NEVER Phase II Auctions

Resources and Contacts

- **Telegram:** @laugan
- **Wallet:** 0:fd080fb5fc9266226ec59b062f0cdde85c818ef1d4ac4939804ee7616ec352f4
- **Github:** https://github.com/deplant/never2-core

Submission to NEVER Phase II Auctions Contest.

Features

- Implementation-agnostic Auction & Bank (AuctionManager)
- Implementations of all Bid types (TIP3, CurrencyCollections, EVER)
- Implementations of all DAuction types (TIP3, CurrencyCollections, EVER)
- Distributed programming, no mappings

Technical Requirements Analysis

The solution MUST implement the auctions for NEVER tokens automatically initiated upon price reveal as implemented by Defi#24	**Done**. (*Bank* contract starts auctions with startAuctions() method called from NotElector from Defi#24). Tests use a mock of NotElector.
The auctions MUST fulfill the requirements mentioned in the corresponding section	**Done**. Check corresponding sections.
The D'Auctions MUST be implemented fulfilling the requirements mentioned in the corresponding section	**Done**. Check corresponding sections.
The NEVER Tokens MUST be referred via the interface (or set of interfaces) that MUST be applicable, at least, both for NEVER native tokens (Currency Collections) and for TIP-3 style NEVER tokens	**Done**. Implementation provides 2 abstractions that are used by Bank & Auctions: ● **ITokenRoot** interface. TIP3 Root already support it, CC Root/Giver must follow its rules (at least for mint()) ● **AbstractBid** abstract contract. Any Bid implementation must extend this abstraction. Both abstractions are suited for both CC & TIP3.
The solution MUST be deployed to any test network (local usage is not enough)	**Done**. Deployed to dev.net (see README.MD)
The demonstration of the workable solution	**Ready**. Explanation video will be posted at

MUST be provided (both at AMA session as well as at the personal environment of each juror)	forum at submission time or shortly after. Please, contact @laugan for AMA & questions.
The manual for verifying the solution MUST be provided and MUST provide extremely clear and detailed instruction	**Done**. Refer to “Resources and Contacts” section.
In case the implementation of NEVER token is required for the demonstration its stub implementation MUST be provided	**Done.** All bid/auction logic done & submitted for **BOTH TIP3** and **CC**. TIP3 implementation of NEVER token is a part of submission. For CC-NEVER token its **Bid** implementation is also provided (*BidCC* contract) with all extra currency logic, security checks and so on. CC-NEVER-Root (source of fresh NEVERs) implementation can’t be done now as details of CC mint are yet unclear.
All the contestants MUST provide their contact information as a Telegram ID	**Done**. Refer to “Resources and Contacts” section.
All the participants are strongly encouraged to reuse the results of Defi#14, otherwise they MUST provide a strong explanation and description of their approaches	**Done**. Implementation follows original design.

Auction Requirements Analysis

Auction is performed on demand with a minimum lot size. It sells as many NEVERs as demanded by the winners but filtering participants by the required amount (one cannot participate if bids are valued smaller than predefined).	**Done**. Minimum bid is implemented and can be changed by Bank governance.
It is designed as Vickrey auction (sealed-bid, second price).	**Done**.
The bid sealing is performed by the commit-reveal scheme	**Done**.
It has the predefined zero position based on quoting result, so that the participant has to submit the bid higher (or same) than quoting price to get the bid accepted. The quoting price is taken from NEVER the Elector that was implemented during the first stage of the	**Done**. Minimum price goes from Elector as auction round start parameter. Reverse auction gets reversed minimum price.

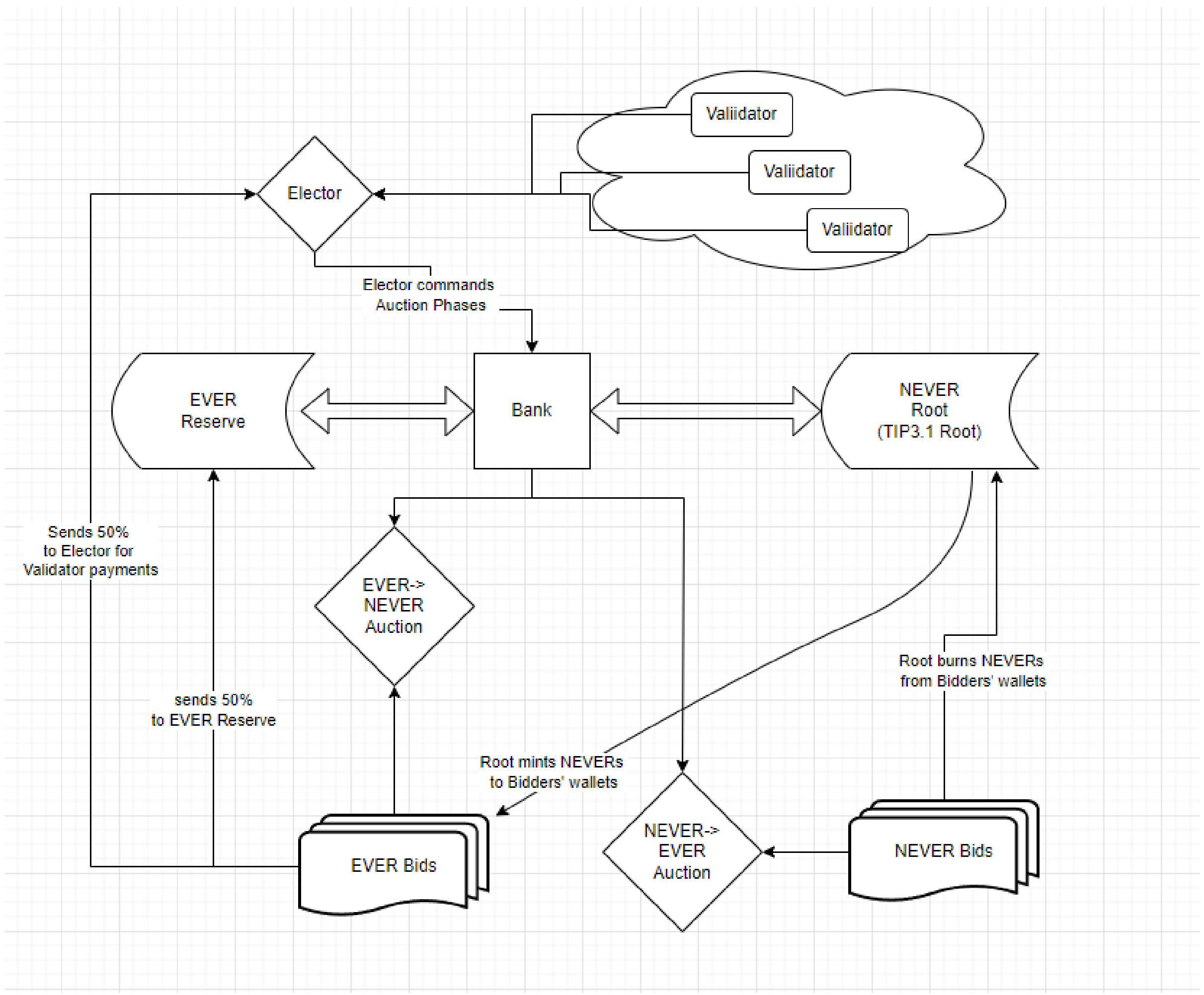
NEVER set of contests.	
The quoting price should not be disclosed before the auction starts (according to #2).	**Done**. Auctions are started with undisclosed price. Minimum (Quoting) Price is disclosed only on “reveal” phase.
The winner address, winning and paid (second) price are disclosed at the end of auction.	**Done**. See *AuctionSucceded()* event of **Auction** contract and *getWinner()* method.
If no one wins, the auction is considered as failed and the lot is not sold.	**Done**. If auction has failed, a lot is not sold and participants can’t trade in “trade” phase.
Auction is paid (so the bids must be locked and predefined fees are supposed to be taken for storage and winning), and the payment is to go to NEVER the validators, and as a payment for larger liquidity (because of minimum lot value).	**Done**. Funds are added to Bids on reveal, so it’s impossible to return funds until you lose. Payments are processed too both to Reserve Fund and to Validators (still, it’s up to Elector to distribute his ½ to Validators. As a validators list can be quite large, it’s a bad idea to send it around auctions).
9. After the auction ends every participant can buy the desired amount of NEVERs by the price determined at the last auction increased by some factor	**Done**. After the end of “reveal” phase, winner bid is traded and other participants can trade their “loser” bids (as a single amount or in parts). “Loser factor” that is added to price can be set in Bank settings prior to Auction.
10. Participants are encouraged to base their solutions on the existing implementations of Vickrey auctions (see DevEx SG contests #26 and #16). However, it’s up to participants to decide if such solutions are applicable or should be developed from scratch.	**Developed from scratch**. Explanation: NEVER Auctions have important differences from DevEx SG contests #26 and #16: ● Reversed auction (NEVER->EVER) ● Single pair of auctions ● Auction is based not on amount, but on price. All these differences are too much to use previous works.

D’Auction Requirements Analysis

Aggregator (representor). Account which owns the dAuction contract. It’s obligations include making a proper bid (which should be not less than a certain percentage of cumulative buying demand, algorithmical way to calculate the bid is preferrable), making a bid in a main auction at proper time.	**Offline+Online**. Here’s the reason: It seems that full online implementation of algorithmical bid calculation will easily disclose DAuction’s commit real bid price much earlier than it’s needed. We don’t see a way to implement it without ZKP, so currently we decided to stick to offline calculation.
Participant. Account which gives the contract rights to bid from her name acting together	**Done**.

with aggregator	
Intermediate media. Accumulates the bids in a proper way, allows the representor to make a price bid, sends the bid to the auction contract, pays all correct fees, collects the results and distributes the won lot between them all and returns or rebids the original bids if the contract loses the auction. Should be implemented in a decentralized way where the specific implementation of the items mentioned above is spreaded between Aggregator and mostly Participant.	**Done**. To make a distributable DAuction without extensive use of mappings, it was decided to make it behave more like Wrapper/StakedTon. When participant places bid to DAuction, he receives DAuction tokens as proof of having a bid. When bid is traded or unsuccessful, tokens are burned in exchange of real funds.
The aggregator and participants' buy price is finally adjusted by their amounts and roles (aggregator has some additional benefits as a reward for being a representor) All the D'Auction constants are subject to be determined at implementation phase, may vary from one contract to another and should establish adequate incentives for all players.	**Offline+Online**. Here's the reason: It seems that full online implementation of algorithmical bid calculation will easily disclose DAuction's commit real bid price much earlier than it's needed. We don't see a way to implement it without ZKP, so currently we decided to stick to offline calculation.

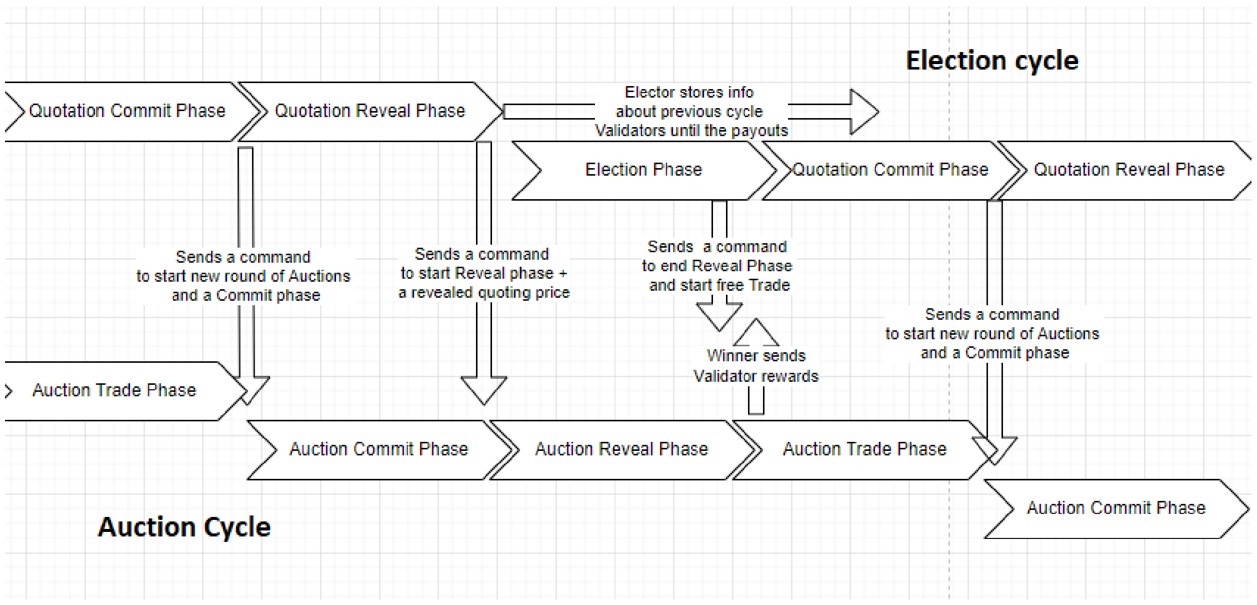
Auctions Architecture


Validator
Validator
Validator
Elector
Elector commands Auction Phases
EVER Reserve
Bank
NEVER Root (TIP3.1 Root)
Sends 50% to Elector for Validator payments
EVER-> NEVER Auction
Root burns NEVERs from Bidders' wallets
sends 50% to EVER Reserve
Root mints NEVERs to Bidders' wallets
NEVER-> EVER Auction
EVER Bids
NEVER Bids

Bid Flow consists of:

- After the start of **Auction** cycle (see Phases section below), **Bank** (AuctionManager.sol) spawns 2 **Auctions** (1 for EVER->NEVER auction, 1 for NEVER->EVER auction).
- Now, participants can commit their hashes to auctions. For each such commit, **Auction** spawns a **Bid** contract. It stores info about commited hash and some rules of its parent **Auction**.
- When **Elector** commands Reveal phase, bidders can send their actual bids to **Bid** contracts. **Bid** contract will check if reveal corresponds to commited hash (price, amount & salt are checked). Also, **Bids** are checked that real funds are locked on this step.
- Each reveal talks to **Auction** & tries to win the **Auction**.
- When Trade phase is called by **Elector**, **Auction** sends request to winner's **Bid** contract and forcefully exchange his **Bid**. All other participants can trade (fully or in parts), but with a *loserFactor* applied to exchange price.
- If a contract is a loser, but doesn't want to trade, it can apply for a refund of his locked funds.
- Exchange goes through NEVER Root (that can be both TIP3 Root and CC Giver) and EVER Reserve contracts.

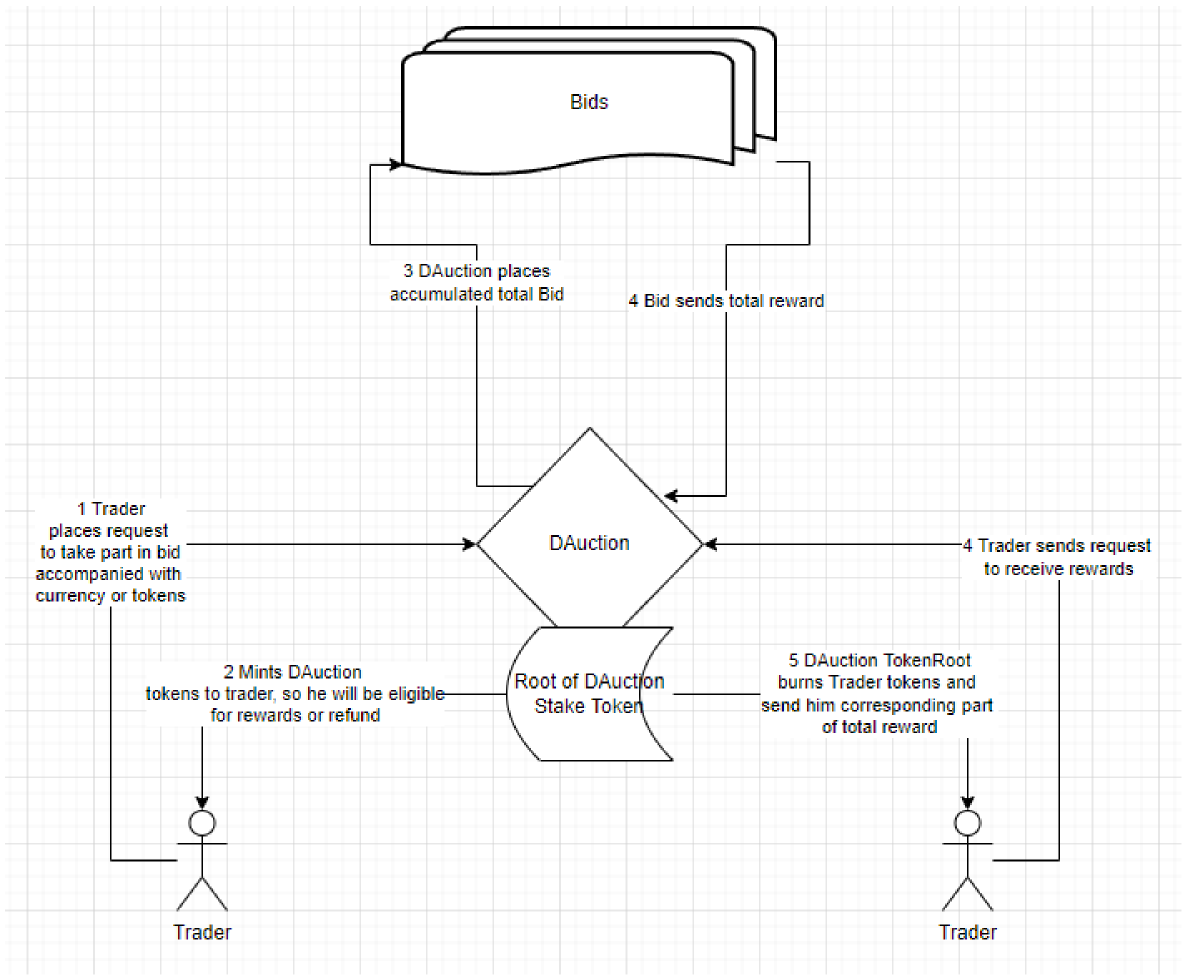
Phases Architecture


Election cycle
Quotation Commit Phase
Quotation Reveal Phase
Elector stores info about previous cycle Validators until the payouts
Election Phase
Quotation Commit Phase
Quotation Reveal Phase
Sends a command to start new round of Auctions and a Commit phase
Sends a command to start Reveal phase + a revealed quoting price
Sends a command to end Reveal Phase and start free Trade
Winner sends Validator rewards
Sends a command to start new round of Auctions and a Commit phase
Auction Trade Phase
Auction Commit Phase
Auction Reveal Phase
Auction Trade Phase
Auction Commit Phase
Auction Cycle

Cycle steps between Elector and Auctions

1. At the end of quotation commit phase Elector sends *startAuctions(uint64 dtStart_)* to Bank. This request destroys old auctions and starts a new pair of Auctions (EVER->NEVER and NEVER->EVER).
2. At the end of quotation reveal phase Elector sends *revealAuctions(uint128 minPrice_)* to Bank. This request reveals minimum price for bids. Now participants can reveal their bids.
3. Before the start of a new quotation commit phase Elector sends *tradeAuctions()* request to Bank. This request spends winner Bid and reward its owner with opposite currency. Part of EVER bids goes to Elector. This value should be distributed between validators, so elector should store its info about validators that have taken part in price discovery for extra cycle.
4. Cycle returns to step 1.

DAuctions Architecture


Bids
3 DAuction places accumulated total Bid
4 Bid sends total reward
1 Trader places request to take part in bid accompanied with currency or tokens
DAuction
4 Trader sends request to receive rewards
2 Mints DAuction tokens to trader, so he will be eligible for rewards or refund
Root of DAuction Stake Token
5 DAuction TokenRoot burns Trader tokens and send him corresponding part of total reward
Trader
Trader

Dauction Flow:

- When DAuction owner wants to take part in Auctions, it places one of DAuction contract implementations and set it up with collateral token root.
- As some trader wants to take part in DAuction, he sends a request placeTraderBid() to DAuction contract.
- In exchange for his bid, trader receives special DAuction tokens that are used as proof-of-posession.
- After Bid wins, trader is eligible to receive part of the rewards in exchange for burning his DAuction tokens.

Limitations

- Current implementation was made with a single Auction pair (EVER->NEVER & NEVER->EVER) in mind
- CurrencyCollection Root/Giver is not tested or implemented (no network to test)

Thanks for your time!