# Plutus Pioneer Program

Second Cohort, Week 1

Jeehoon Kang [**GEST**]

2nd July, 2021

## 1 Background Knowledge

From the Plutus Documentation, the Plutus Platform is defined as the following:

> "The Plutus Platform is a platform for writing **applications** that interact with a **distributed ledger** featuring **scripting capabilities**"

As Plutus Pioneers, we are one of the first people on Earth that will be using that platform, developing smashing 'applications' that will run on our favourite distributed ledger called the 'Cardano Blockchain'. Cool! Right?

While that's all rainbows and marshmallows, let's first think about what that all means. Focussing on the boldfaced words in the definition above, we can explore this space in depth.

**Applications** - they're cool; they're the things you install on your phone, right? Somewhat correct! An application is a useful piece of opinionated and purposeful software that connects the user and the core logic/structures of a digital system.

In our case, we(Plutus application developers) are creating a piece of code that will allow users(Cardano end-users) to interact with the Cardano blockchain(the ledger), such that much more exciting actions (than just simple asset/token transfers) happen on that blockchain. We're creating that missing link between humans and blockchain!

Now that may sound like an impressive superpower, but as we all know: "With great power comes great responsibility". The developers of such applications are responsible for considering all the peripherals from user experience to synchronisation issues. Especially when valuable assets are at stake, that responsibility is magnified. This is where the platform steps in. The Plutus Platform attempts to provide a safe and expressive space for developers to manage all that. The last piece is on us; we still need to be on edge, constantly studying and testing our future world-changing services.

**Distributed Ledger** - "Alright, alright, I promise to build secure software for the good of all of us and to always be responsible. So, what services can we actually make? What is a ledger?" Knew you would ask that! Historically, ledgers originate from late medieval churches having a **book**(typically scripture) that lies **permanently** in a **open and shared** area. The main idea still carries on to the modern definition of ledgers. We record 'transactions'(typically of the transfers of valuable assets) between multiple parties, and everybody involved can verify and check the recordings.

The purpose of this so-called book-keeping is **consensus**. Such that no rogue statements like 'no I didn't', 'when did we ever agree on that?', or 'Come on! You told me so!' arises. All the parties involved with a ledger agrees what is written on the ledger is the unrefutable truth between them. (And thus, to add a bit more faith to it, when recording a 'transaction', it is typically 'signed' by all the parties being involved in that transaction.) This type of 'who did what when to whom' type of history-keeping naturally emanates widespread usage in various fields.

- Simple book-keeping such as tracking money transfers, organising academic records, verifying immunisation records, product version management etc
- Extended applications such as on/offline social networking, game state tracking, advertising host-client matching, copyright and ownership checking, shared computing cluster management, voting and on-and-on.

Up until this point, we've only discussed what a ledger is, but let's dig deeper and see what the Cardano Blockchain[our favourite ledger ;)] is all about.

The Cardano Blockchain is a digital **distributed** ledger. Here the word distributed implies **'replicated, shared, and synchronised'**. This makes reaching consensus a bit harder than a central organisation (like the church on the historical example above) managing the entire ledger. Still, if such consensus can be achieved, **trustlessness** comes to town, and this can provide numerous (hopefully) positive socioeconomic effects. Its like people having a bible on their own in the church example. You don't have to trust that the church hasn't tampered with the single source of truth, and can independently act upon your own copy. As such it is also possible to quickly confirm someone else's claim with your own knowledge. You also don't need to provide personal information to the church in order to access it, etc.

The Cardano blockchain manages that consensus with a philosophy called 'Proof of Stake', which at its simplest means 'temporarily transferring power (to write on the ledger) over to the party that has the most to lose if things go wrong on the ledger', but there are layers and layers into this. Also, numerous other consensus algorithms, so I beg the reader to read more about them! Here's a link to the Cardano website describing the Ouroboros protocol.

By now, I hope that we're all on the same page (Ha! consensus!) and that we're all filled with exciting ideas that apply 'distributed ledgers' to crucial human problems, needs, and want!

**Scripting Capabilities** - This naturally leads on to the central part of the Plutus platform: Scripting Capabilities! Before we delve into this topic, let's think about some typical ways a distributed ledger is composed.

**Account based** - This is pretty self-explanatory. The ledger records a **list of accounts** of all the players involved and the **balance per account**. Transactions require the global state of the accounts and their balances and output an increase/decrease of balance for the accounts involved. This is how the current Ethereum blockchain is run. It helps with the expressiveness and programmability of the system, but shared mutual states create side effects. It will be hard for the developer to manage everything, especially the unexpected or unwanted side effects.

**UTxO based** - UTxO stands for 'Unspent Transaction Outputs'. Here the unit of transaction isn't the balance of accounts. It's the UTxOs that can be thought of as a **cake with a nametag**. The nametag simply represents who's cake it is (and one person can have many cakes too! mmm!), and the cake itself is some money/token. The cake must be the unit of transaction, and we do not cut the cake into pieces and give a piece to the transaction. The transaction cuts/merges/or-whatever-it-wants-to-with the cake to create new cakes. A transaction in this sense takes in a cake (or several cakes), and slices the pieces inside of it according to the transaction data, and re-assigns the pieces either within the cake or to a different cake(s) and assigns new nametags to the produced cakes. As such, it's not the balance of accounts that are updated, but just the size and nametag of the cakes. Cake(s) goes in(is spent) and new (unspent) Cake(s) come out. This means that the transaction only has to deal with the input cake(s) instead of the global state of accounts and balances (Hmm?! I smell functional, I smell no side-effects, I smell HASKELL!!). At their simplest, Bitcoin and Cardano use this UTxO model as their base ledger system.

Now, Cardano here extends the UTxO model into something called an Extended Unspent Transaction Output model (EUTxO). It is this 'Extension' that provides the 'scripting capabilities' to the Cardano blockchain. **The scripts are there to define and construct the complex transactions and provide 'validation' upon conditions on the consumption of UTxOs by those transactions.**

By adding this additional capability, we can deploy so-called **'smart contracts'**, that adds interesting logic such as 'if', 'while', and so on to the "'who did what when to whom' type of history-keeping" that was discussed previously. Isn't that awesome?! There's got to be so much more we could do with that!!

This awesomeness is what we'll be working on as Plutus Pioneers. What type of interesting logic can we develop using that EUTxO model?! How do we do so?!

# 2 Week 1 - The EUTxO model and an English Auction

The week 1 lecture on the Plutus Pioneer Program (Second Cohort) kicks off by explaining more about the 'scripting capabilities' discussed in the section above.

## 2.1 UTxOs

Unspent Transaction Outputs(UTxO) are defined as

> "Transaction **outputs from previous transactions** on the blockchain that have **not yet been spent**"

Each transaction can take in an arbitrary number of inputs and produce an arbitrary number of outputs. Here the input and outputs are the UTxOs(The cakes ;)), and they are consumed as a whole(You don't give a transaction a half-eaten/piece of a cake!). To check whether a transaction can spend a UTxO or not, digital signatures(of the input owners) are added to the transaction.

So when you're trying to send someone 10 precious ADA, and you have one UTxO that says 100 ADA under your nametag, you don't split that UTxO into 2 UTxOs of 90 ADA and 10 ADA, and only give the 10 ADA UTxO to the transaction. You provide the full UTxO that contains the 100 ADA under your name. The transaction itself will produce two UTxOs, one with 90 ADA(as change) under your nametag and another one with 10 ADA under the recipient's nametag.

Likewise, if you have two UTxOs from two different transactions, one with 10 ADA and the other with 20 ADA, both with your nametag, and you need to spend 30 ADA, you don't merge it into 1 big UTxO of 30 ADA and give it to the transaction. You give two full UTxOs of 10 ADA and 20 ADA, and the transaction will handle the main logic.

That makes things clean and straightforward, and thus we can simply think that the sum of the inputs to a transaction is equal to the sum of the output of the transaction(The Conservation of Value!). The transaction itself is then just a re-distribution of assets. (But wait! There are exceptions!)

The two exceptions happen when we introduce:

1. Transaction Fees: When a transaction is written into a block and then added to the ledger(Cardano Blockchain), a small fee(depending on the transaction complexity) is required to incentivise the 'block writing' of the network. This, therefore, means that normally transaction inputs are slightly larger than their outputs.

2. Native Tokens: When minting(creating a token) or burning(putting it out of circulation) a token, the sum of input and outputs may be different. You either have more outputs(minting a token) or more inputs (burning a token)

### 2.2 (E)UTxOs

To borrow the expression, EUTxOs are UTxOs on steroids, and by steroids, I mean 'smart contracts'. Typically, the validation of the consumption of an UTxO relies on its digital signature checking. To elaborate, the UTxO being used as input is attached with a function called a 'validator' with the spender's public key address. The transaction input proves its eligibility with the 'redeemer', which holds a hash made with the authorised spender's private key. The validator receives the redeemer information, checks if the key and hash information matches, and thus confirms whether the specified UTxO can be spent at that transaction input or not.

But with EUTxOs, on top of that digital signature, we can **add arbitrary logic** (compiled as Plutus core). Instead of the validator and the redeemer holding addresses of public keys and hash, it contains addresses to **'Scripts'** that contain arbitrary logic. It will be this 'script' that accesses the Redeemer(+ Datum which will be explained later) of the transaction input and the transaction context to validate the associated UTxO as a valid input or not.

### 2.3 Context of the Scripts

We all know the Goldilocks story, and the same analogy can be shown in the context of the scripts in three major blockchains currently in use. Let's see whether our favourite blockchain Cardano is too hot, too cold, or just right!

- **Bitcoin** The Bitcoin blockchain also has (not-so) smart contracts written in Bitcoin Scripts. These scripts only have access to the Redeemer of the input. Within the scale of the current transaction and up to the full blockchain state, Bitcoin's UTxO scripts have

minimal context (of just the redeemer of the associated transaction input). Straightforward and clean, but its going to be hard to express anything exciting.

- **Ethereum** The scripts on the Ethereum blockchain sees the whole shared state of the blockchain. This allows it to be very expressive, as it can incorporate various states and conditions of the blockchain. Yet, non-determinism(side-effects) and concurrency is an issue (which might lead to security problems).

  During the time between a transaction being constructed and being incorporated into the blockchain, a lot of stuff can happen concurrently. The script's context is the whole blockchain state, and it won't be able to predict everything that happens before the blockchain state updates. Thus whether or not the transaction eventually fails or not, a gas fee(transaction fee) is always required(as you only get to know whether the transaction failed until a block is added to the blockchain).

- **Cardano** is in the 'just right' zone between Bitcoin scripts and Ethereum scripts. It holds on to the semantic simplicity of the UTxO model yet also doesn't give up on the expressiveness shown by Ethereum. This is possible with the following ideas.

  1. The context is at the transaction level(not the blockchain level), where the scripts have information not limited to the spender's input but all the inputs and outputs of the transaction. This limited scope helps with analysing the script, predicting what it will do, etc.
  2. The context also includes a **'Datum'**, which is an arbitrary piece of data associated with the UTxO. This allows for historical and exterior contexts present on the blockchain or elsewhere to be incorporated into the scripting. On top of the Redeemer, the Datum is added as a part of the validation.

  Thus on Cardano, it is possible to check whether the transaction will validate or not before going on-chain. If someone else has already consumed the output, the transaction that was going to spend it will fail before being processed into a block, and thus you won't need to pay transaction fees for failed transactions. I think we found our 'just right' porridge!

Alright, so what's the recipe to make this 'just right' porridge? Who is responsible for providing the Redeemer, Datum, and the Validator(Script)? The answer is the spending transaction.

Now, some of the more pedantic readers here might have noticed that in section 2.2, it is explained that the output is associated with a Script and that the Redeemer is the one given by the spending transaction input. Conceptually, that is correct, as the validator is meant to be part of the output. However, the actual scripts and datum value are provided as part of the spending transaction inputs. The output will typically only include the address/hash of the script/datum such that the script and Datum can be specified. This helps to reduce the memory footprints of the EUTxOs (as they only include the hash, and the transaction itself provides all the Script, Redeemer, Datum), which can help with the rapid access of them while validating for transactions, etc.

But wouldn't that mean that you need to *know* the datum/script in advance to include it in your transaction to spend a particular input? Yes, that is right, but sometimes that isn't always possible, or it might be problematic for certain types of applications. That is why you can optionally include the entire Datum in the EUTxO that will be consumed. Other off-chain methods that haven't been explained in week 1 can be used as well, and hopefully, those will be uncovered during the following weeks.

Obviously, formalising such a concept is not bound to a specific programming language, but the Plutus Pioneer Program, as the name suggests, will focus on the usage of Plutus and its mother-language Haskell.

To gain a more formal view about the EUTxO model, I recommend the paper originally presenting the EUTXO model. It's only 15 pages long, and the vocabulary or expressions not too daunting, so yeah, please do have a go at it!

### 2.4 English Auction

First off, an English Auction is the type of auction you see a lot in films. There's something valuable on auction and a minimum 'bid' on that item, which is the smallest sum of money you can bid on the item to 'win' over the item. People shout out their bid, which has to be monotonically ascending for the auctioneer to accept it. While you have the highest bid on the item (a standing bid), you are the temporary winner of the item. That changes when your bid is displaced by a higher one, and you're no longer the winner. This goes on and on for a given set of time, and the standing bid at the end of the timeframe is the final winner. How do we design such a scheme on the Cardano Blockchain with the EUTxO design above, and how would it work?

As described above, we can first parameterise what attributes an 'Auction' has to hold. Those would be:

- The owner (of the item on auction)
- The item itself (an NFT in our example)
- The minimum bid
- The current highest bid
- The deadline

On that, we can define the sort of transactions that can happen on it. Namely:

- **Start and Auction** - An auction is created with the attributes above. All of the attributes have to exist in a valid sense(you can't start an auction on an item that is not there, the minimum bid has to be positive, the deadline has to be beyond the current time).
- **Bid on the Auction** - Someone can bid on a started auction. This transaction will be constrained by the minimum bid attribute, the highest bid attribute, and the deadline(you need to bid higher(than the minimum or the previous highest bid) and bid before the deadline).
- **Close the Auction** - Once the given deadline is over, the auction has to be closed, and the item goes to the winner + the highest bid goes to the owner. This is constrained only by the deadline(you can only close the auction after the deadline).

When developing an application, it is helpful to organise a schematic like the above, explore what data types are required, what sort of transactions are necessary, what constraints they have, validation conditions, etc.

Anywhoos, let's try delving into the example.

1. Alice wants to auction off her NFT(Non-Fungible Token), which is a native token living on the Cardano network, that 'exists exactly once' as Lars puts it!

2. So first, Alice starts the auction by creating a UTxO with the NFT as the value (typically, you cannot only have a token as a value for the UTxO, you need ADA too) at the Script address holding the Auction Script. The auction script holds the minimum bid parameter(let's say 100 ADA) and the deadline. The 'highest bid' parameter will be carried forward by the Datum, which currently is there is none.

3. Bob wants to bid for the NFT, and bids the minimum bid of 100 ADA by creating a transaction with 2 inputs and 1 output

   Here the inputs will be the **Auction UTxO**, and **Bob's UTxOs holding Bob's ADA**. The transaction itself will return the **updated Auction script** with the value being updated to the NFT and the 100ADA bid, and the Datum(holding data about the highest bid) updated to hold the information that "The highest bid is 100 ADA under Bob's name".

   This transaction's Redeemer will hold data for its validation of using the Auction UTxO as input. This data will include information about the bid, and the validator provided by the auction script can check whether or not it can be consumed by the transaction or not using that Redeemer. Here it can check certain conditions listed above, such as the bid happening before the deadline, the bid being high enough, and that the correct input and output are there for the transaction given as context(e.g. Datum updated correctly? etc.).

4. At his point, Charlie wants to bid higher and bids 200ADA by creating a transaction again with Charlie's UTxOs containing Charlie's ADA and the updated Auction UTxO from before.

   As Charlie's bid is higher than Bob's, it will output the updated Auction UTxO with Charlie's bid + NFT as value, and the Datum changing to information on Charlie's bid. On top of this new auction UTxO, another output is made that returns Bob's 100 ADA bid back.

5. Once the deadline is over, the auction is set, and either Alice(who will get the 200 ADA) or Charlie(who will get the NFT) who are incentivised the close the auction will do so, and that closing transaction does the following:

   Just the auction UTxO goes into the closing transaction, and two outputs, one to Alice with the ADA and one to Charlie with the NFT, are produced. Again the Redeemer and the validator will check if the auction UTxO can be consumed by the closing transaction by checking if the deadline was reached and that the outputs all make sense.

   As a side note, this closing transaction is necessary. UTxOs are just passive data, and unless an outside transaction is presented to the blockchain, the state will not change. **Change is triggered by a transaction**, so if you'd like for an automatic thing to happen after a while, automate the creation of the transaction on the wallet level, not in the validator/ transaction script.

6. If no bid was present on Alice's Auction, then after the deadline, Alice can close the auction with the closing transaction, which will consume the auction UTxO, and produce a UTxO with the NFT as value back to Alice.

### 2.5 So where do we code all that?

Wow! We've come so far from discussing ledger-based applications to EUTXOs, and even an English Auction example! With all that, I'd say we're ready to start looking at code (which we will from week 2! woop woop!). Plutus has two places where the codes live, namely **on-chain** and **off-chain** code.

**On-chain code** - These are the Scripts from the UTxO models. The so-called 'validators'. Its hash is given by the UTxO, and the transaction input provides the actual compiled Plutus code for it. When a Cardano node receives a transaction, it validates the transaction with this on-chain code before accepting it into its mempool, then eventually into a block. For each transaction's input, the corresponding script on the script address is executed, and it must succeed. If it doesn't, it is deemed an invalid transaction. The actual 'code' is written in Haskell but compiled down to **"Plutus Core"**

**Off-chain code** - This is the code that runs in the wallet (not the blockchain!). These construct suitable transactions that will unlock the output to be consumed as input. This is the main logic that can write various different applications. It has to define the input and outputs of a transaction and form suitable smart contracts used by your application.

As you can see, it's a **symbiotic relationship between the two types of code**. The off-chain code actively constructs transactions, and the on-chain code provides the validation of UTxO as the input given the off-chain context.

What really reinforces this symbiosis is that both on and off-chain are written in Haskell (our favourite functional programming language! yay!). This allows for sharing code between the on and off chains, which will really play a part when dealing with state machines (again, a concept to come up in future lectures!).

## 3 Closing

That roughly closes the notes and thoughts for week 1 of the Plutus Pioneer Program(second cohort). I haven't really talked about the setup of the Plutus Playground (use **nix** btw, its awesome!) or simulating the English Auction with it, nor the on/off-chain code that goes into it, but hopefully, this still serves as a good conceptual introduction to the course. I hope to start an engaging conversation on this topic, and like what our favourite distributed ledger tries to do, reach consensus among our understanding and usage of the Plutus Platform. Always remember, we're changing the world slowly but surely!! Share the love! ♡

ESSE QUAM VIDERI
J Kang.