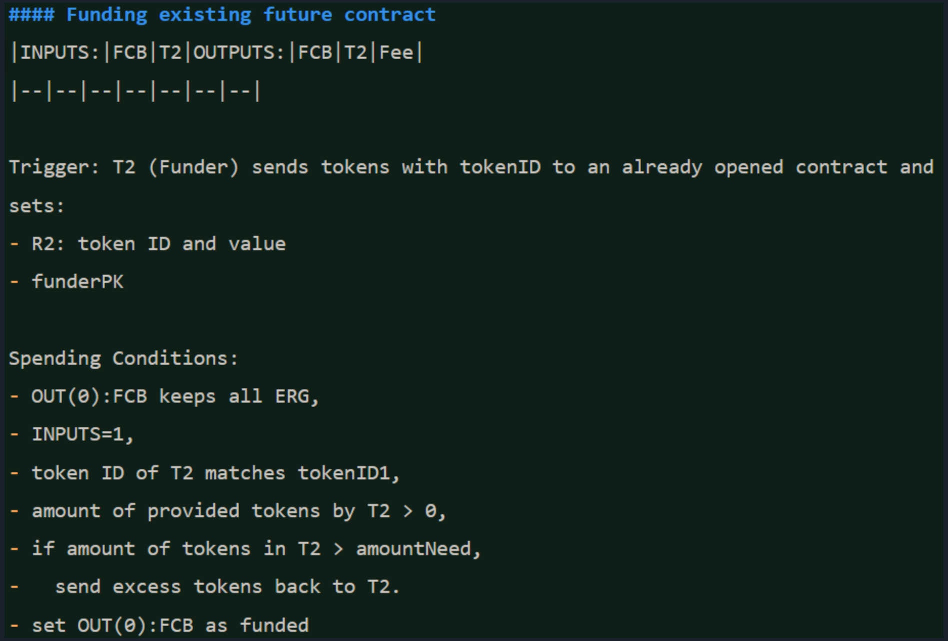


```
# Tensile - Future Trade Contract (ERG to tokenID)
## Registers
|R4: |funded| |
|---|---|---|
|Coll(|Boolean|)|

|R5:|expiryHeight| |
|--|--|--|
|Coll(|int|)|

|R6:|jobID|exRate|amountProv|amountNeed| |
|--|--|--|--|--|--|
|Coll(|long,|long,|long,|long|)|

|R7:|tokenID1|openerPK|funderPK| |
|--|--|--|--|--|
|Coll(|Coll[Byte],|Coll[Byte],|Coll[Byte]|)|
```

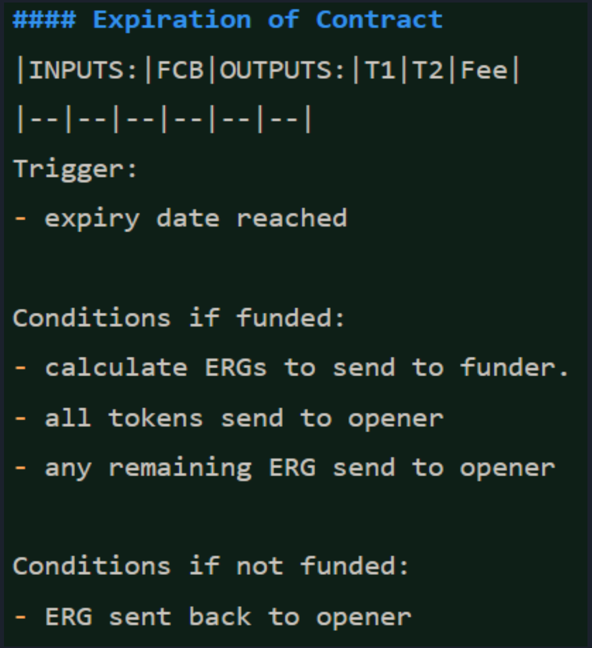


```
# Transactions
#### Opening a future trade contract (created by off-chain code)

|INPUTS:|T1|OUTPUTS:|FCB|T1|Fee|
|--|--|--|--|--|--|

Trigger: T1 (Opener) sends ERG and sets:
- jobID
- expiryHeight
- exRate
- amountProv
- amountNeed
- tokenID1
- openerPK

Conditions: all set variables valid
```

Future Code

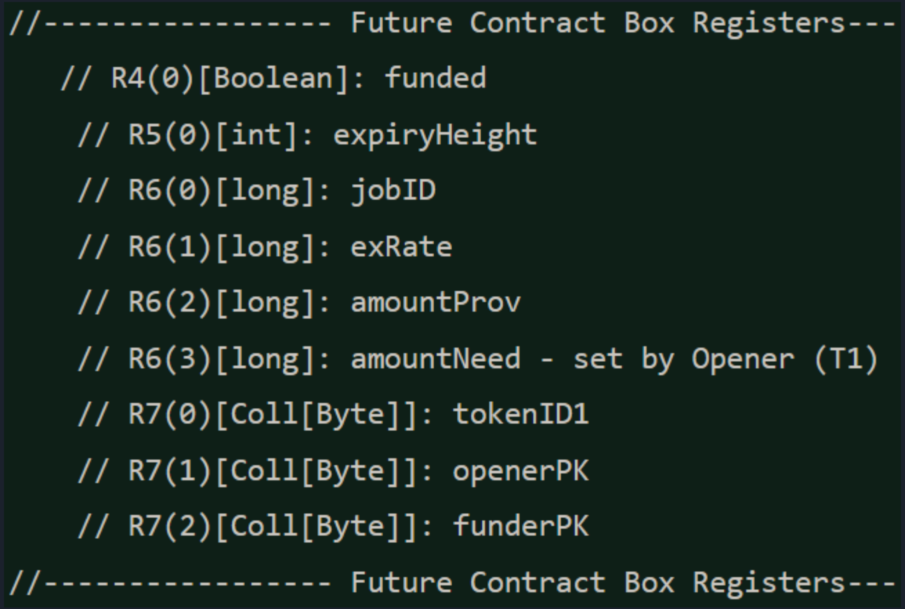


```
#### Funding existing future contract
|INPUTS:|FCB|T2|OUTPUTS:|FCB|T2|Fee|
|--|--|--|--|--|--|--|

Trigger: T2 (Funder) sends tokens with tokenID to an already opened contract and
sets:
- R2: token ID and value
- funderPK

Spending Conditions:
- OUT(0):FCB keeps all ERG,
- INPUTS=1,
- token ID of T2 matches tokenID1,
- amount of provided tokens by T2 > 0,
- if amount of tokens in T2 > amountNeed,
-   send excess tokens back to T2.
- set OUT(0):FCB as funded
```

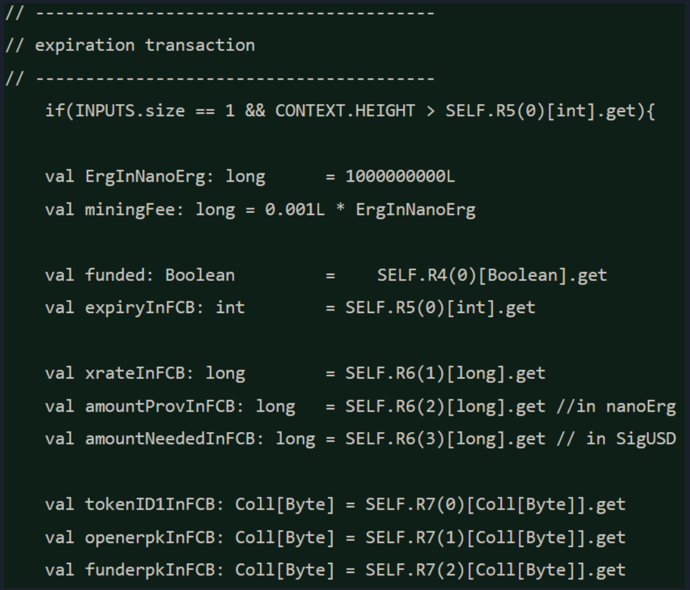


```
#### Expiration of Contract
|INPUTS:|FCB|OUTPUTS:|T1|T2|Fee|
|--|--|--|--|--|--|
Trigger:
- expiry date reached

Conditions if funded:
- calculate ERGs to send to funder.
- all tokens send to opener
- any remaining ERG send to opener

Conditions if not funded:
- ERG sent back to opener
```

Future Code

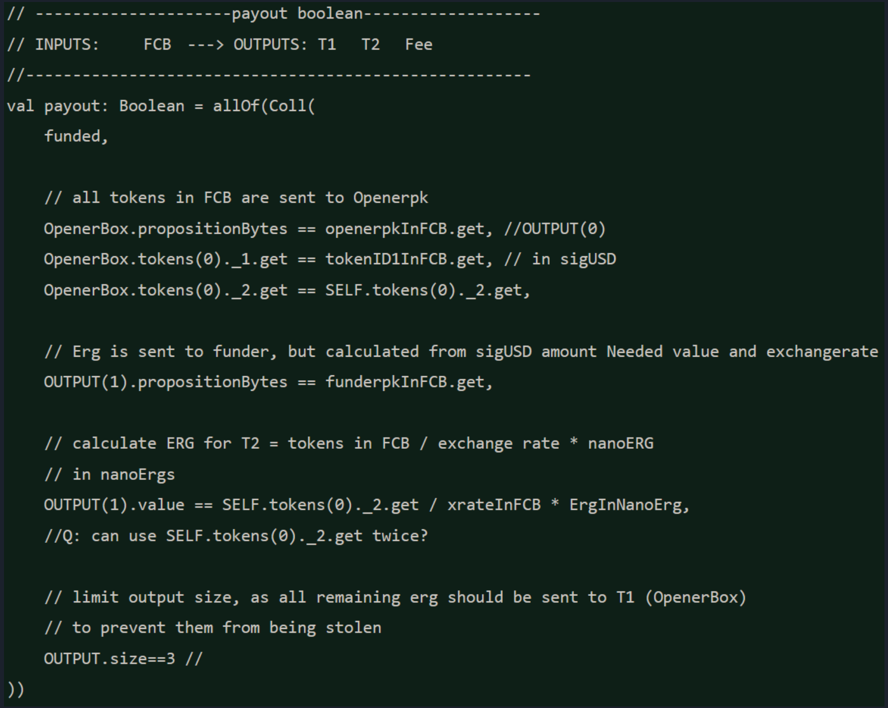


```
//----------------- Future Contract Box Registers---
   // R4(0)[Boolean]: funded
    // R5(0)[int]: expiryHeight
    // R6(0)[long]: jobID
    // R6(1)[long]: exRate
    // R6(2)[long]: amountProv
    // R6(3)[long]: amountNeed - set by Opener (T1)
    // R7(0)[Coll[Byte]]: tokenID1
    // R7(1)[Coll[Byte]]: openerPK
    // R7(2)[Coll[Byte]]: funderPK
//----------------- Future Contract Box Registers---
```

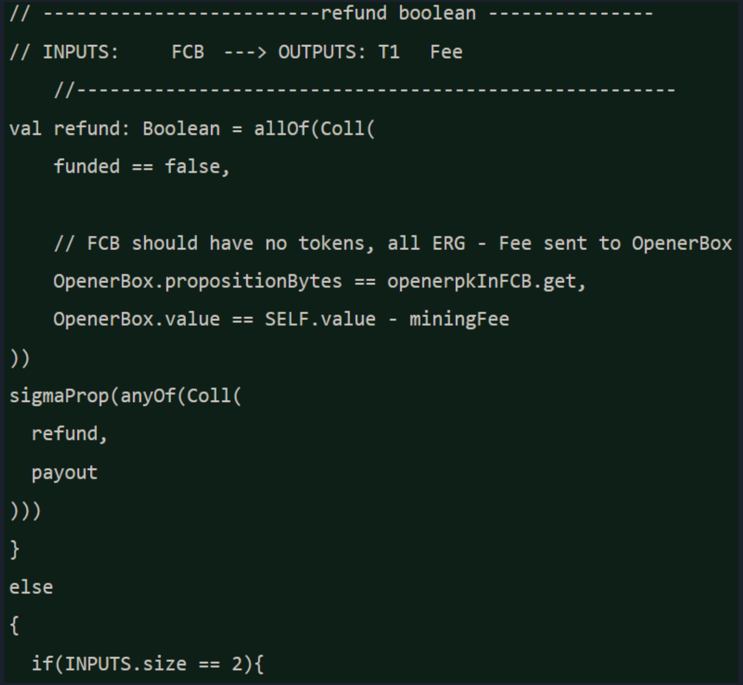


```
// ----------------------------------------
// expiration transaction
// ----------------------------------------
    if(INPUTS.size == 1 && CONTEXT.HEIGHT > SELF.R5(0)[int].get){

    val ErgInNanoErg: long      = 1000000000L
    val miningFee: long = 0.001L * ErgInNanoErg

    val funded: Boolean         =    SELF.R4(0)[Boolean].get
    val expiryInFCB: int        = SELF.R5(0)[int].get

    val xrateInFCB: long        = SELF.R6(1)[long].get
    val amountProvInFCB: long   = SELF.R6(2)[long].get //in nanoErg
    val amountNeededInFCB: long = SELF.R6(3)[long].get // in SigUSD

    val tokenID1InFCB: Coll[Byte] = SELF.R7(0)[Coll[Byte]].get
    val openerpkInFCB: Coll[Byte] = SELF.R7(1)[Coll[Byte]].get
    val funderpkInFCB: Coll[Byte] = SELF.R7(2)[Coll[Byte]].get
```

Future Code

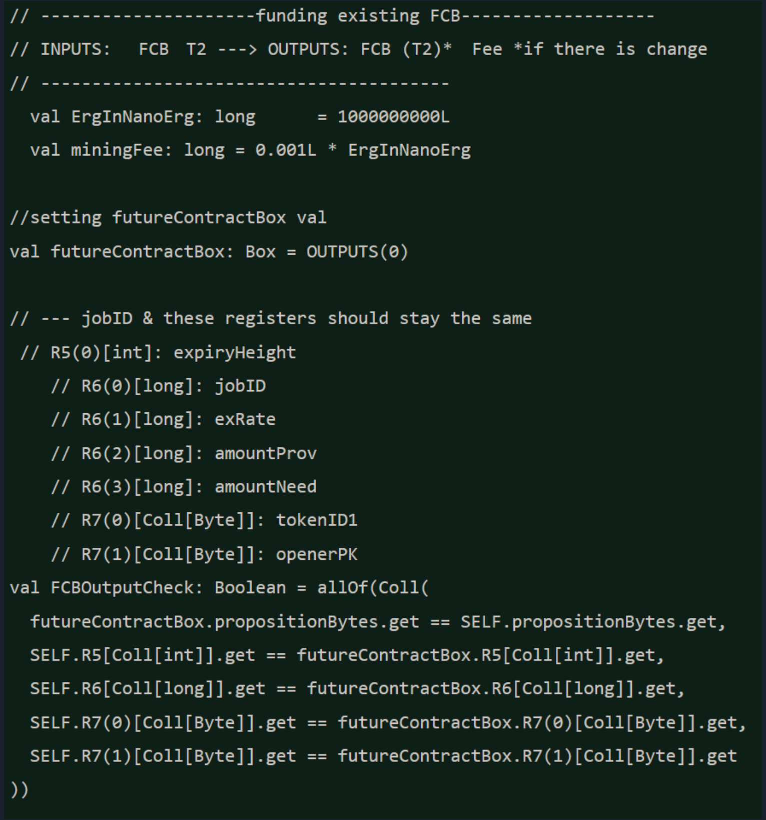


```
// --------------------payout boolean------------------
// INPUTS:     FCB  ---> OUTPUTS: T1   T2   Fee
//---------------------------------------------------
val payout: Boolean = allOf(Coll(
    funded,

    // all tokens in FCB are sent to Openerpk
    OpenerBox.propositionBytes == openerpkInFCB.get, //OUTPUT(0)
    OpenerBox.tokens(0)._1.get == tokenID1InFCB.get, // in sigUSD
    OpenerBox.tokens(0)._2.get == SELF.tokens(0)._2.get,

    // Erg is sent to funder, but calculated from sigUSD amount Needed value and exchangerate
    OUTPUT(1).propositionBytes == funderpkInFCB.get,

    // calculate ERG for T2 = tokens in FCB / exchange rate * nanoERG
    // in nanoErgs
    OUTPUT(1).value == SELF.tokens(0)._2.get / xrateInFCB * ErgInNanoErg,
    //Q: can use SELF.tokens(0)._2.get twice?

    // limit output size, as all remaining erg should be sent to T1 (OpenerBox)
    // to prevent them from being stolen
    OUTPUT.size==3 //
))
```

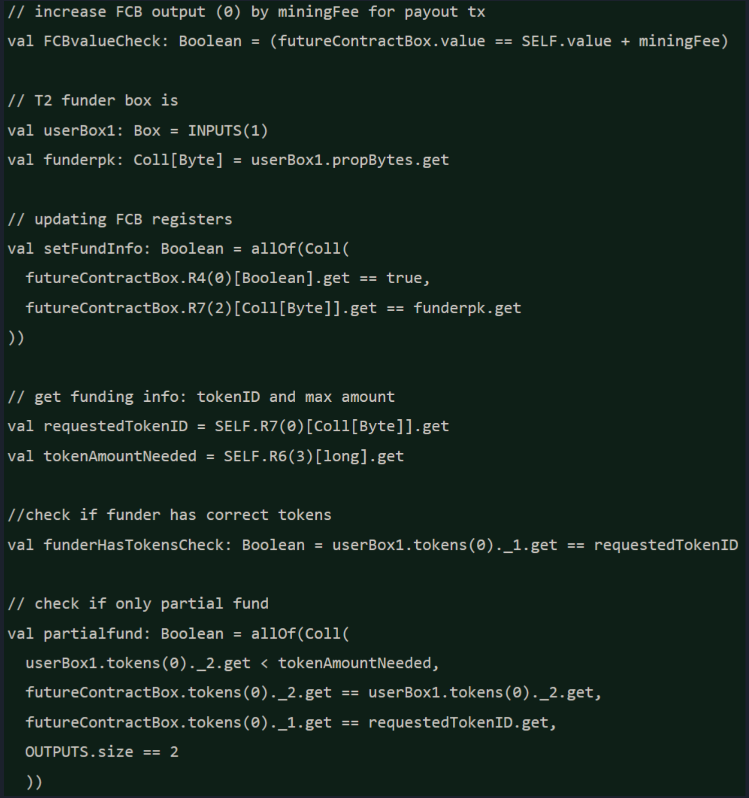


```
// -------------------------refund boolean ---------------
// INPUTS:     FCB  ---> OUTPUTS: T1   Fee
    //---------------------------------------------------
val refund: Boolean = allOf(Coll(
    funded == false,

    // FCB should have no tokens, all ERG - Fee sent to OpenerBox
    OpenerBox.propositionBytes == openerpkInFCB.get,
    OpenerBox.value == SELF.value - miningFee
))
sigmaProp(anyOf(Coll(
  refund,
  payout
)))
}
else
{
  if(INPUTS.size == 2){
```

Future Code

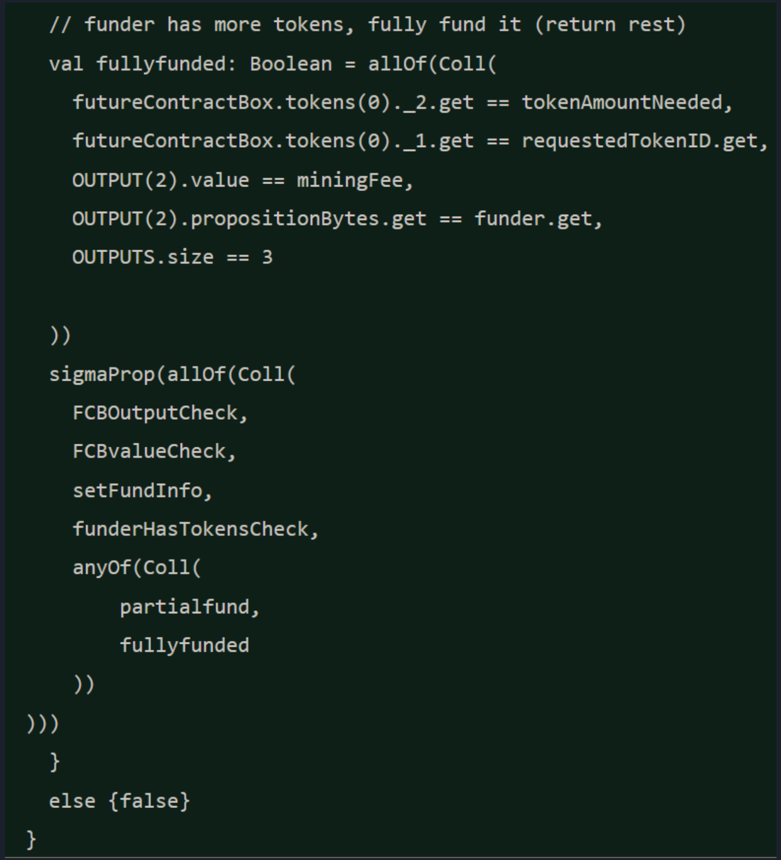


```
// --------------------funding existing FCB------------------
// INPUTS:   FCB  T2 ---> OUTPUTS: FCB (T2)*  Fee *if there is change
// ----------------------------------------
  val ErgInNanoErg: long      = 1000000000L
  val miningFee: long = 0.001L * ErgInNanoErg

//setting futureContractBox val
val futureContractBox: Box = OUTPUTS(0)

// --- jobID & these registers should stay the same
 // R5(0)[int]: expiryHeight
    // R6(0)[long]: jobID
    // R6(1)[long]: exRate
    // R6(2)[long]: amountProv
    // R6(3)[long]: amountNeed
    // R7(0)[Coll[Byte]]: tokenID1
    // R7(1)[Coll[Byte]]: openerPK
val FCBOutputCheck: Boolean = allOf(Coll(
  futureContractBox.propositionBytes.get == SELF.propositionBytes.get,
  SELF.R5[Coll[int]].get == futureContractBox.R5[Coll[int]].get,
  SELF.R6[Coll[long]].get == futureContractBox.R6[Coll[long]].get,
  SELF.R7(0)[Coll[Byte]].get == futureContractBox.R7(0)[Coll[Byte]].get,
  SELF.R7(1)[Coll[Byte]].get == futureContractBox.R7(1)[Coll[Byte]].get
))
```



```
// increase FCB output (0) by miningFee for payout tx
val FCBvalueCheck: Boolean = (futureContractBox.value == SELF.value + miningFee)

// T2 funder box is
val userBox1: Box = INPUTS(1)
val funderpk: Coll[Byte] = userBox1.propBytes.get

// updating FCB registers
val setFundInfo: Boolean = allOf(Coll(
  futureContractBox.R4(0)[Boolean].get == true,
  futureContractBox.R7(2)[Coll[Byte]].get == funderpk.get
))

// get funding info: tokenID and max amount
val requestedTokenID = SELF.R7(0)[Coll[Byte]].get
val tokenAmountNeeded = SELF.R6(3)[long].get

//check if funder has correct tokens
val funderHasTokensCheck: Boolean = userBox1.tokens(0)._1.get == requestedTokenID

// check if only partial fund
val partialfund: Boolean = allOf(Coll(
  userBox1.tokens(0)._2.get < tokenAmountNeeded,
  futureContractBox.tokens(0)._2.get == userBox1.tokens(0)._2.get,
  futureContractBox.tokens(0)._1.get == requestedTokenID.get,
  OUTPUTS.size == 2
  ))
```

Future Code



```
  // funder has more tokens, fully fund it (return rest)
  val fullyfunded: Boolean = allOf(Coll(
    futureContractBox.tokens(0)._2.get == tokenAmountNeeded,
    futureContractBox.tokens(0)._1.get == requestedTokenID.get,
    OUTPUT(2).value == miningFee,
    OUTPUT(2).propositionBytes.get == funder.get,
    OUTPUTS.size == 3

  ))
  sigmaProp(allOf(Coll(
    FCBOutputCheck,
    FCBvalueCheck,
    setFundInfo,
    funderHasTokensCheck,
    anyOf(Coll(
        partialfund,
        fullyfunded
    ))
)))
  }
  else {false}
}
```

Tensile

Stretching Liquidity To Its Fullest Strength

Decentralized Derivatives on Ergo

Thank you for your attention.

DeCo’s Ergoscript Course final presentation by ahrnsetido and zuli