DoubleLiquid

Liquid Staking Protocol

Build as part of Sui Liquid Staking Hackathon

Secure SUI and earn rewards without your funds locked in.

iSUI or SUIDOUBLE_LIQUID_COIN, which automatically increases in value with staking rewards.

Take a look

- dApp: https://doubleliquid.pro/
- demo video: https://www.youtube.com/watch?v=a4WDbK9iCa0
- TDD simulations browser: https://doubleliquid.pro/simulations
- github: https://github.com/suidouble/doubleliquid-dapp
- follow DoubleLiquid on twitter: https://twitter.com/double2liquid

On the blockchain

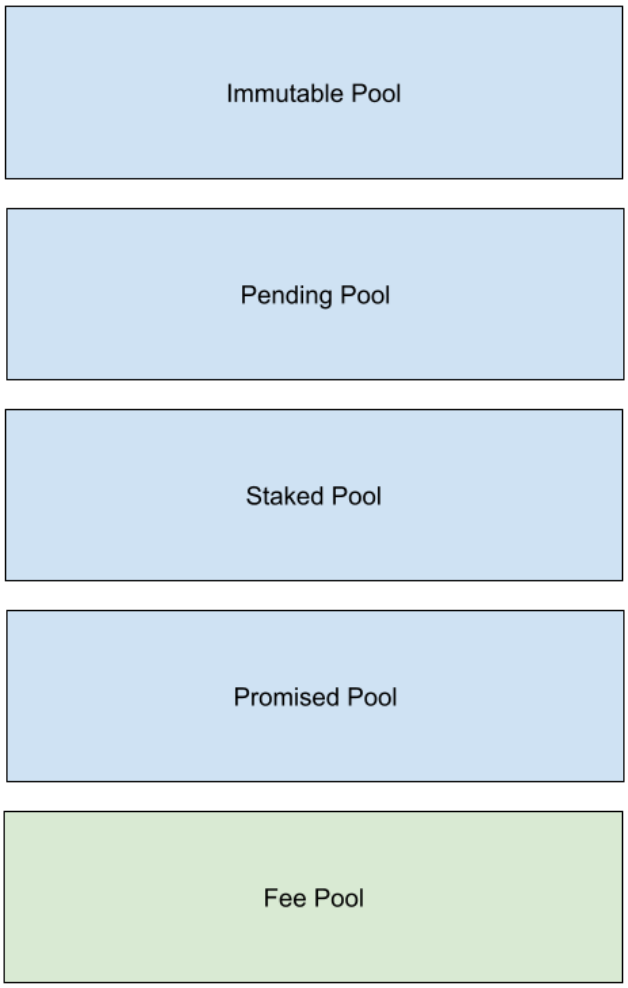
- protocol: 0x7372d1e5655441ccf754a637bf9f9e37a8ca113a2fa00f6e2b8f793abbfbdccd
- pool: 0x78d9273a9f774a2bd6c35cf79fbcf4029ee076cc249207610a3bcc0d6d0efc34

Goals

- decentralization
- non-custodial, no oracles, no off-chain data needed, no off-chain processes required
- no minimum stake, any amount is supported
- unstake promise ticket is NFT, can be traded and exchanged even before fulfilled
- algorithmically sorting/choosing validators, improving APY over time
- thousands of epochs in simulation have been run in TDD

Table of contents

LiquidPool structure



PendingPool

Stores deposited SUIs

StakedPool

Stores staked StakedSui

PromisedPool

Pool with information of user withdrawals, stores the SUI and Perfect Staked SUIs ready to take out in exchange of fulfilled LiquidStoreWithdrawPromise

ImmutablePool

Some amount of SUI and SUIDOUBLE_LIQUID_COIN stored as immutables for price stabilization. See edge-case.

Fee Pool

Amount of SUI and iSUI fees stored in LiquidPool available for admin to withdraw. See more info about fees.

Deposit to LiquidPool / Minting a LiquidPool tokens

Users can deposit SUI into the LiquidPool by adding SUI tokens into PendingPool (contract method: suidouble_liquid::deposit). This also mints an amount of SUIDOUBLE_LIQUID_COIN based on the calculated price and sends minted tokens to the user.

After deposit SUI is simply stored in PendingPool until the next once_per_epoch_if_needed function execution.

Goals here:

- minimize gas prices for the end user
- grouping different users SUIs together (so user can deposit any amount into LiquidPool)
- no restrictions on minting amount of SUIDOUBLE_LIQUID_COIN

Cons here:

- We may lost an epoch of StakedSui's rewards (note: simulation proofs we are fine)

Withdraw from LiquidPool / Burning a LiquidPool tokens

Basic SUI withdrawal is delayed for 2 epochs (settings as WithdrawPromiseCooldownEpochs) via LiquidStoreWithdrawPromise NFT.

Users can convert their SUIDOUBLE_LIQUID_COIN tokens back into SUI by calling the withdraw method (suidouble_liquid::withdraw). It calculates the current price, burns SUIDOUBLE_LIQUID_COIN tokens, increments the value of promised_amount(u64, amount to be fulfilled later in once_per_epoch_if_needed function), and issues an instance of LiquidStoreWithdrawPromise NFT to the user.

LiquidStoreWithdrawPromise has fields with the expected amount of SUI (sui_amount) and fulfilled_at_epoch with a value representing epoch when LiquidStoreWithdrawPromise can be converted back to SUI with the contract's fulfill method.

Fulfilling a LiquidStoreWithdrawPromise

When the system epoch reaches the value of LiquidStoreWithdrawPromise.fulfilled_at_epoch, users can convert LiquidStoreWithdrawPromise back to SUI. No price is calculated on the step, the contract issues the amount of SUI promised to the user when LiquidStoreWithdrawPromise is created.

SUI is taken from FulfilledPromisesPool (filled via once_per_epoch_if_needed function), so it's there and available any time (when system.epoch >= LiquidStoreWithdrawPromise.fulfilled_at_epoch).

LiquidStoreWithdrawPromise is burned.

Cons:

- we would still generate some rewards if a user would not want to burn fulfilled LiquidStoreWithdrawPromise. Covered!

Fast Withdrawal, with an extra fee

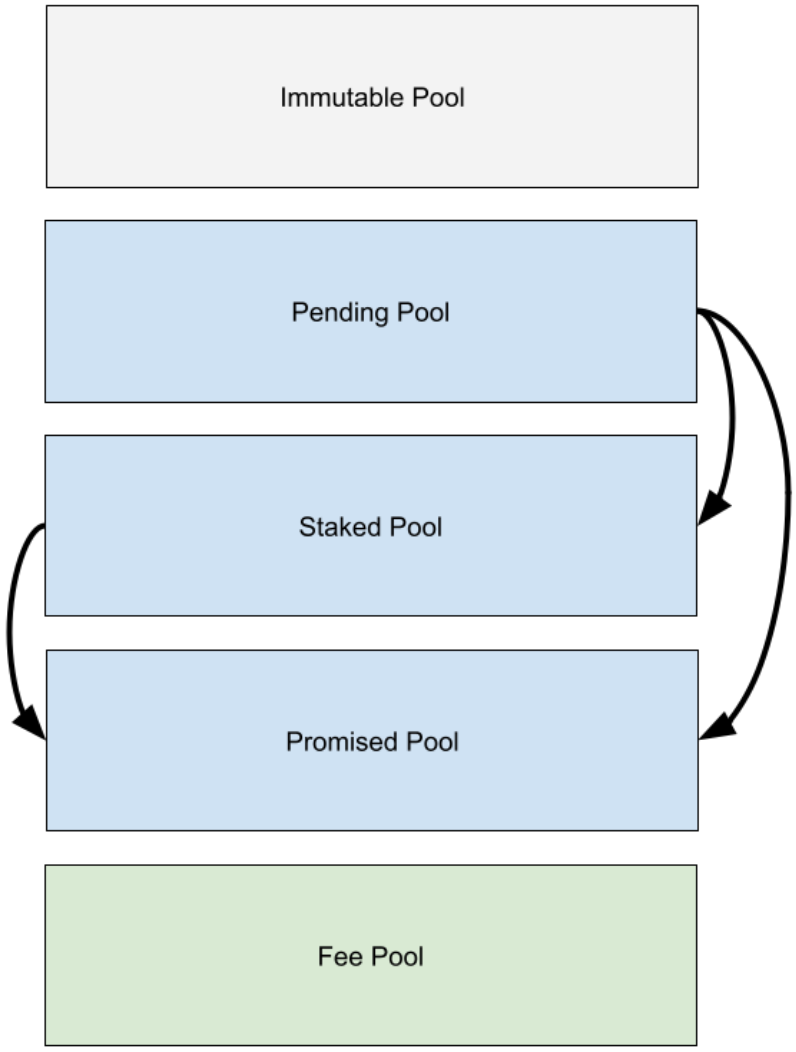
Also, an option to withdraw SUI in a fast way is implemented. It's subject to availability, as we have to have enough money in PendingPool or StakedPool in ready StakedSui. The function name is: **withdraw_fast**

We take **2%** out of exchanged SUI for admin to cover maintenance costs. Greatly depends on the user behavior, but we'll probably burn some of it to increase liquidity. More info about fees.

In the future, we expect some Dex to have iSUI-SUI as a trading pair and cover this functionality with a third-party smart contract.

once_per_epoch_if_needed

Is a contract's private function executing each time a system epoch updates and becomes > LiquidStore.liquid_store_epoch . It's executed in the background on each user's call (deposit, withdraw, fulfill), no maintainer or oracles needed.



The function updates internal blocks of the LiquidPool doing:

- Fulfilling required LiquidStoreWithdrawPromise objects for the current epoch (SUI is stored in PromisedPool and is ready to take out in exchange for LiquidStoreWithdrawPromise at any time).
- Staking PendingPool into StakedPool
- Emitting EpochEvent event for dApps to display historical data

Price calculation

Internal PendingPool price represents the current rate of exchange of SUIDOUBLE_LIQUID_COIN to SUI. Price is calculated for each swap transaction (deposit, withdraw), and emitted as events (PriceEvent) after successful swaps, so client-side Dapps can generate UI/statistics/graphs of current and historical data.

The price formula is:

get_current_price

amount of SUI you get from 1 SUIDOUBLE_LIQUID_COIN

let price = (balance(PendingPool) + balance(StakedPool) + expectedRewards(StakedPool) + balance(ImmutablePool) - promised_value(PromisedPool)) / supply(SUIDOUBLE_LIQUID_COIN)

get_current_price_reverse

amount of SUIDOUBLE_LIQUID_COIN you get from 1 SUI

let price_reverse = 1 / get_current_price()

Internally(and in events) price is represented as u64, it's multiplied by (1_000_000_000), so dApps have to do appropriate conversion on their side.

Algorithm to prioritize validators with higher APY

At the start of the Pool, while staking PendingPool each epoch, we appoint a validator on random, picking the random one available and active, so we are giving each Sui validator an equal chance to get presented in our pool.

However, we keep sorting our StakedPool StakedSui's by projected APY checked over the last 5 epochs, moving StakedSui with the lowest APY to the beginning of that StakedPool, keeping StakedSui with the highest APY at the end of it. In this way, when we do an unstake on behalf of the user, we automatically unstake StakedSui with the lowest APY available.

Also, we store a dictionary of relations (pool_id -> validator_address), so we can stake to the pool with the highest APY (from validators we've already deposited to).

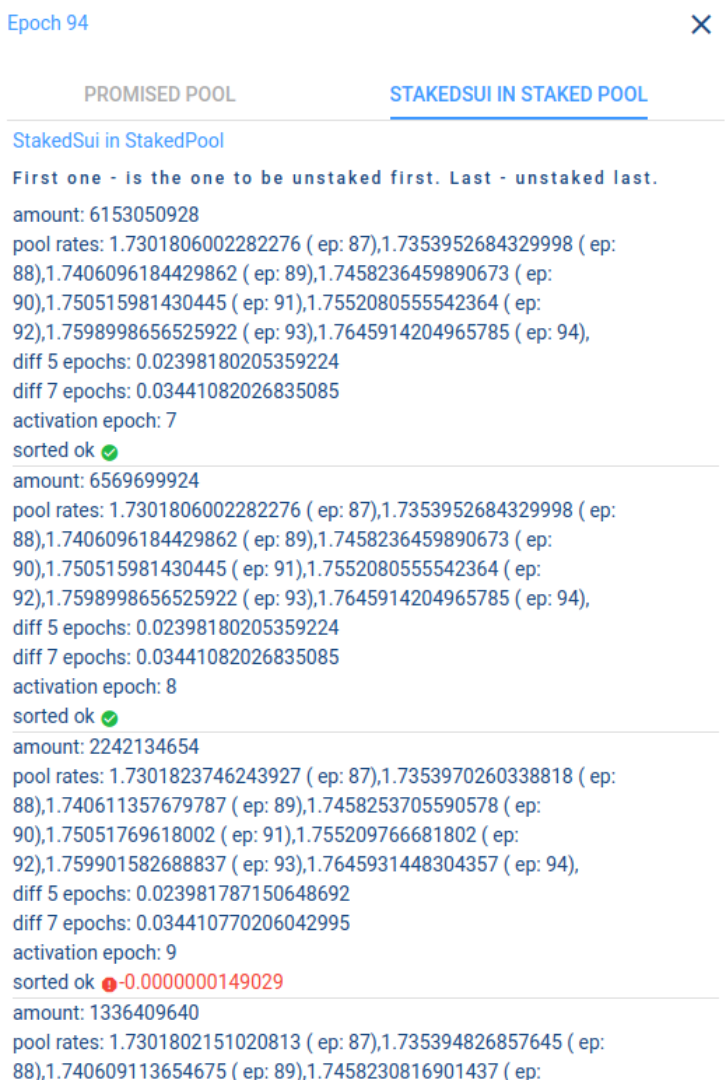
It's a simple object of type SuidoubleLiquidStats, using it is optional, The same methods are available with extra SuidoubleLiquidStats as a parameter:
deposit_v2
withdraw_v2
fulfill_v2
once_per_epoch_if_needed_v2

Doing the same as other methods, but with storing extra data in the SuidoubleLiquidStats object, and using it for better (higher expected APY) staking.

Logic is implemented with Option:: checking, so it works ok mixing normal and '_v' calls. However, I am currently forcing _v2 usage in the dApp

Current logic is staking 4/5 of the pending SUI to the pool with the highest active APY (from validators we've already deposited to) and 1/5 to the random active one. So we support decentralization. And, of course, if pending is < 5 SUI but > 1 SUI - everything goes to the random one.

Before sorting algorithm implemented:



Simulation: 1694023087971_4_users(Stake5PlusUnstake10_JustAddSome_JustAddSome_JustAddSome)_100epochs.json

Note StakedSui #3 has a lower growth rate than StakedSui #1 and StakedSui #2, but without algorithm, we'd unstake StakedSui#1 and #2 first, though they are better in APY. After implementing sorting algorithm:

Simulation: 1694026102958_4_users(Stake5PlusUnstake10_JustAddSome_JustAddSome_JustAddSome)_100epochs.json

We always have a StakedSui with lowest APY in the beginning of the array to be unstaked first.

Unstake deactivated StakedSui

As a side bonus effect of this process - we select StakedPools with no growth in the last 5 epochs(probably because they belong to deactivated Validators) and move them to be unstaked first.

Edge Case : Losing a epoch in stake_activation_epoch

By default architecture, **LiquidPool** stores all deposited SUI in the **PendingPool** till the next successful execution of the **once_per_epoch_if_needed** background function, after which this amount will be moved to StakedPool into StakedSui. This helps us accumulate SUI to increase the size of StakedSui, so there are more chances for it to fit into perfect StakedSui, and to have some SUI available for fast withdrawal. This is ok, but there's one great disadvantage of this - we are getting StakedSui with stake_activation_epoch greater by 1. And losing some rewards.

How we can get them?

This is why the helping **stake_pending_no_wait** function is introduced. It's very optional and what it does is simply run staking of PendingPool before the next epoch. So in perfect condition - our goal is to run it by cron or something just a few moments before the next Sui epoch arrives. Getting stake_action_epoch closer to us, and increasing our rewards.

In the future, when we have enough transaction density to be sure something is executed at least a few times per hour, we can move this logic into per-user transactions (check the current time and if it's close to the next epoch - run staking).

With this little change, these values in simulations:

Epoch	ImmutablePool	Token Supply	Calculated Price	PendingPool	StakedPool	StakedPool + Rewards
0	10000000	0.010000000	1	0.000000000	0.000000000	0.000000000
6	10000000	1.990163560	0.9999178157548053	1.980000000	0.000000000	0.000000000
7	10000000	3.970380684	0.9999041191940105	1.960000000	2.000000000	2.000000000
8	10000000	5.950652059	0.9998904230084329	1.940000000	4.000000000	4.000000000
9	10000000	7.923592654	1.0036195791025226	1.920000000	6.000000000	6.022272724
10	10000000	9.885660169	1.009195415221192	1.900000000	8.000000000	8.066562913
11	10000000	11.836158836	1.0151955031419773	1.880000000	10.000000000	10.126015224
12	10000000	13.773996335	1.0218429976655383	1.860000000	12.000000000	12.204861698
13	10000000	15.698482103	1.0289471489168405	1.840000000	14.000000000	14.302908395
14	10000000	17.609175752	1.036389519136644	1.820000000	16.000000000	16.419965176
15	10000000	19.505802178	1.0440916718084179	1.800000000	18.000000000	18.555845591
16	10000000	21.388200326	1.0519990672765869	1.780000000	20.000000000	20.710366783
17	10000000	23.256290437	1.060072304563692	1.760000000	22.000000000	22.883349381
18	10000000	25.110052161	1.0682820264592583	1.740000000	24.000000000	25.074617392
19	10000000	26.949509389	1.0766058006967165	1.720000000	26.000000000	27.283998122
20	10000000	28.774719427	1.0850261163941672	1.700000000	28.000000000	29.511322071
21	10000000	30.587176539	1.0926772775704803	1.680000000	30.000000000	31.731912775
22	10000000	32.386952792	1.1003929306184903	1.660000000	32.000000000	33.968373885
23	10000000	34.174134185	1.1081648737938674	1.640000000	34.000000000	36.220575084
24	10000000	35.948817642	1.1159863059686286	1.620000000	36.000000000	38.488388170

Simulation: 1694555340679_2_users(AddALittleEachEpoch_AddALittleEachEpoch)_20epochs.json

Becomes these values:

Epoch	ImmutablePool	Token Supply	Calculated Price	PendingPool	StakedPool
0	10000000	0.010000000	1	0.000000000	0.000000000
6	10000000	1.990163560	0.9999178157548053	0.000000000	1.980000000
7	10000000	3.970380684	0.9999041191940105	0.000000000	3.960000000
8	10000000	5.950652059	0.9998904230084329	0.000000000	5.940000000
9	10000000	7.916408569	1.0072875755230606	0.000000000	7.920000000
10	10000000	9.866108887	1.0155971747796864	0.000000000	9.900000000
11	10000000	11.800670237	1.0235593379544097	0.000000000	11.880000000
12	10000000	13.719733090	1.0318405128777302	0.000000000	13.860000000
13	10000000	15.623133201	1.040346436695063	0.000000000	15.840000000
14	10000000	17.510825687	1.0490185877754057	0.000000000	17.820000000
15	10000000	19.382842767	1.05781820189134	0.000000000	19.800000000
16	10000000	21.239268395	1.0667183220036078	0.000000000	21.780000000
17	10000000	23.080222150	1.0756994659671715	0.000000000	23.760000000
18	10000000	24.905848537	1.084747117064604	0.000000000	25.740000000
19	10000000	26.717504160	1.0931288058147046	0.000000000	27.720000000
20	10000000	28.514159719	1.1022721706127554	0.000000000	29.700000000
21	10000000	30.297473828	1.110535585915138	0.000000000	31.680000000
22	10000000	32.067594567	1.1188300927178638	0.000000000	33.660000000
23	10000000	33.824674168	1.1271516297494995	0.000000000	35.640000000
24	10000000	35.568867622	1.135496826473727	0.000000000	37.620000000

Simulation: 1694559844663_2_users(AddALittleEachEpoch_AddALittleEachEpoch)_20epochs.json

Visibly increasing the price and Staked Sui amount.

Edge Case : burning too much coins

If somebody burns almost all SUIDOUBLE_LIQUID_COIN, leaving only a few mSUIDOUBLE_LIQUID_COIN in circulation, the price instantly goes to the moon(as we gather extra SUI in the delayed unstaking). This is fine from a market point of view, but makes the pool almost unusable in this case - you'll have to spend thousands of SUI to get the price back to the Ok level.

See the simulation of the smart contract version with this bug:

token_total_supply	pending_amount	staked_amount	staked_with_rewards_balance	all_time_promised_amount	promised_amount	promised_fulfilled	price_calculated	epoch
99.99	1	100	100	1.01	1.01	0	1	5
99.9801	0	102	102	2.0199	2.0199	0	1	6
99.959142568	0	103	104.139529584	3.041096294	3.041096294	0	1.011397563972	7
99.927273897	0.023015851	102.99	105.268310091	4.07369859	3.06369859	1.01	1.0230202762999	8
99.884855369	0.013115851	102.99	106.429474833	5.11758651	3.09768651	2.0199	1.0346403745815	9
0.000998849	0.059490453	101.99	106.53381898	109.633360504	106.59226421	3.041096294	1.0464274379811	10
0.000998849	0.574014957	106	106.020919144	109.633360504	105.559661914	4.07369859	1036.4651583973	11
0.000998849	0.570655098	105	105	109.633360504	104.515773994	5.11758651	1056.0966712686	12
0.000998849	0.054092503	2	2.009514562	109.633360504	0	0	2065.9850137508	14

1693604998244_strategy_1.csv

Solution: We can just keep some tokens ourselves and promise users we would not burn them, so there is always some liquidity. **But**, one of our goals is to implement a **non-custodial** smart contract, so we are going to add an ImmutablePool - a small portion of SUI + SUIDOUBLE_LIQUID_COIN added to the smart contract on its creation, with no ability to withdraw.

Adding a very small immutable pool with an amount of 10 SUI works just fine, see simulation:

token_total_supply	pending_amount	staked_amount	staked_with_rewards_balance	all_time_promised_amount	promised_amount	promised_fulfilled	price_calculated	epoch
109.99	1	100	100	1.01	1.01	0	1	5
109.9801	0	102	102	2.0199	2.0199	0	1	6
109.960146569	0	103	104.139529584	3.040060171	3.040060171	0	1.0103612343158	7
109.930252191	0.023015851	102.99	105.268310091	4.070580142	3.060580142	1.01	1.0209268473705	8
109.890726686	0.013115851	102.99	106.429474835	5.111350288	3.091450288	2.0199	1.0314895880331	9
10.000998908	0.060526578	101.99	106.533818982	109.211825155	106.171764984	3.040060171	1.0421539560076	10
10.000998908	0.062295077	100.99	106.535757476	109.211825155	105.141245013	4.070580142	1.1455663224636	11
10.000998908	0.131044766	99.949229854	106.459946515	109.211825155	104.100474867	5.111350288	1.2489268850943	12
10.000998908	0.513362603	3	3	109.211825155	0	0	1.3512012877224	14

1693608783969_strategy_1.csv

ImmutablePool is implemented as commit [150476b60eb25fee398c05cbf95925e7b6e71057]

Fields in LiquidStore: immutable_pool_sui, immutable_pool_tokens

Thoughts: There's a temptation to make immutablePool virtual. But this would probably not be fair for the end users. Better to keep it low enough to allow the market to do its job with price, still covering edge cases.

Todo: we can stake that immutable_pool_sui and gather some profits

EdgeCase : Token burned, but user doesn't hurry to exchange the Promise

And has a right to. She/he can keep the LiquidStoreWithdrawPromise as NFT in their wallet as long as they want to. They may never exchange it for SUI. And we should store the SUI available for them to withdraw.

Would it be cool to keep getting rewards for that SUI?
This would be great.

Solution: find_the_perfect_staked_sui function. When it's time to fill LiquidStoreWithdrawPromise, we try to find and split out the perfect StakedSui for this exchange:

- amount >= MIN_STAKING_THRESHOLD
- if unstaked at the current rate, makes the amount of SUI needed to fulfill LiquidStoreWithdrawPromise

If we find such StakedSui - we add it to PromisedPool with relation to LiquidStoreWithdrawPromise, so when user is ready / wants to exchange it to SUI - we have a ready StakedSui object that can be fulfilled to the expected amount of SUI, + getting some profits on top of it.

We re-stake that extra profits on the next once_per_epoch_if_needed execution, increasing our pool liquidity.

Note: Getting the matched StakedSui is not always possible (see the function source code for math). So the find_the_perfect_staked_sui function returns Option<StakedSui>, so when we can't find/split out good StakedSui, we go with a common simple unstaking flow.

Implemented as commits (cbde85ac542502a5ea8b5f84e8f3e94d3da39ee2) and (c864f202c910140023ff1b150774f06c69a1f5a7 , improved and refactored version)

Simulation:

Running a simulation of the user withdrawing 2% of her pool, delaying exchanging promise by 5 epochs (available to exchange after 2 epochs). Difference with switched off/on find_the_perfect_staked_sui increases pool price and gives user > 1 SUI (as increased pool liquidity, not directly).

You can check out simulation csv files in github repository:
https://github.com/suidouble/suidouble-liquid/tree/main/simulations

Take a look at results in screenshots below:

_rewards_balance	all_time_promised_amount	promised_amount	promised_fulfilled	promised_amount_in_staked	extra_staked_in_promised	all_time_extra_amount	still_waiting_for_sui_amount	price_calculated	epoch	user_balance_sui
0	0	0	0	0	0	0	0	1	3	3989.91250760
100	0	0	0	0	0	0	0	1	4	3889.88197393
101	10.1	10.1	0	0	0	0	0	1	5	3888.87297129
103.152935752	19.393867048	19.393867048	0	0	0	0	0	1.0114265189436	6	3887.86395649
105.31699873	27.962196357	27.962196357	0	0	0	0	0	1.0239838553932	7	3886.85492953
97.077379925	35.87742451	25.77742451	10.1	0	0	0	0	1.0377580518868	8	3885.8489272
89.95285575	43.195491957	23.801624909	19.393867048	0	0	0	0	1.0515044143972	9	3884.84476714
83.178392135	49.967165266	22.004968909	17.862196357	0	0	0	0	1.0651406394336	10	3893.9386649
77.059981567	56.23105887	20.35363436	16.483557462	0	0	0	0	1.0772675581734	11	3902.2274079
71.470568638	62.031338276	18.835846319	15.2332956	0	0	0	0	1.0892632300552	12	3909.79061324
66.374845337	67.408402702	17.441237436	14.089740756	0	0	0	0	1.1011178821178	13	3916.70071734
61.738827561	72.399207416	16.168148546	13.035566913	0	0	0	0	1.1128237564168	14	3923.01366074
57.521809771	77.037557062	15.006218786	12.06417301	0	0	0	0	1.1243751910121	15	3928.78021000
53.687338799	81.354366792	13.94596409	11.177343832	0	0	0	0	1.1357670549619	16	3934.03897956
50.20199285	85.377903001	12.978695585	10.36786914	0	0	0	0	1.1469949758577	17	3938.8341349
47.035202763	89.13400449	12.096447428	9.62915436	0	0	0	0	1.1580553961155	18	3943.20607529
44.159008602	92.646285623	11.291918831	8.955159376	0	0	0	0	1.1689456301079	19	3947.19175596
41.547843395	95.936322909	10.558419908	8.340345939	0	0	0	0	1.1796639040945	20	3950.82498156
39.143737485	99.021179315	9.887174825	7.779637698	0	0	0	0	1.1891548314083	21	3954.13666724
36.961448367	101.919346018	9.273060395	7.268382622	0	0	0	0	1.1984903081449	22	3957.15507940
33.792721886	128.202637637	32.266314728	6.802318419	0	0	0	0	1.2076703618	23	3960.93687445
31.019744667	128.202637637	29.181458322	6.374893692	0	0	0	0	1.2341168172	24	3964.45214726
28.367764218	128.202637637	26.283291619	5.983023109	0	0	0	0	1.2582007916	25	3967.74517622
2	128.202637637	0	29.181458322	0	0	0	0	1.2800526282	26	3970.83302429
2	128.202637637	0	26.283291619	0	0	0	0	1.2800526282	27	3973.75904394
2.015167675	128.202637637	0	0	0	0	0	0	1.2815693957	28	4000.04317734
2.030333021	**128.202637637**	0	0	0	0	0	0	1.2830859303	29	4000.04308851

find_the_perfect_staked_sui switched **off**: 1693697710080_strategy_ExtraStaked.csv

> Everything goes through normal un-staking flow, we fulfill LiquidStoreWithdrawPromise with SUI as soon as it's LiquidStoreWithdrawPromise.fulfilled_at_epoch

And if we switch perfect staked sui logic, on the same data we have:

_with_rewards_balance	all_time_promised_amount	promised_amount	promised_fulfilled	promised_amount_in_staked	extra_staked_in_promised	all_time_extra_amount	still_waiting_for_sui_amount	price_calculated	epoch	user_balance_sui
0	0	0	0	0	0	0	0	1	3	3989.910904008
100	0	0	0	0	0	0	0	1	4	3889.880370332
101	10.1	10.1	0	0	0	0	0	1	5	3888.871367692
103.152935752	19.393867048	19.393867048	0	0	0	0	0	1.0114265189436	6	3887.862352892
105.31699873	27.962196357	27.962196357	0	0	0	0	0	1.0239838553932	7	3886.853325932
97.392058474	35.87742451	25.77742451	0	10.1	0	0	0	1.0377580519122	8	3885.84309954
90.171643714	43.194825763	23.800958715	0	19.393867048	0	0	0	1.0514073658796	9	3884.832860988
83.5852002	49.965157693	22.002961336	0	17.862196357	0.224928246	0.224928246	0	1.0649251438626	10	3893.92573852
77.479303978	56.246541408	20.369116898	0	16.483557462	0.194418929	0.419347175	0	1.0802661829533	11	3902.21344298
71.903976663	62.078456915	18.883631152	0	15.232629406	0.167921988	0.587269163	0	1.0952453085737	12	3909.775591461
66.814877051	67.497480718	17.532323025	0	14.087733183	0.153586867	0.74085603	0	1.1098608909919	13	3916.684620546
62.159691583	72.538171375	16.291629967	0	13.051715645	0.140594815	0.881450845	0	1.1242701721534	14	3922.995804491
57.904087333	77.232293588	15.153836673	0	12.113299222	0.128820774	1.010271619	0	1.138460153927	15	3928.759900873
55.016167359	81.609037146	14.111556428	0	11.25093931	0.118368376	1.128639995	0	1.1524190480702	16	3934.0350308
51.465625408	85.695235867	13.157064492	0	10.45971446	0.108851931	1.237491926	0	1.1661412379945	17	3938.860674279
48.23371113	89.51622477	12.283931182	0	9.73481287	0.100191276	1.337683202	0	1.1798371700436	18	3943.273407814
45.285569318	93.094540009	11.485502863	0	9.070865771	0.092325458	1.43000866	0	1.193291246879	19	3947.307789963
42.597534091	96.450866765	10.755630898	0	8.462942279	0.085181702	1.515190362	0	1.2064971023307	20	3950.995585428
40.110081541	99.601537631	10.085312861	0	7.907187624	0.074759503	1.589949865	0	1.2183921231604	21	3954.365983998
37.842812941	102.564180652	9.469640643	0	7.399304142	0.065522391	1.655472256	0	1.2299334945598	22	3957.445819491
34.777142224	129.426873664	32.976006899	0	6.934641995	0.060772559	1.716244815	0	1.2456607581	23	3961.292246978
31.898640196	129.426873664	29.825336033	0	6.506997622	0.056454742	1.772699557	0	1.2789548978	24	3964.873408349
29.183660843	129.426873664	26.862693012	0	6.113313887	0.052529624	1.825229181	0	1.3093667388	25	3968.232581237
3.094776206	129.426873664	0	26.862693012	2.962643021	0.048920288	1.874149469	0	1.3371000359	26	3971.399436235
3.118429522	129.426873664	0	26.862693012	0	0.045639256	1.919788725	0	1.3443573963	27	3974.36620994
3.142079181	129.426873664	0	0	0	0	1.919788725	0	1.3512862878	28	4001.256743736
3.165725207	**129.426873664**	0	0	0	0	1.919788725	0	1.3536508904	29	4001.25664274

find_the_perfect_staked_sui switched **on**: 1693696870602_strategy_ExtraStaked.csv

> Some extra SUI generated from StakedSui in PromisedPool, still available any time for pay out as SUI

Fees

DoubleLiquid has an ongoing management fee of (~0.5% p.a.) to support further product development. It automatically takes issuing token supply of (total_token_supply / 365 / 1000 * 5) each epoch (generated tokens are stored in FeePoolToken and available for withdrawal by admin.

While this amount is relatively small, and covered by staking SUIs in validators, we can earn more. Both for us and for our users.

Our ambitious goal is to overperform the average StakedSui in APY for end users, still getting some more fees for us.

How is that possible?

There are 3 places where we get extra SUI from rewards.

The first two are an extra amount we get directly from StakedSui rewards. We do fulfill promises later than the SUI amount is calculated, so we usually receive some extra rewards:

- for 2 epochs, if we go with normal flow, and StakedSui is unstaked at the time Promise becomes fulfilled.
- for 2 + N epochs, if we go with perfect StakedSui and user can exchange her/his Promise on the epoch (epoch + 2) or later.

And the third one:

- when the user uses fast withdrawal and we pay out SUI with a fast withdrawal discount.

What we a going to do is to take the third one (from fast withdrawal) fee for us (FeePool). Send the first one to the PendingPool again fully (increasing out pool liquidity). And split the second one by half, keeping half as our fees and adding the other half back to liquidity. So the general idea is that these extra rewards will cover our management fee of (0.5% p.a.) and possibly even more.

While all this is a subject for user behavior we can't much predict for now, but it works okay in simulation. And we'll be able to adjust values later for better APY (or just add our fees as liquidity directly to the pool).

Simulations

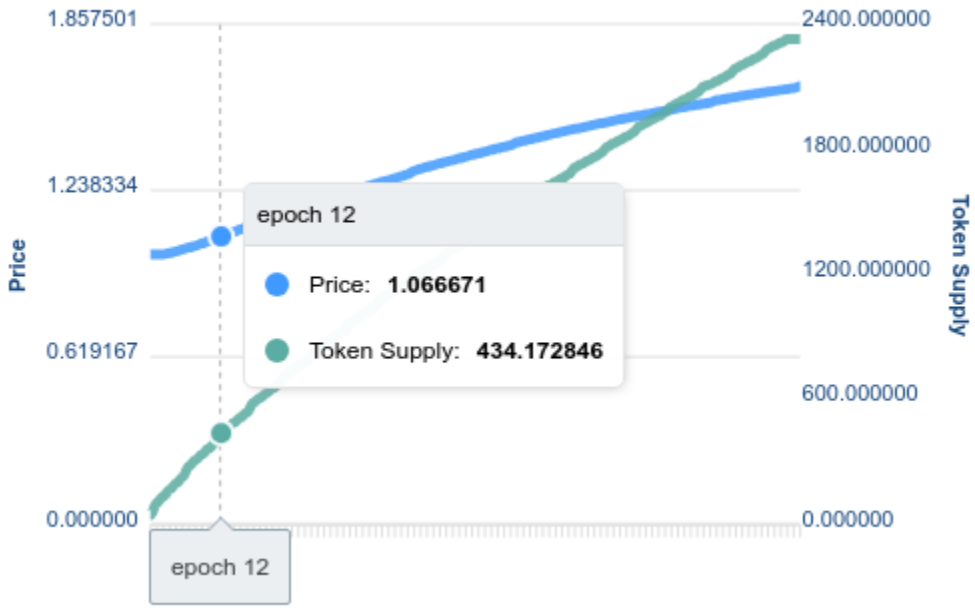
Some (many) of the simulations are available to take a look online:
https://doubleliquid.pro/simulations

Built a quite complex system to simulate and test liquid pool smart contract on the local node. As a side of it - a javascript library for interaction with it from the browser or node js side is available too.

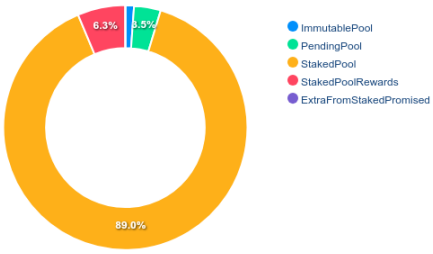
Hundreds of tests have been run, with different settings and different sets of rules through the process of Move code optimization. Some simulation results are cached and available to take a look at the simulation browser on our dApp. Check it out.

Notes to notice there:
For each simulation:

2 charts are displayed.
Price + Token distribution (move a mouse over it to take a closer look at values).



And **Pool distribution** (amount of Sui stored in each part of the Pool each epoch. Also, there's a little player where you can navigate different epochs.



There 2 tabs below. **Epochs** and **Transactions**.

Epochs

Epochs contains gathered information about the state of the Pool after each epoch. You can also click on the row to get some more detailed insights.

		STORE						TRANSACTIONS			
Epoch	Token Supply	Calculated Price	PendingPool	StakedPool	StakedPool + Rewards	Promised (next epochs)	Promised (ready, StakedSui)	Promised (ready, SUI)	PromisedPool (all time)	extra_staked_in_promised	extra_staked_in_promised (all time)
5	39.970000000	1	29.970000000	0.000000000	0.000000000	0.000000000	0.000000000	0.000000000	0.000000000	0.000000000	0.000000000
6	97.840000000	1	30.940000000	60.000000000	60.000000000	3.100000000	0.000000000	0.000000000	3.100000000	0.000000000	0.000000000
7	129.520000000	1	30.910000000	95.000000000	95.000000000	6.390000000	0.000000000	0.000000000	6.390000000	0.000000000	0.000000000
8	160.850062645	1.0052181919372483	30.880000000	126.934530482	127.575860207	6.766451062	3.100000000	0.000000000	9.866451062	0.000000000	0.000000000
9	191.811302012	1.0116276019796815	30.850000000	158.717008686	160.316814203	7.125206716	6.390000000	0.000000000	13.515206716	0.000000000	0.000000000
10	222.398371974	1.0188215897978308	30.820000000	190.354172145	193.220251304	7.455988401	9.866451062	0.000000000	17.322439463	0.000000000	0.000000000
11	275.450023609	1.0257935552733344	30.790000000	247.858915500	252.122049412	10.357190394	10.415206716	0.000000000	23.872397110	0.100093832	0.100093832
12	303.077402946	1.0328830853838868	30.860093832	279.246895892	285.169263326	12.985834093	10.932439463	0.000000000	30.308273556	0.101414131	0.201507963
13	330.578728691	1.0411430572918476	30.931507963	308.091918473	316.022884755	12.774644453	14.005946048	0.000000000	36.647041563	0.102187072	0.303695035
14	357.943608245	1.0495360228387252	30.003695035	338.101142258	348.260809219	12.589793256	16.793066840	0.000000000	42.898066812	0.106190133	0.409885168
15	385.163219175	1.0580492425701775	30.079885168	367.255839089	379.864487707	12.422720561	19.324602100	0.000000000	49.069762124	0.109716240	0.519601408
16	434.172846254	1.0666708807742105	30.159601408	422.544751330	437.832183660	14.872252746	19.025669702	0.000000000	57.770319558	0.186923464	0.706524872

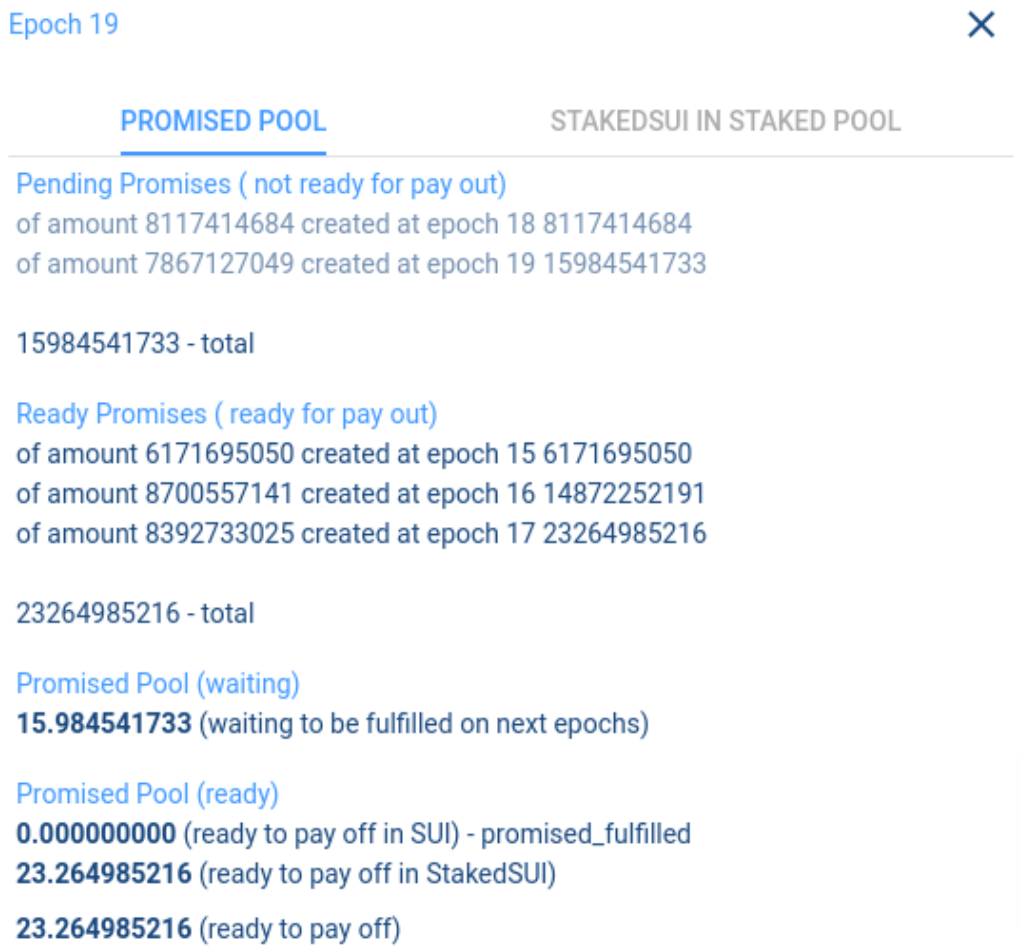
More info about columns there:
epoch - epoch number (note that we run simulations on the local node, so epochs start from 0
Token Supply - the total amount of SUIDOUBLE_LIQUID_COIN currently in circulation
Calculated Price - the price of Sui/SUIDOUBLE_LIQUID_COIN to be used for the next **deposit** or **withdraw** transaction. These are the values calculated on the client side using information about the pool state. As part of debugging/optimization, I checked that the next transaction matches this value.
PendingPool - amount of SUI in the PendingPool. This is the balance users added via the **deposit** method, which was not yet passed to the StakedPool (it will be on the next once_per_epoch run)
StakedPool - amount of SUI in StakedPool, as balance directly moved into.
StakedPoolWithRewards - StakedPool + expected amount of SUI we'd get if we'd unstake all StakedSui from the StakedPool just now.
Promised(next epochs) - the amount of Sui we promised to users when they burn their SUIDOUBLE_LIQUID_COIN via the **withdraw** method. This is the amount not yet fulfilled. We are going to fulfill it as soon as the user's Promises are ready (in 2 epochs by current settings)
Promised(ready, StakedSui) - an amount that is ready for payout via the **fulfill** method stored as perfect StakedSui.
Promised(ready, Sui) - the amount in PromisedPool that is ready for payout via the **fulfill** method stored directly in the Sui balance.

PromisedPool(all time) - just helpful info showing how much Sui we promised all the time till the current epoch
extra_staked_in_promised - the amount of extra Sui we got this epoch because we stored promised payout in perfect StakedSui and got extra when unstaking it (if user calls **fulfill** method later than she/he can).
extra_staked_in_promised(all time) - just helpful info showing how much extra Sui we got for our pool as we implemented the perfect StakedSui algorithm.

When you click on the table row - you see the popup dialog with 2 tabs: (**PromisedPool** and **StakedSui in StakedPool**)

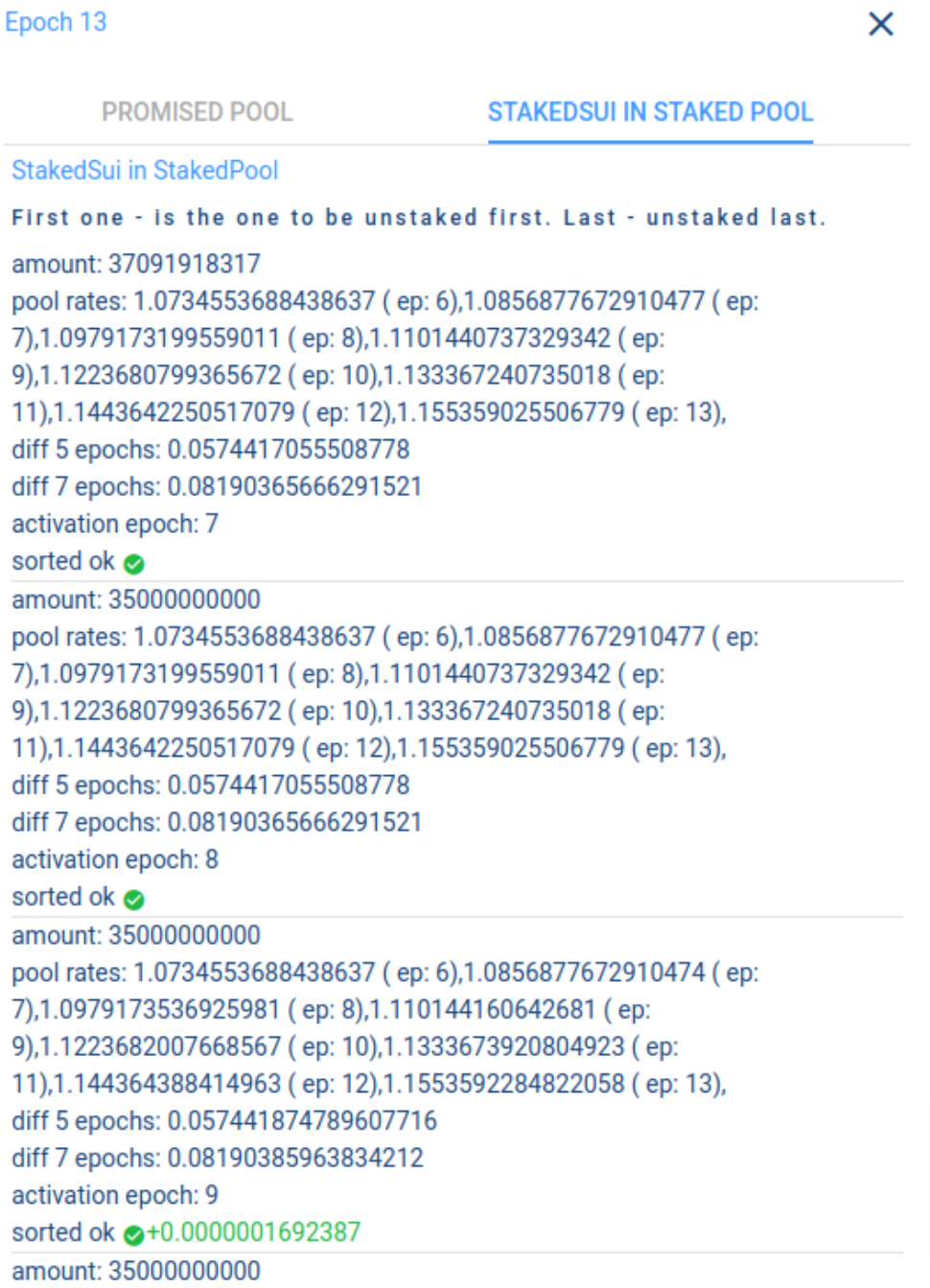
PromisedPool

Information we gather on both the client and smart contract side for debugging the PromisedPool and the user's promises. You can take a closer look at the Promises we promised to users.



Promised Pool (waiting) should match Pending Promises (not ready for payout). And Promised Pool (ready) should match Ready Promises (ready for payout).

StakedSui in StakedPool



Shows a list of StakedSui our StakedPool currently operates. Both amount, activation epoch, and StakedPool's validator's pool prices (PoolTokenExchangeRate.sui_amount / PoolTokenExchangeRate.pool_token_amount) with a goal to keep StakedSui with the lowest pool price growth rate closer to the start of the list, so it gonna be unstaked first, while keeping the StakedSui with highest pool price growth closer to the end of the list, optimizing our rewards rate.

Transactions

24	deposit	9.99 SUI	8.797360183 TOKEN	Alice -199.80 SUI, 189.21 TOKEN (~ 214.86 SUI), +0.00 SUI in 0 Promises, total: ~ 15.06 SUI
24	deposit	9.99 SUI	8.797360183 TOKEN	Eve -199.80 SUI, 189.21 TOKEN (~ 214.86 SUI), +0.00 SUI in 0 Promises, total: ~ 15.06 SUI
24	deposit	9.99 SUI	8.797360183 TOKEN	Kate -199.80 SUI, 189.21 TOKEN (~ 214.86 SUI), +0.00 SUI in 0 Promises, total: ~ 15.06 SUI
25	deposit	5.0 SUI	4.37178984 TOKEN	Sapyada -121.85 SUI, 74.43 TOKEN (~ 85.14 SUI), +37.65 SUI in 4 Promises, total: ~ 0.95 SUI
25	withdraw	7.442763111 TOKEN	promise of 8.514402986 SUI	Sapyada -121.85 SUI, 66.98 TOKEN (~ 76.63 SUI), +46.17 SUI in 5 Promises, total: ~ 0.95 SUI
25	fulfill		7.639585845 SUI	Sapyada -114.21 SUI, 66.98 TOKEN (~ 76.63 SUI), +46.17 SUI in 5 Promises, total: ~ 8.59 SUI

Shows a list of transactions we run on behalf of different users while running the simulation. Along with the balance of the user after that transaction. Note that the price of SUI/SUIDOUBLE_LIQUID_COIN for guessing displaying the total is taken from the 'epochs' data (where we have only the value on the last check of the epoch).

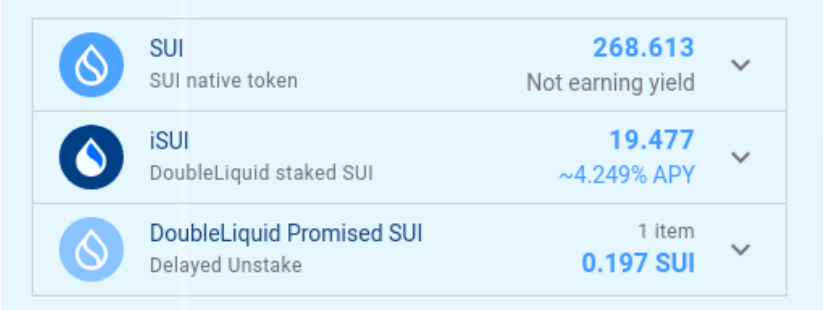
dApp

One page, yet powerful UI has been built for dApp to interact with DoubleLiquid LiquidPool.

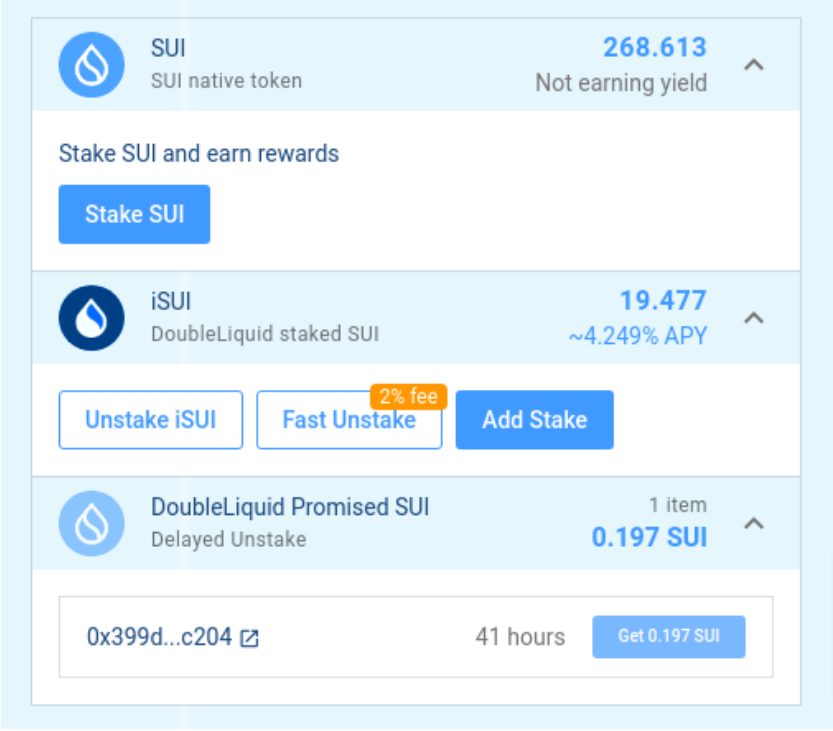
Available online at https://doubleliquid.pro/

Two main parts are:

Assets Block

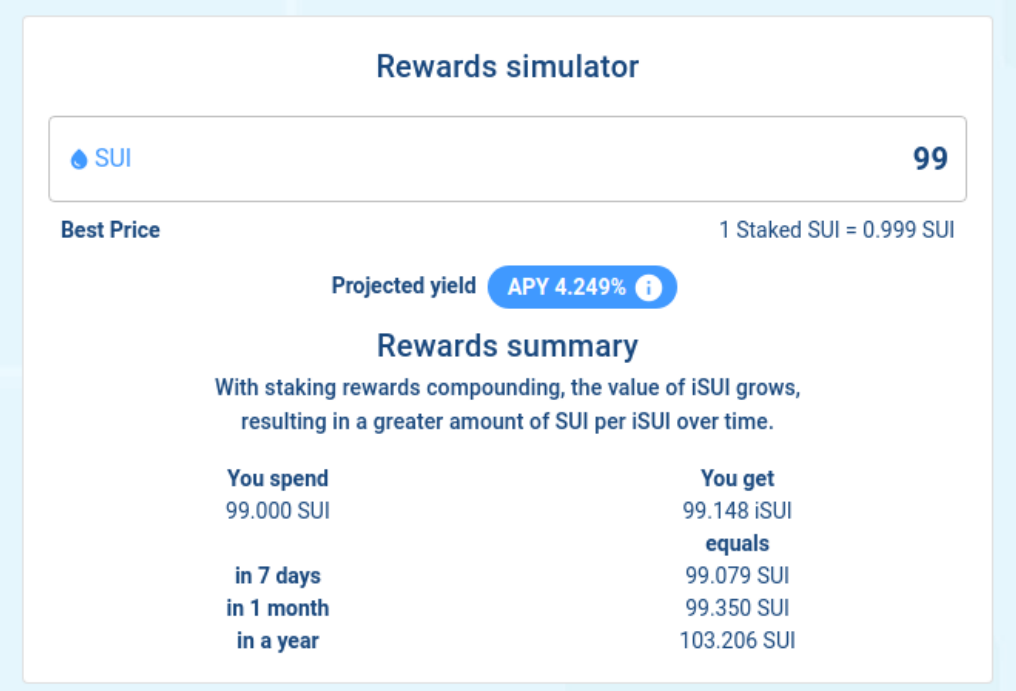


And with sections expanded:



The place where the user can Stake SUI, Unstake iSUI, or exchange DoubleLiquid Promises

Rewards calculator



The place where the user can estimate how much SUI she/he is going to make with DoubleLiquid liquid staking protocol.