



Ocean Market Balancer Simulations For Token Engineering Academy

May 21, 2021

Trent McConaghy

@trentmc0 @oceanprotocol

[\[video\]](#)

Outline

- Summary: Research Q's
- Intro
 - What's Ocean?
 - EE Simulation & Verification
 - TE Simulation & Verification
- Ocean System TE
 - Base
 - SW Verification w TokenSPICE
- Ocean Market TE
 - V3 base
 - V4.1 base
 - V4.1 SW Verification w TokenSPICE & EVM
- Roadmap - this research & beyond
- Conclusion



Summary: Research Q's

Summary: Research Q's

Basis:

- Ocean Market uses Balancer AMMs
- Doing TE to model, verify and optimize Ocean Market is highly useful on its own
- And it's well-defined subset of broader Balancer ecosystem; we can extend scope once we've got a handle on Ocean Market dynamics

Research Q's

- Can we capture the dynamics of Ocean Market / data ecosystem for Ocean V3? (system identification problem). Includes capturing observed issues .
- Ocean has new mechanisms, aiming to address the observed issues. How well do those mechanisms work?
- Tool: use TokenSPICE with EVM-in-the-loop
<https://github.com/oceanprotocol/tokenspice>

What's Ocean?



What's Ocean?

● Ocean V3 is now live →

Tools for the Web3 Data Economy

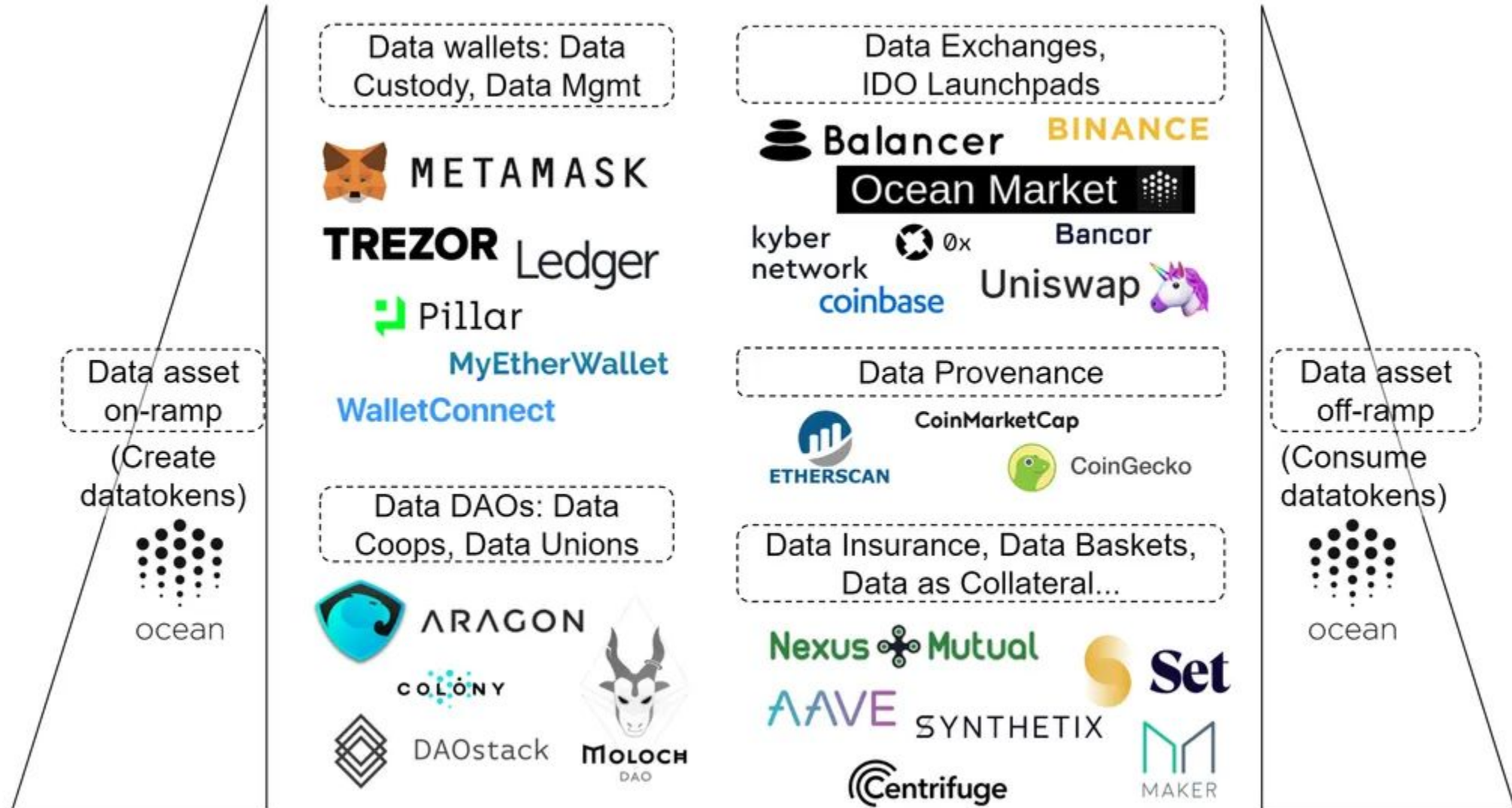
Use **Ocean Market app** to earn by selling data and curating / staking on data. Use Ocean Protocol libraries to **build your own app** for secure, privacy-preserving data exchange.

In Ocean Protocol, each data service gets its own **datatoken**. This enables data wallets, data exchanges, and data co-ops by directly leveraging crypto wallets, exchanges, and more.

OCEAN MARKET APP



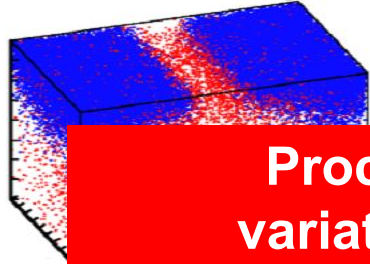
Ocean Datatokens: On-ramp data services into data assets, and off



Electrical Engineering (EE) Simulation & Verification

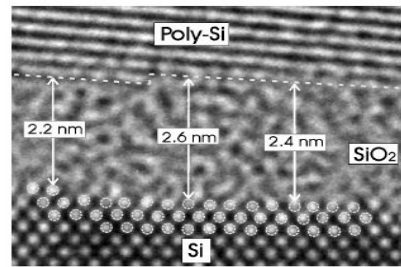
Variation = atoms out of place

...Propagating from devices to performance & yield



Process
variation ↑

Random dopant effects

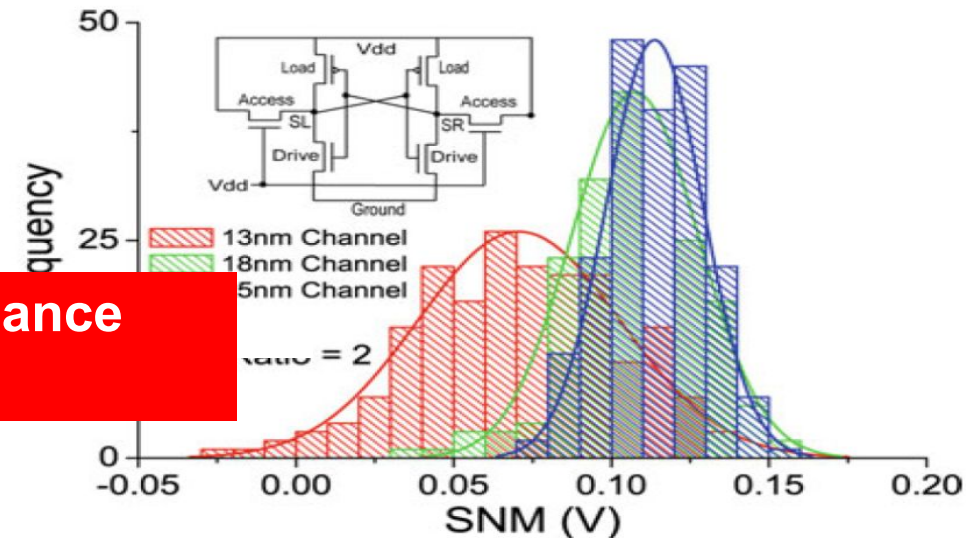


Oxide thickness

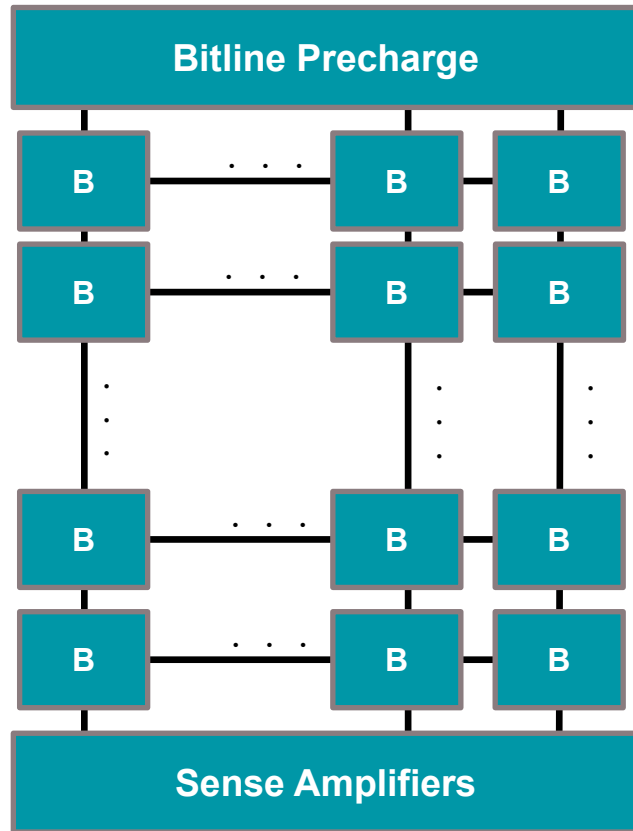
⋮

Device performance
variation ↑

Circuit performance
variation ↑



Rare Event Verification for Memory: Problem



Consider a 256Mb SRAM:

- 256M bitcells
- 64k sense amps
- 4k bitcells / sense amp

Want overall yield to be 90-99%.

For the SRAM to yield, need:

Bitcell sigma $\approx 6\sigma$

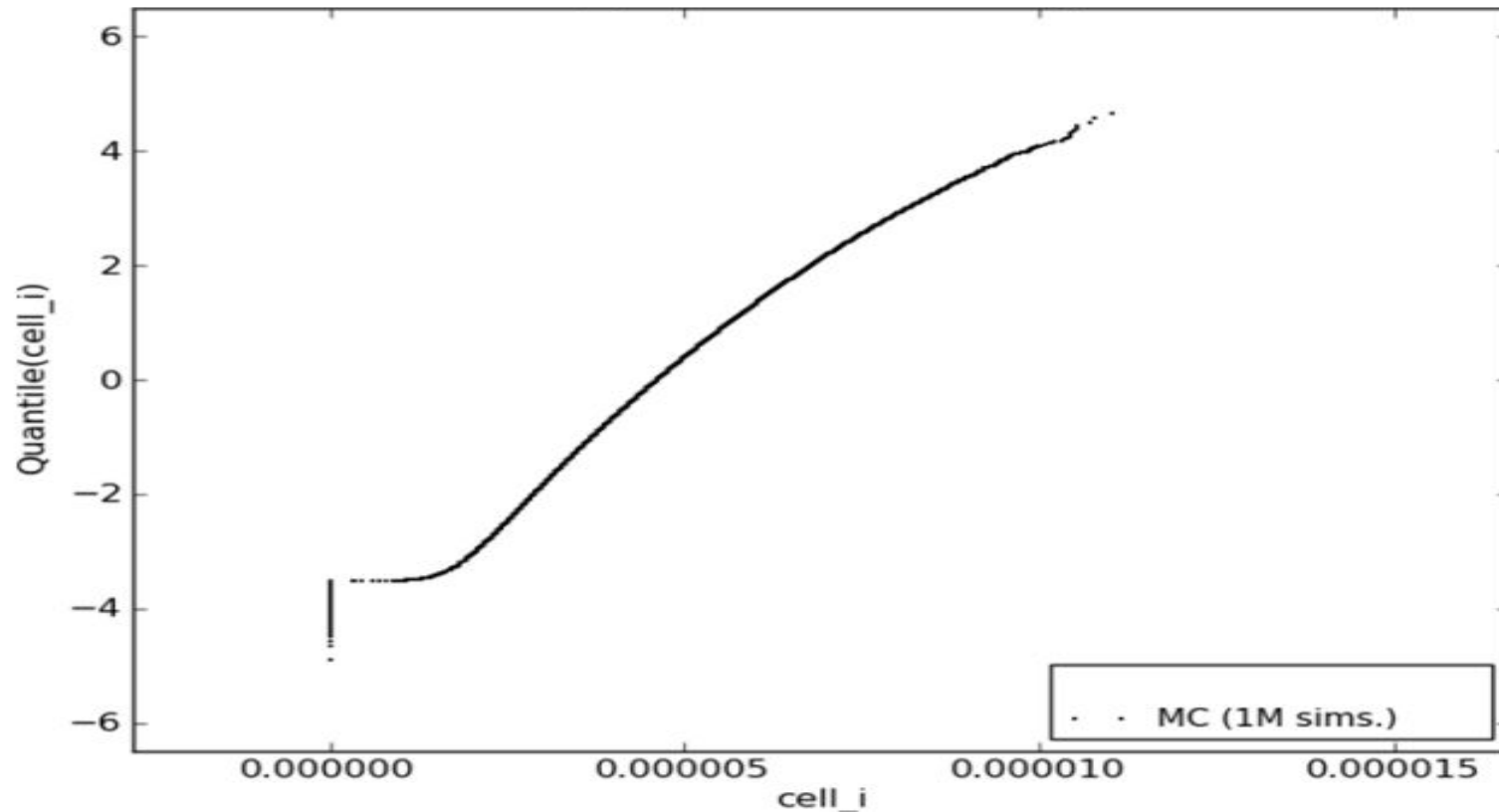
Sense Amp sigma $\approx 4.5\sigma$

1% improvement in overall yield makes a huge difference.

Memory is the leading edge of billion dollar fabs...

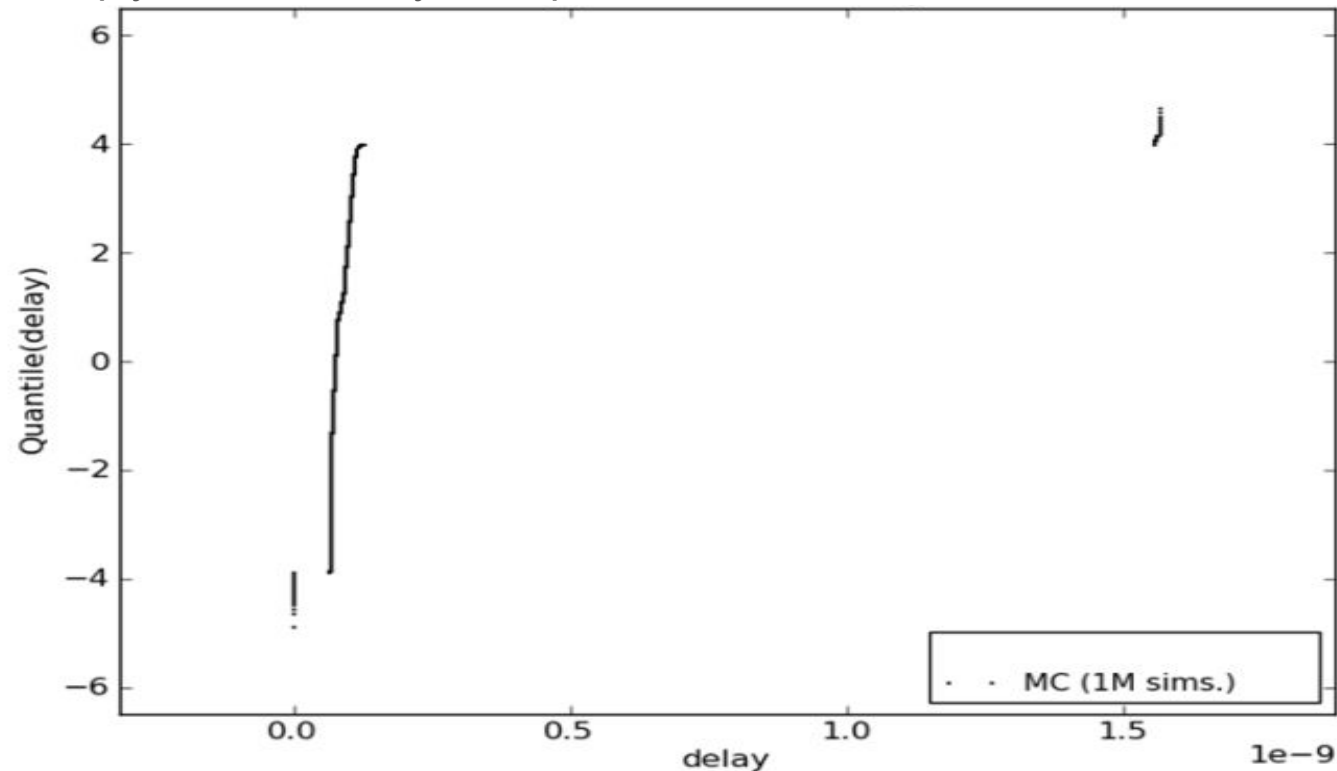
Rare Event Verification of a Memory Bitcell via SPICE-in-the-loop & AI tricks to reduce # sims

- 6 devices x 10 local process variables / device = 60 variables
- Simulated 1M MC samples. Each dot in curve is a sample.
 - The bend means quadratic response in that region
 - The dropoff / vertical means a flat response in that region (in this case, transistors turning off)



Rare Event Verification of a Memory Sense Amp via SPICE-in-the-loop & AI tricks to reduce # sims

- 15 devices x 10 local process variables / device = 150 variables
- The three vertical “stripes” mean
 - three modes
 - tight distributions in each mode, almost flat response
 - left mode is “off”, right mode is “extreme”
 - gaps between stripes imply a discontinuity in response

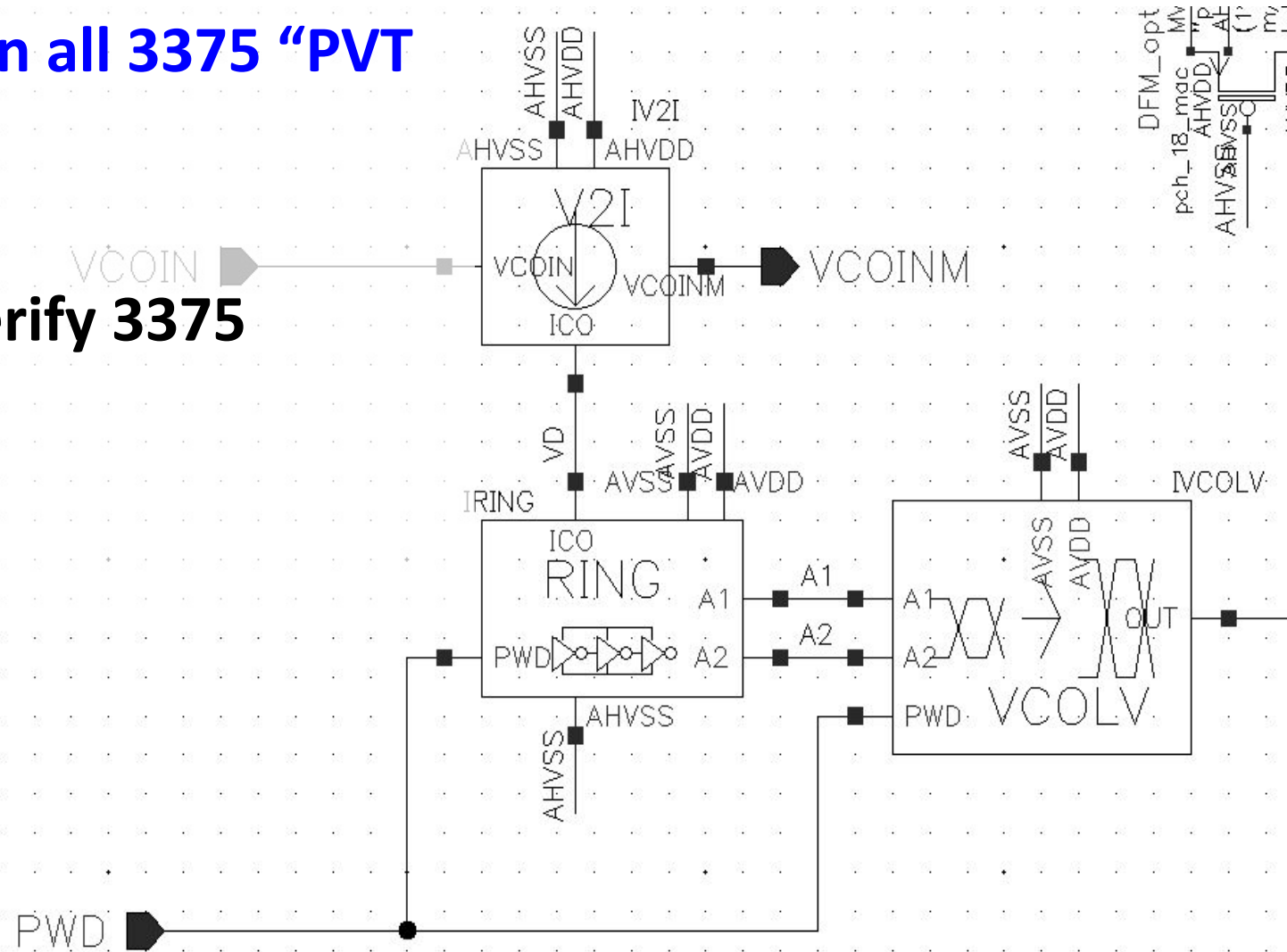


Worst-Case Verification for VCO of a PLL: Problem

Q: Does circuit meet constraints on all 3375 “PVT corner” combos?

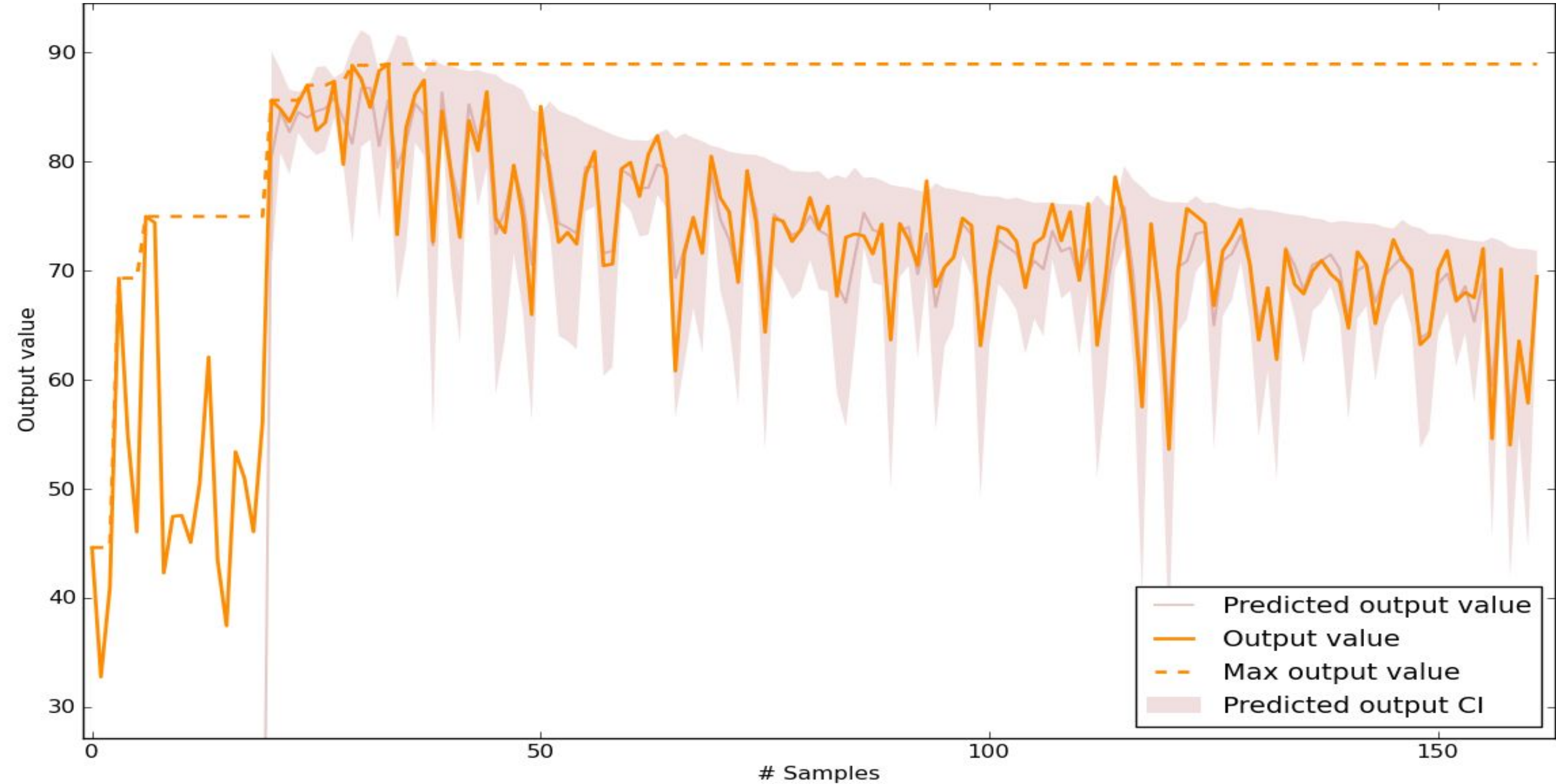
Result: used 171 evaluations to verify 3375 corners

- $3375/171 = 19.7x$ speedup
- 65.6 h \rightarrow 3.3 h (1 core) or 20 min (10 cores)



Worst-Case Verification for VCO of a PLL: Solution

SPICE-in-the-loop + AI/optimization to reduce # sims



TE Simulation & Verification

On Verifying Token-Based Systems

A Token Engineering perspective



Trent McConaghy

Jul 14 · 11 min read



| *“Trust, but verify” — Russian proverb*

Defining the Problem

TE Verification is about evaluating the token-based system to find out whether it meets the specified requirements. The system could be a simple tool or a full tokenized ecosystem, instantiated as one or more smart contracts or even L1 blockchain networks.

<https://blog.oceanprotocol.com/on-verifying-token-based-systems-c33eca757ecf>

As a starting point, let's recognize that we are talking about *dynamical systems*. A *failure* is when the dynamical system gets stuck in a region of state space that does not reflect design intent: it's a *dynamical system fault*. It's like getting caught in the wrong loop.

There could be failures on the *logic* side; this is a *digital* problem. There could be failures on the *incentives* side; this is an *analog* problem. There could be failures on the combination; this is a *mixed-signal* problem.

Type of TE Verification	Type of TE Design	Verify What?	Where?	Parallel in Circuit Examples
Digital Verification	Digital: Discrete time (clocked), discrete-valued signals (typically binary).	Digital behavior, instantiated in smart contract logic .	Single smart contract, or a set of smart contracts.	Multiplexers, floating-point units, ARM cores
Analog Verification	Analog: Continuous time, or continuous valued-signals	Analog behavior, instantiated in smart contracts incentive design / economic model and other analog signals.	Single smart contract, set of smart contracts, or system level (may be across >1 chains).	Amplifiers, filters, memory bitcells.
Mixed-Signal Verification	Mixed-signal: Some digital and some analog blocks. Overall system is analog.	Mixture of analog and digital behavior.	Set of smart contracts, or system level.	Analog-to-digital converters, RF transceivers, memory columns

On TE Verification

It's pragmatic to do verification in **phases** of increasing fidelity:

1. **Humans.** Subjective discussions, with increasing # people. 1 → 2 → key stakeholders
2. **Software modeling,** with increasing fidelity. Spreadsheet → agent-based sim → high-fid sim
3. **Economic (live).** Can ratchet value-at-risk over time. People can choose risk/reward tradeoff.
Phased approach.

SW modeling with increasing fidelity

1. **Spreadsheet-based agent-based system.** Each row is a different time step. Some columns are state variables; some are input variables; some are output variables. The next row's state variable values are a function of that row's input variables and the previous row's state variables. Output variables are a function of the current row's state variables and input variables.
2. **Custom software for agent-based modeling, with rough-grained models.** The rough grained models may be subroutines, differential equations or other “behavioral models”. This approach takes more up-front effort than (1), but offers more flexibility. Once that up-front effort is invested, it's also easier to maintain and test, towards building more complex models.
3. **Custom software for agent-based modeling, with fine-grained “smart contracts in the loop”.** This is even higher resolution than (2). Simulation time is longer but it starts to go with shades of gray into real-world behavior. It's akin to hardware-in-the-loop simulation.

Ocean System TE

Ocean System TE: Process

<I did the following process over 8 weeks with a collaborator - Julien Thevenard @ Fabric.vc>

1. Goals:
 - a. Write out goals (first-cut)

2. Design:
 - a. Explore various designs to achieve the goals. Loop back to update goals.

3. Verification:
 - a. (Better formalize what “verification” can mean, and how to do it - see blog post on TE Verification)
 - b. Approach: Manual analysis & conversation. Loop back to update goals or design.
 - c. Approach: SW to answer Q’s. Loop back to update goals or design.
 - i. spreadsheet-based
 - ii. agent-based sim - TokenSPICE



Ocean System TE: Goals

Find a design to enable...

- Ecosystem *sustainable and growing*, towards *ubiquity*
- Funding goes to teams improving L1-L3 etc, over the long term (10+ years)
- \$OCEAN grows as usage of Ocean network grows

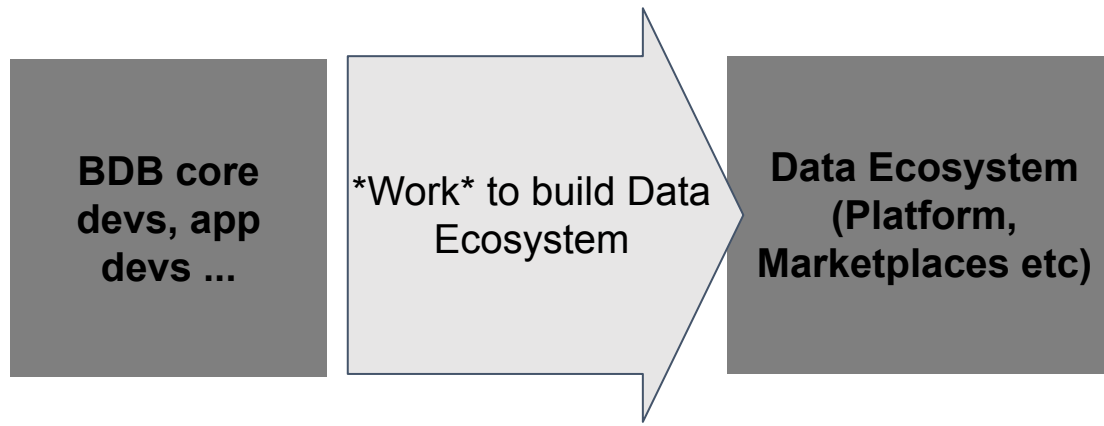
Including:

- Basic design is simple to understand and communicate
- Can be implemented in a pragmatic fashion, over time
- Get people to do “work”,
- Encourage skin-in-the-game by users

A choice of system-level design will lead to goals of sub-blocks in the system.



Early 2020: BigchainDB is building Ocean data ecosystem (on behalf of OPF).

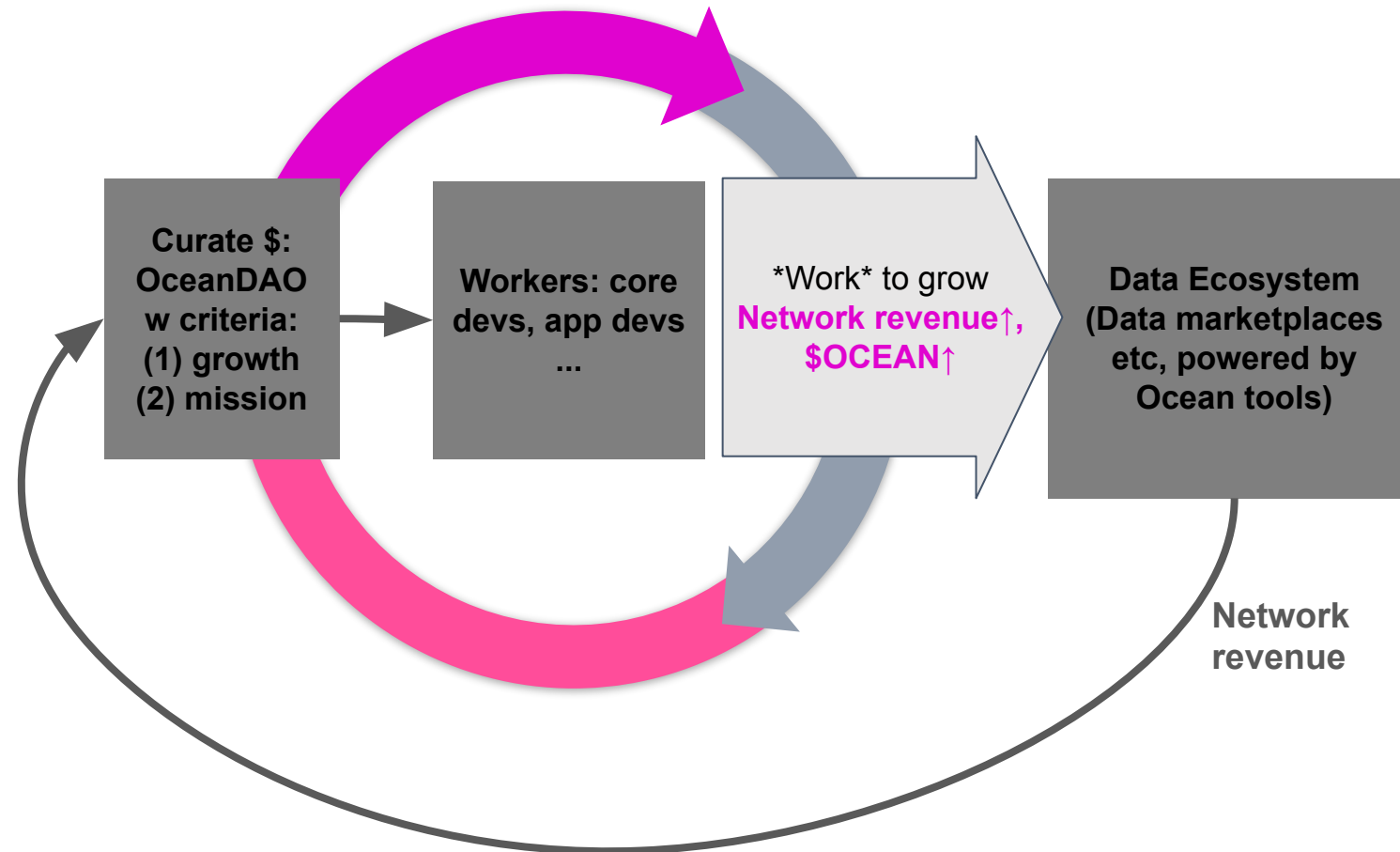


How do we make the data ecosystem sustainable and growing? Including having funds to keep improving platform etc.

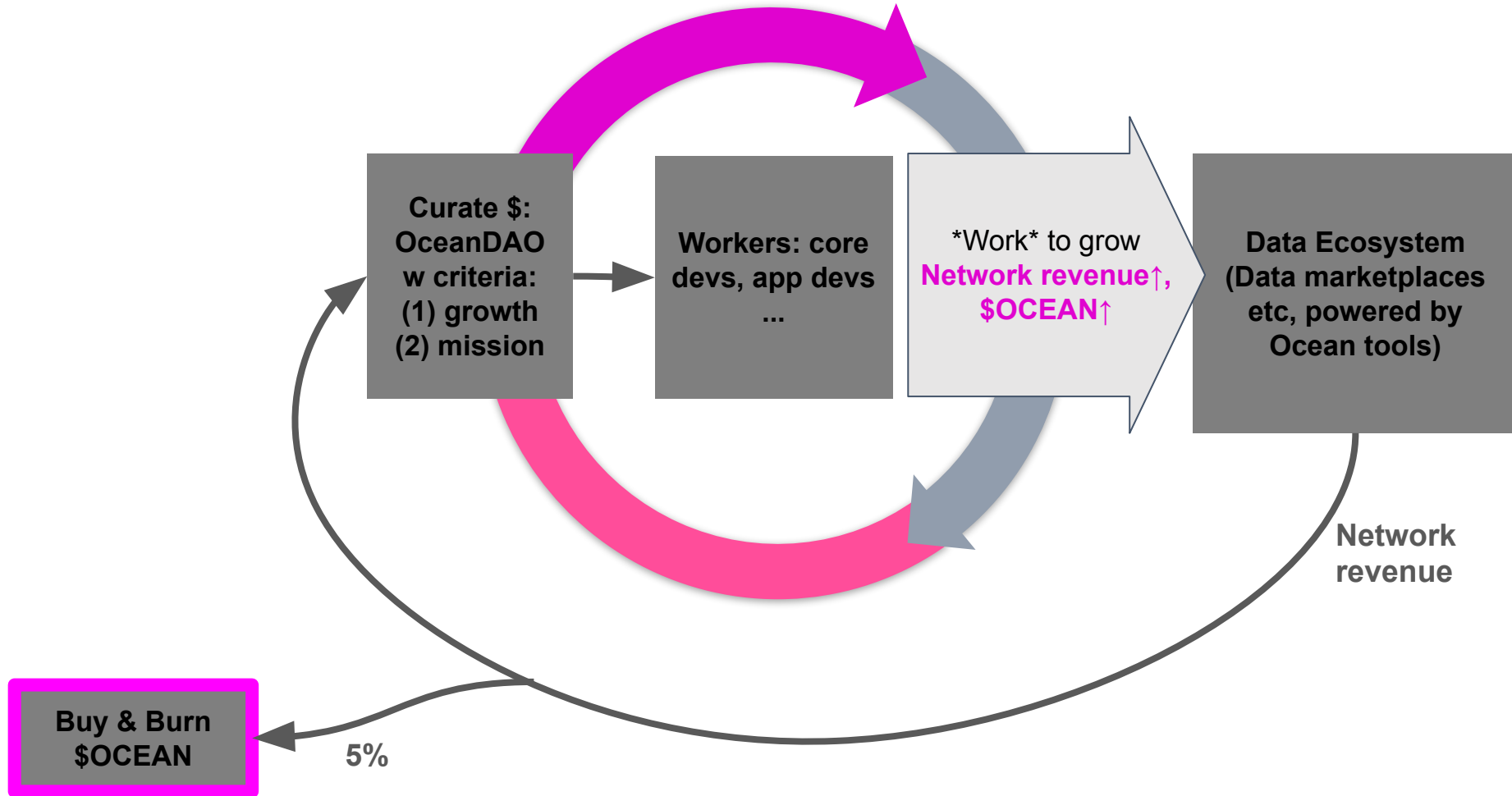
- Get network revenue as % of marketplace revenue
- To pay for work
- That grows marketplaces revenue
- That grows network revenue

A loop! It's \cong the loop of any sustainable business.

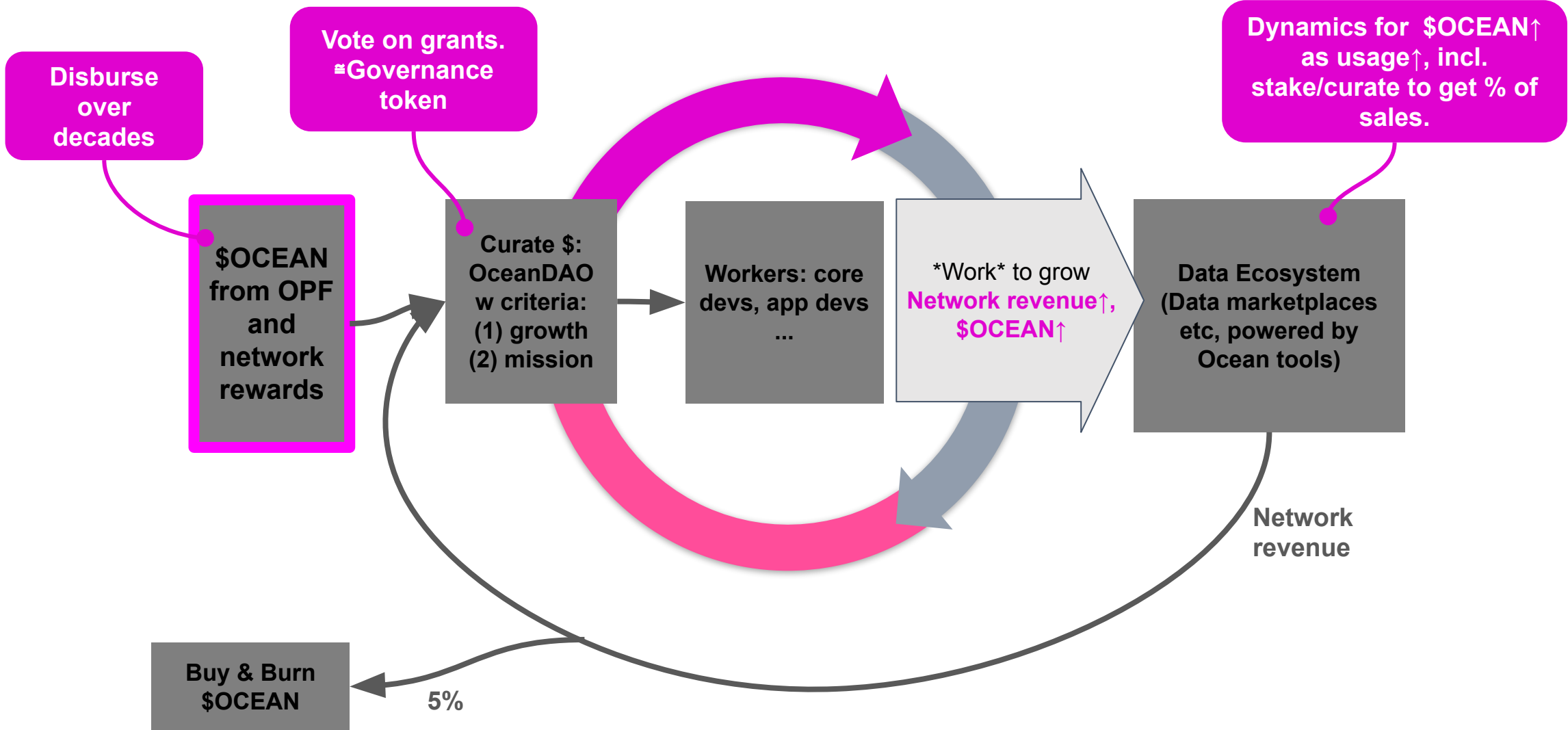
But adapted for the Ocean Web3 ecosystem:
Give space for the community to discover more value.



Burn a % of all revenue so
that stakeholders benefit
from revenue growth.

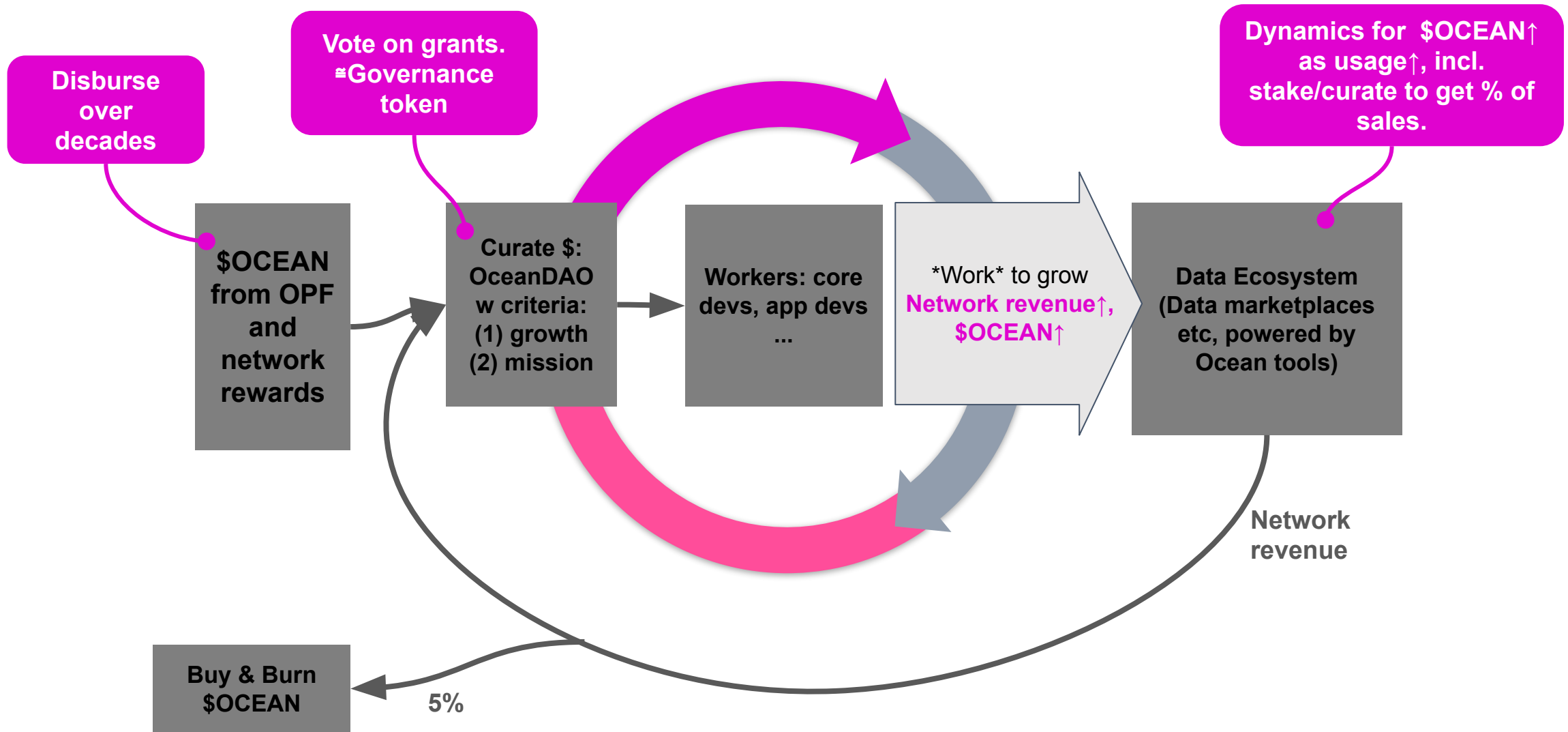


- Q1. How to ensure long-term sustainability? = Ensure long-term funding to core devs etc?
- Q2: Network revenue will not be significant for 4 or 5 years, and maybe a lot longer. How to fund before that?
- A: Disburse 51% of token supply, over decades to fund work by core devs etc. And use OPF treasury.



Ocean System TE: Design Result → Schematic

New Pattern: Web3 Sustainability Loop



Ocean System TE: Design Result → Sub-block goals

The system-level design led to specific goals for sub-blocks.

-Top-down constraint-driven design methodology [[ref Henry Chang et al](#)].

Here are the goals for the sub-blocks in Ocean System:

- **Datatoken contracts:** as tx volume goes up, it drives \$OCEAN.
- **OceanDAO:** curation of projects (governance) encourages skin-in-the-game and long-term sustainability
- **Marketplace:** as \$ volume goes up, it drives \$OCEAN. Get “work” and skin-in-the-game by curators, referrers, third-party marketplace owners

To implement:

- Datatoken contracts: implement by taking a % fee in consume.
- OceanDAO: see later section.
- Marketplace: see later section.

Ocean System TE SW Verification

Ocean System TE Verification: Overall

1. **Humans.** Subjective discussions, with increasing # people. 1 → 2 → key stakeholders
 - Discussions among team (& Julien @ Fabric)
2. **Software modeling,** with increasing fidelity. Spreadsheet → agent-based sim
 - more details later - TokenSPICE.
3. **Economic (live).** Can ratchet value-at-risk over time. People can choose risk/reward tradeoff.
 - Doing this!
 - Biggest ratchet of value-at-risk over time: OceanDAO funding. From OPF → from 51% in ratcheted wya.



Ocean System TE Verification: SW Modeling

- We built **TokenSPICE** to model Ocean ecosystem
- Agent-based simulation, in python
- Each “agent” is a class. Has a wallet, and does work to earn \$
- Model the system by wiring up agents, and tracking metrics (kpis)
- It’s easy to adapt for other projects doing Web3 Sustainability Loop, or simply fork it and write new agents for any agent-based simulation
- Initial version at: <https://github.com/oceanprotocol/tokenspice0.1>
- Continually evolving at <https://github.com/oceanprotocol/tokenspice>

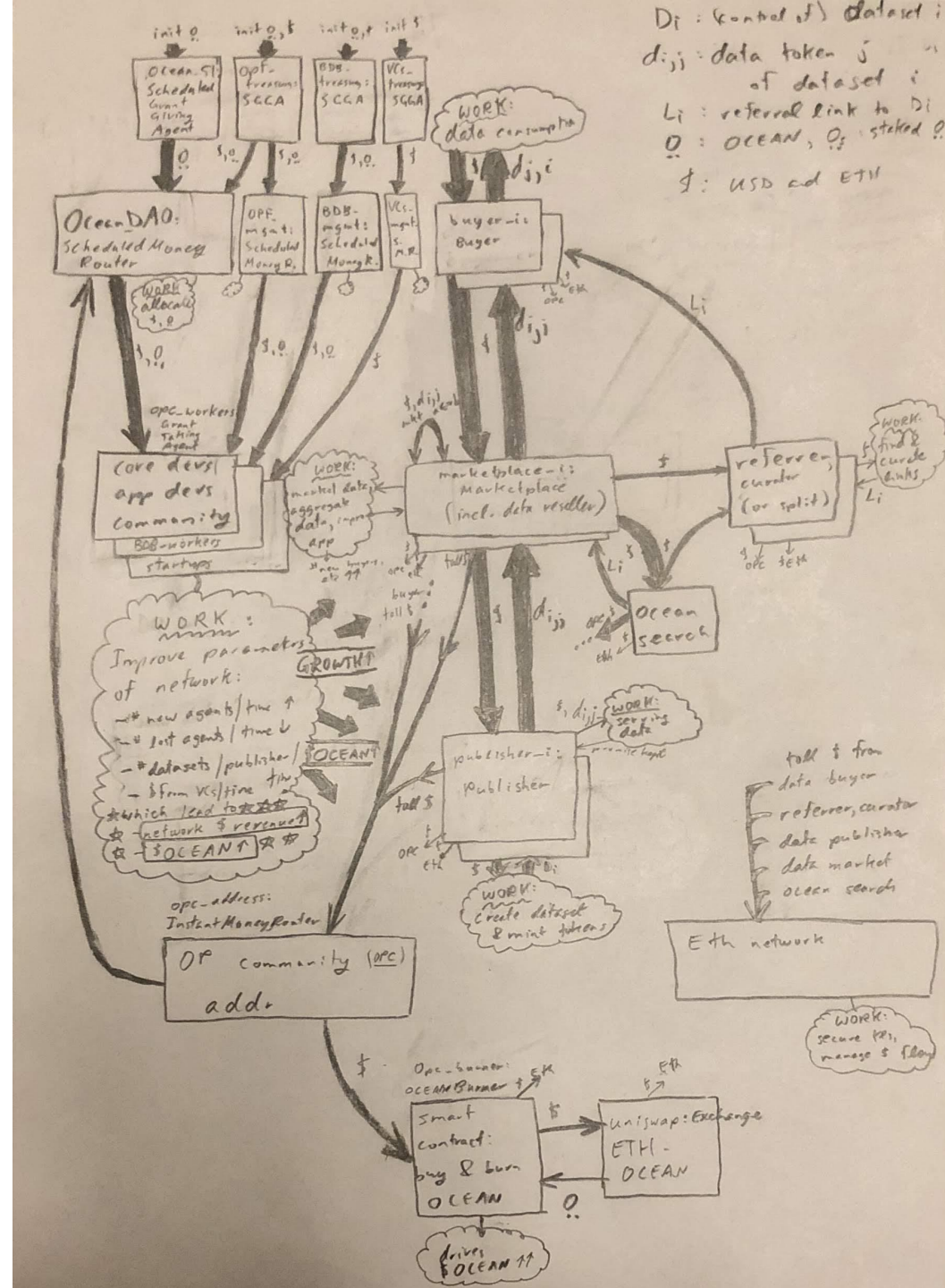


Block diagram: early model

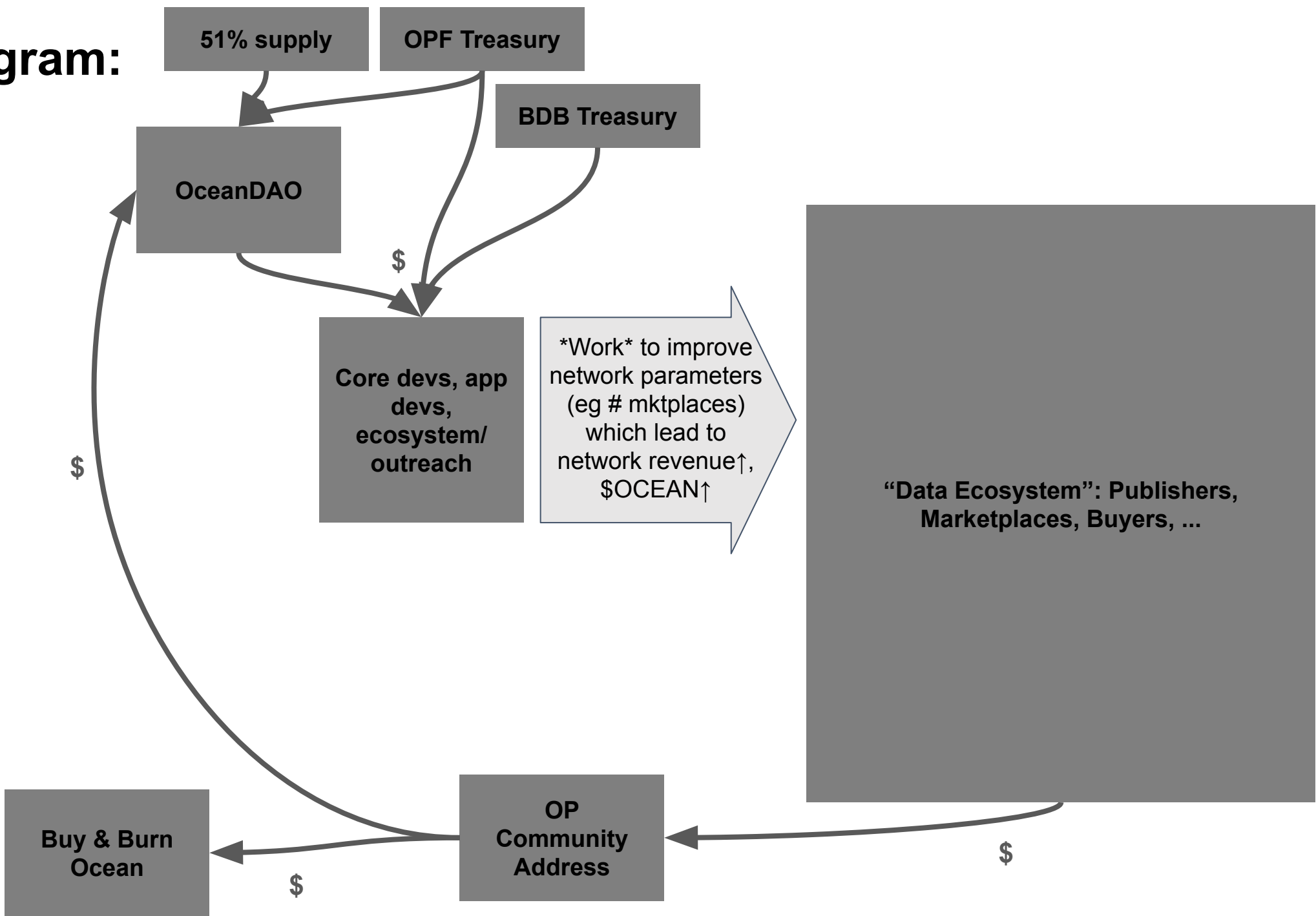
Includes:

- >1 Publisher agents
- >1 Marketplace agents
- >1 Buyer agents
- Referrer / curator
- Ocean search

I [Trent] had started to build this in the tokenSPICE repo; see early commits. However it had a lot of complexity, and my questions were more system-level. So I compressed all of the above into one block simply called "Data ecosystem". See next slide.

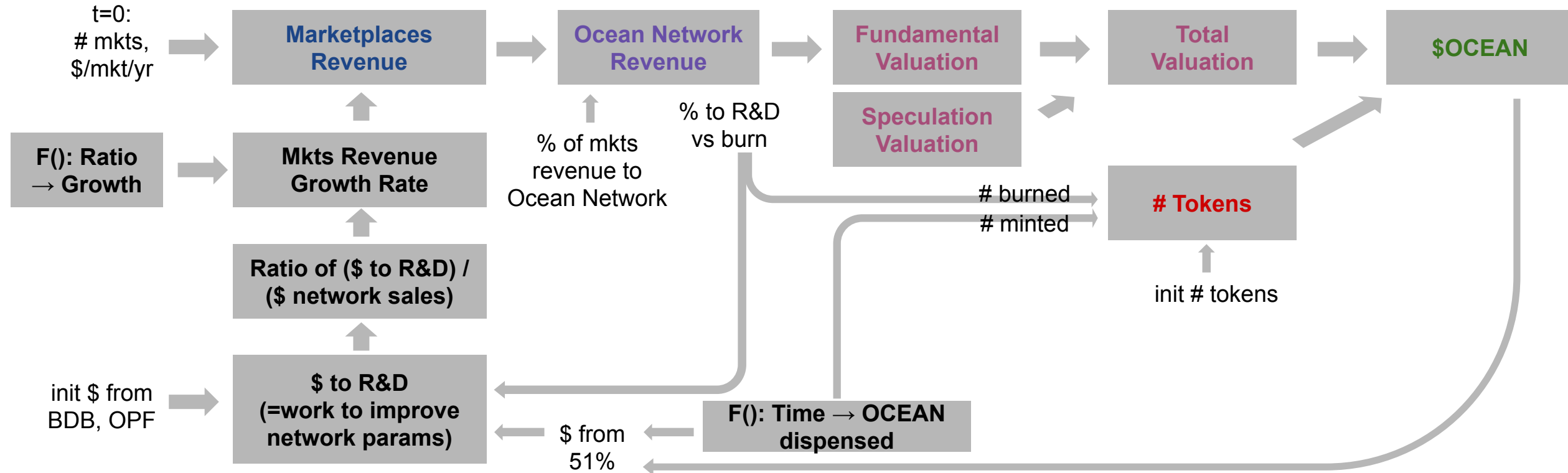


Block diagram: actual

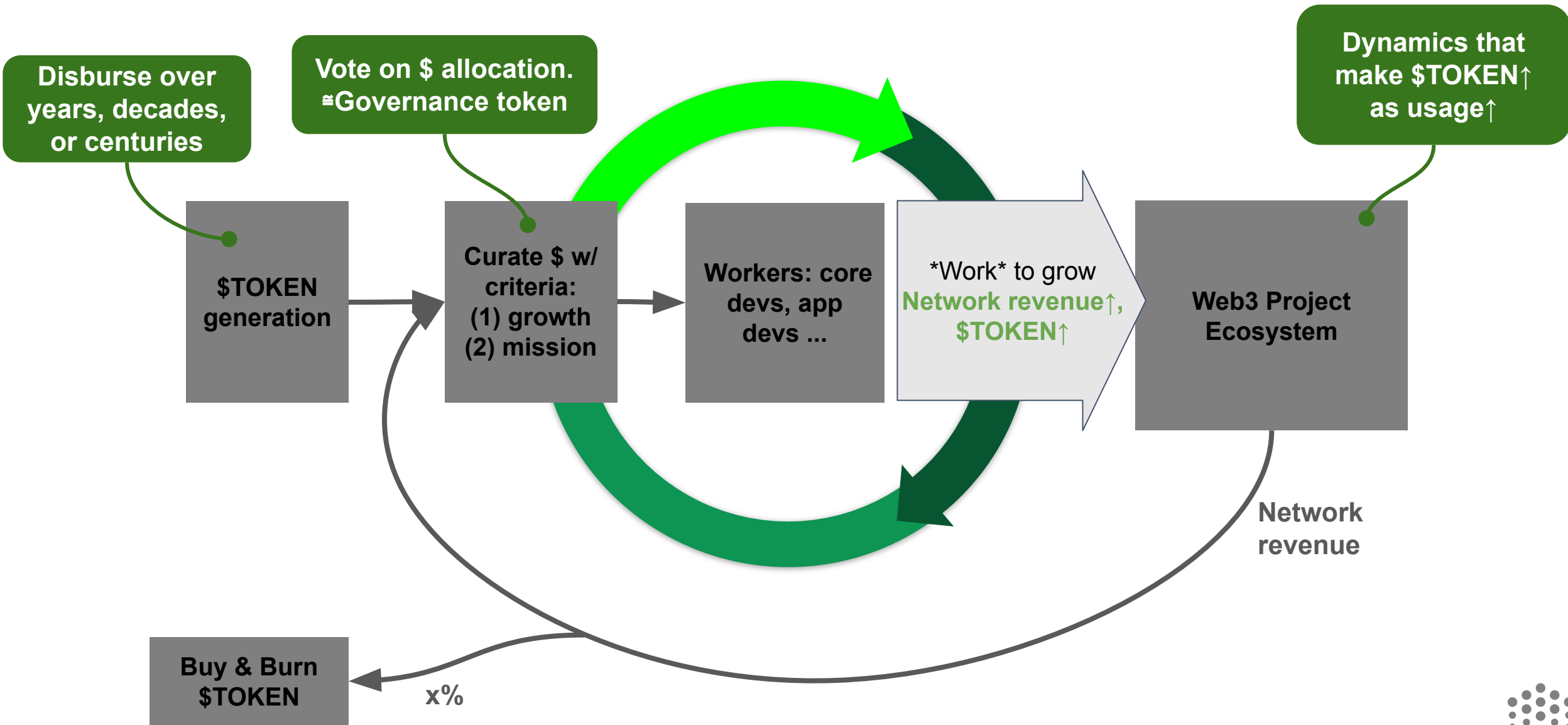


Key variables being modeled

- We can model Ocean revenue and \$OCEAN over time. This helps our decision-making.
 - We can model **marketplaces' revenue**. Depends on initial parameters, and \$ growth rates.
 - From that, we can model **Ocean network revenue**. Depends on % mkts revenue to Ocean network.
 - From that, we can model fundamental **valuation** of Ocean network (e.g. P/S). Can compare this to speculation-based component too.
 - We can also model **# tokens**, including effects of minting and burning
 - From valuation of Ocean network, and # tokens, we can model **\$OCEAN**



Block diagram: simplified version for public



Modeling marketplaces growth rate

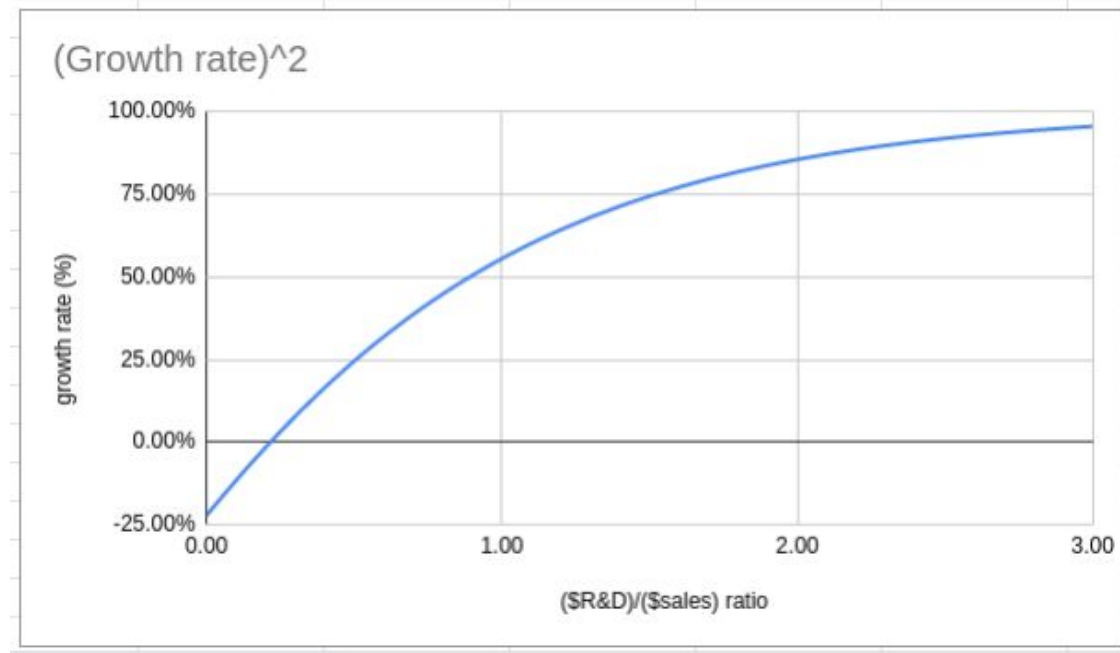
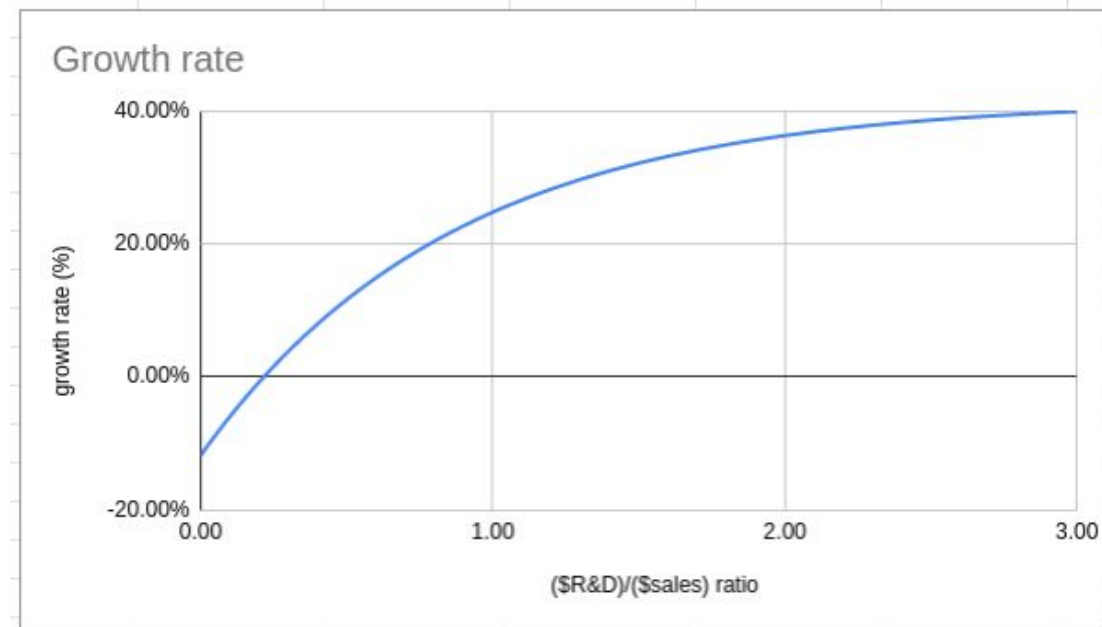
F(): Ratio \rightarrow Growth

Growth should have these characteristics:

- As a function of ratio of (\$ into Ocean R&D) / (\$ Ocean revenue)
- Just like companies!
 - Big slow-growth companies may put 5% of sales into R&D. Ratio = 0.05
 - Small fast-growth companies (i.e. startups) may put 100% or even 300% of sales into R&D. Easy because sales are small.
 - Larger but still fast-growing companies may put 30-50% of sales into R&D (ratio=0.3-0.5), for 20-40% growth. E.g. Facebook, Amazon, Apple.
 - Negative growth if little or no \$ into R&D, whether large or small
- Diminishing returns as more R&D \$ injected

How we model, to capture the target characteristics:

- Model growth as an exponential. This captures diminishing returns.
- Overall marketplaces growth:
 - Has two components: # mkts, \$ rev / market
 - Overall growth is a function of both
 - Growth = $(1 + \text{growth in \# mkts}) * (1 + \text{growth in \$ rev/mkt}) - 1$
- Set parameters for each component as follows:
 - annual growth rate if 0 sales = -11.8%, so that growth rate ^2 is -25%
 - max annual growth rate = 41.5% such that overall rate is 100%
 - growth range = (max annual growth - growth if 0 sales)
 - tau = 0.6. This means: if ratio is 0.6, we'll get 50% of growth range. If ratio is $2*0.6$, we'll get 75% of growth range. Etc. Like half life, but not for time.



Modeling 51% supply schedule

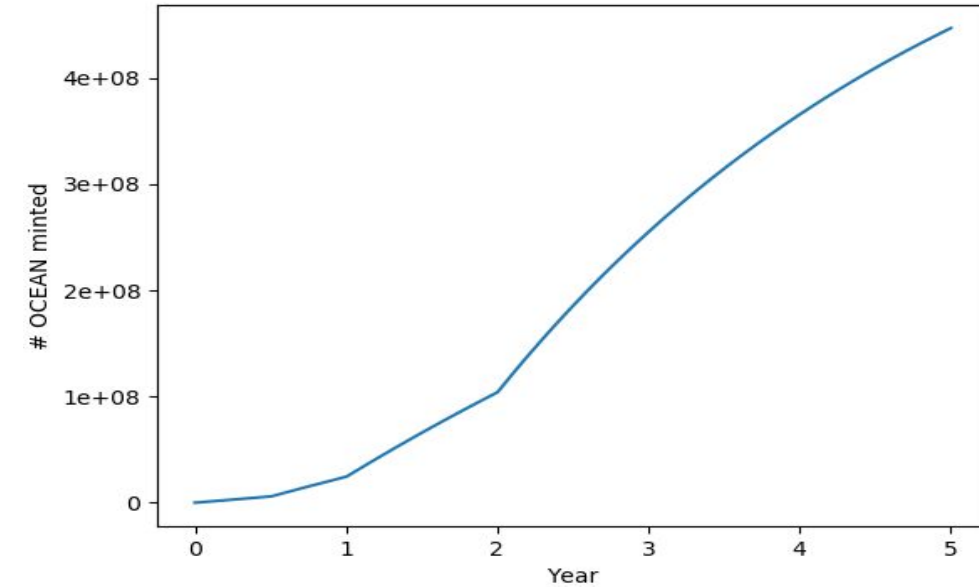
F(): Time → OCEAN dispensed

Concerns to address:

1. Not have sharp dropoff of \$ funding after e.g. 10 years
2. Avoid too much OCEAN entering the market too early, before there's enough liquidity
3. 51% supply is intended for OceanDAO, esp. for R&D. But early OceanDAO will likely take years to stabilize, to be able to handle lots of OCEAN or \$. That is: OceanDAO needs to “bake slowly”, so \$ into it needs to reflect that

How to handle:

- For (1), to avoid sharp dropoff: baseline schedule is **not** uniform for 10 years then stop. Instead, make it an exponential, such that there's funds in 10 years, 20 years, even 50 years.
 - Set the half-life for supply to be 4 years (like Bitcoin). That is, in this baseline, 50% of the (51%) tokens would be dispensed after 4 years, 75% after 8 years, etc. Supply stops after 34 halvings (about 125 years).
- For (2)(3), modify the baseline schedule with a “ratcheting up” in the first few years:
 - Ratcheting schedule (see plot on right):
 - For first 0.5 years: multiply baseline exponential function by 10%
 - For next 0.5 years: 25%
 - For next 1.5 years: 50%
 - Then 100%
 - This schedule ensures R&D funding is approx \$100K/mo for the first decade. After that, funding rises exponentially as \$OCEAN rises exponentially.
 - Ratcheting can be done programatically (“unstoppable”) or manually (until the final 100%, at which time hardcoded). Manually may be more pragmatic, so we can handle unforeseen issues, and tune the % for a steadier R&D \$ supply.



TokenSPICE results

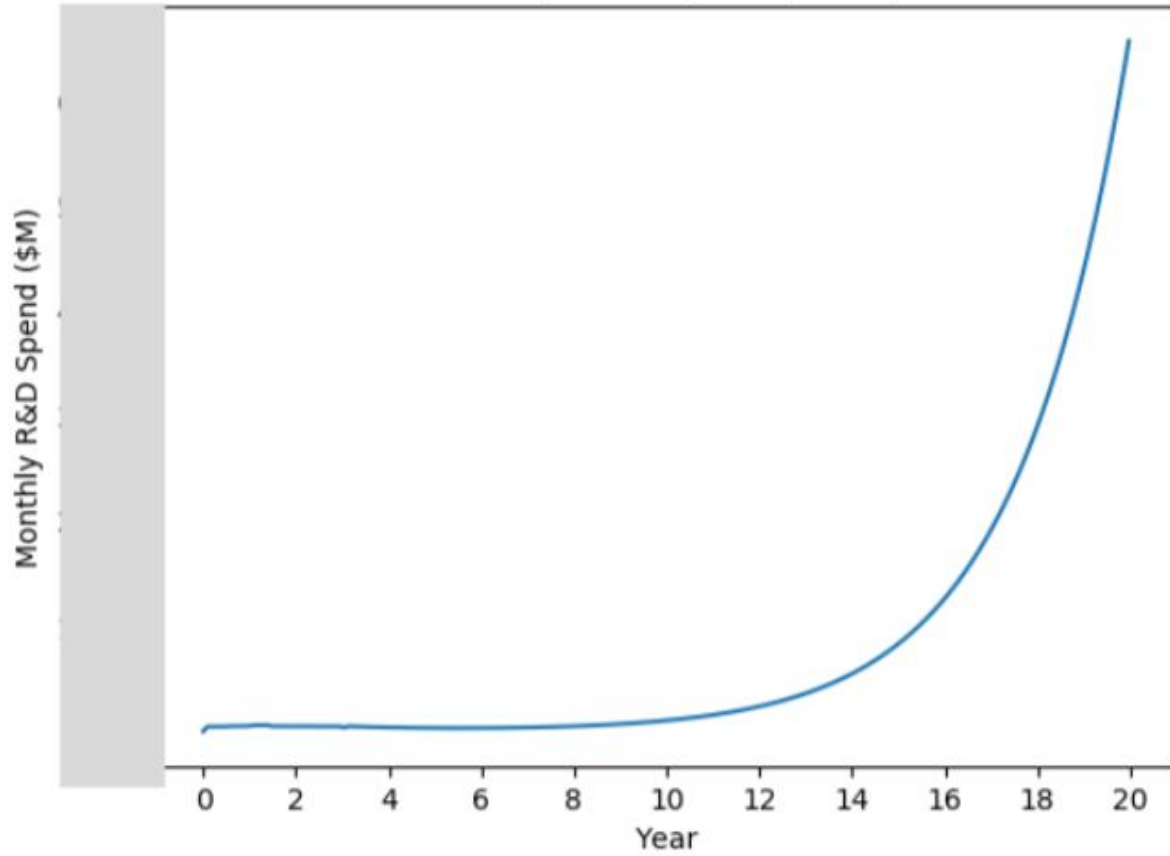
- We have many experiments on [TokenSPICE](#), with many results.
- We put each round of results into GSlides.
- Let's see an example.

Parameter Settings

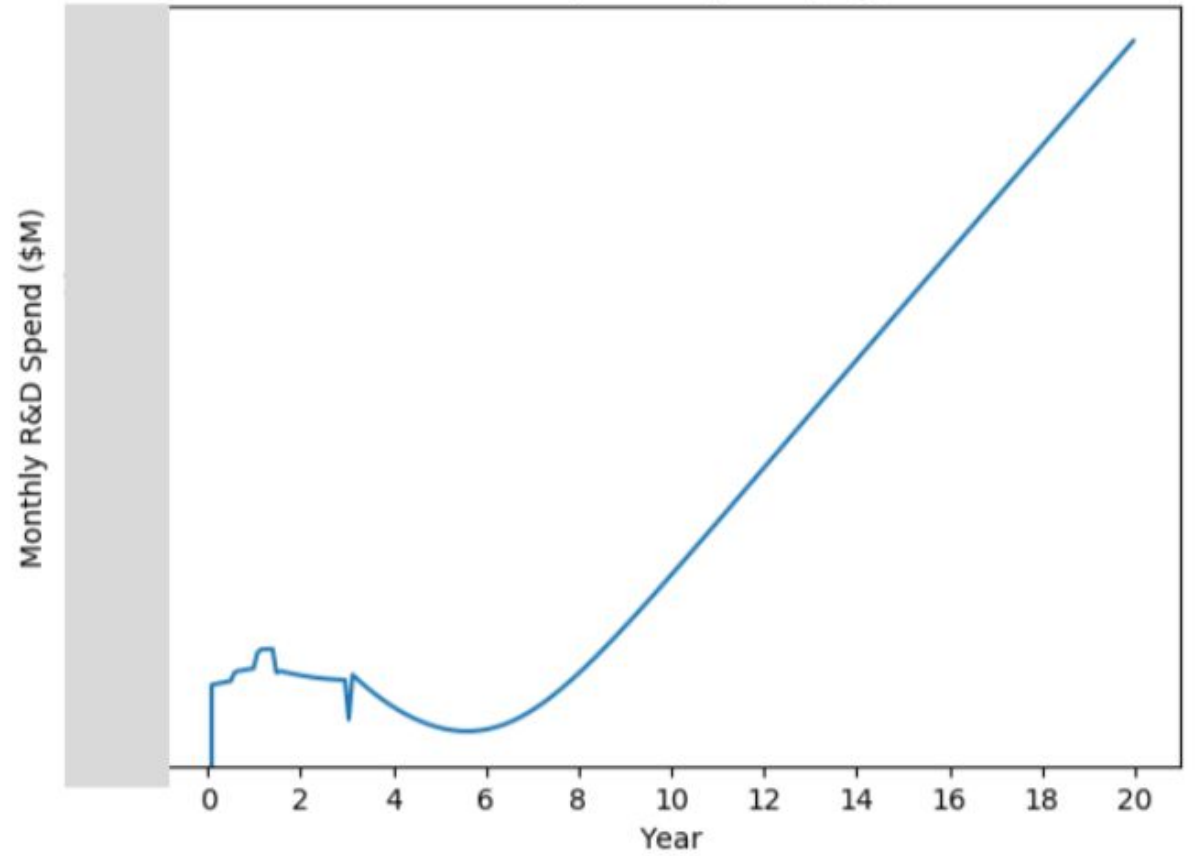
- Simulation time 20 years
- Growth rate info:
 - `growth_rate_if_0_sales` = -11.8% (for total = -25%)
 - `max_growth_rate` = 41.5% (for total = 100%)
 - `tau` = 0.6 (ie ratio needs to be 0.6 just for half the total range. MUCH higher than before)
 - `$ R&D` = `grantTakersMonthlyRevenueNow()`; `$ sales` = `oceanMonthlyRevenueNow()`
- Ocean toll from marketplaces revenue: ____
- Speculation valuation at t=0: ____
- Growth rate of speculation valuation: ____ / year
- Fundamentals valuation approach: P/S = 30x
- % of revenue to burn directly: 5%
- Ramped exponential minting: like right side of 20200505: H=4.0, T0=0.5, T1=1.0, T2=1.4, T3=3.0, M1=0.10, M2=0.25, M3=0.50. Stop after 34 halvings (about 125 years)
- DAO is funded by:
 - minting
 - OPF: uniformly per month over 36 months
 - BDB: “”, 17 months

Monthly R&D Spend

Monthly R&D Spend (linear)

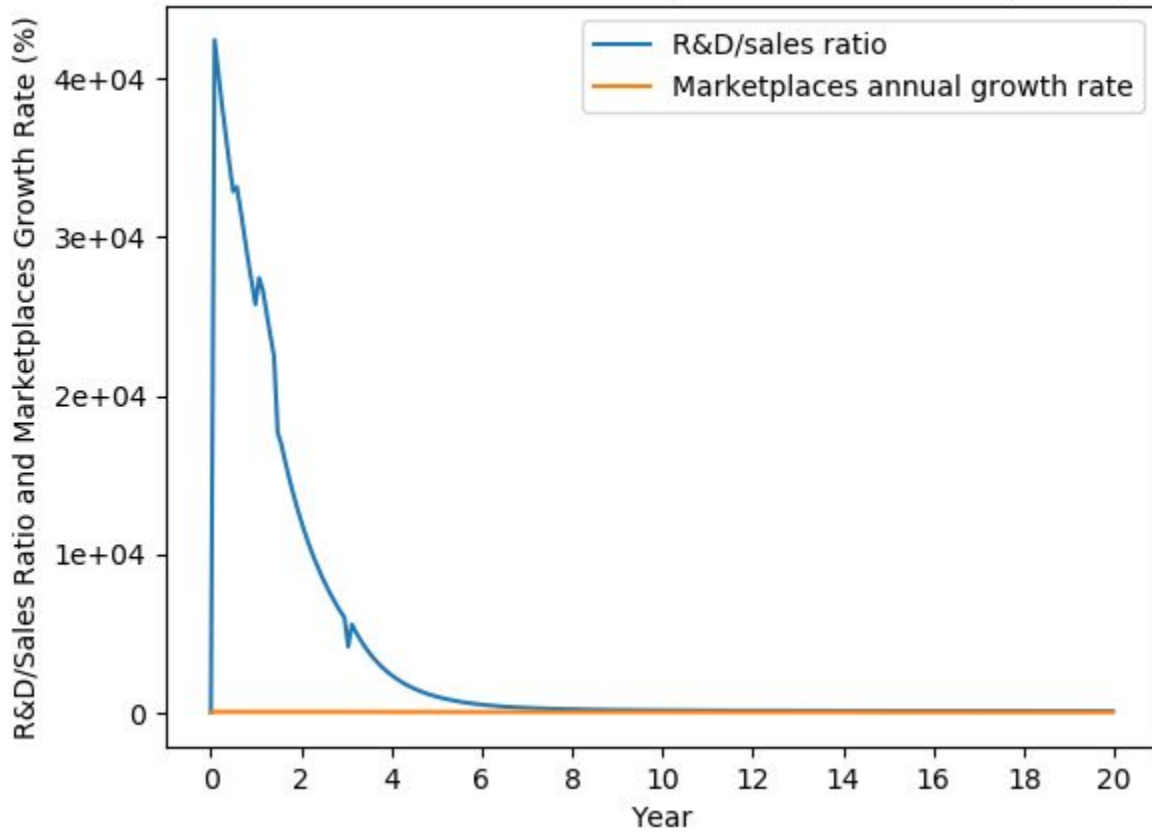


Monthly R&D Spend (log)

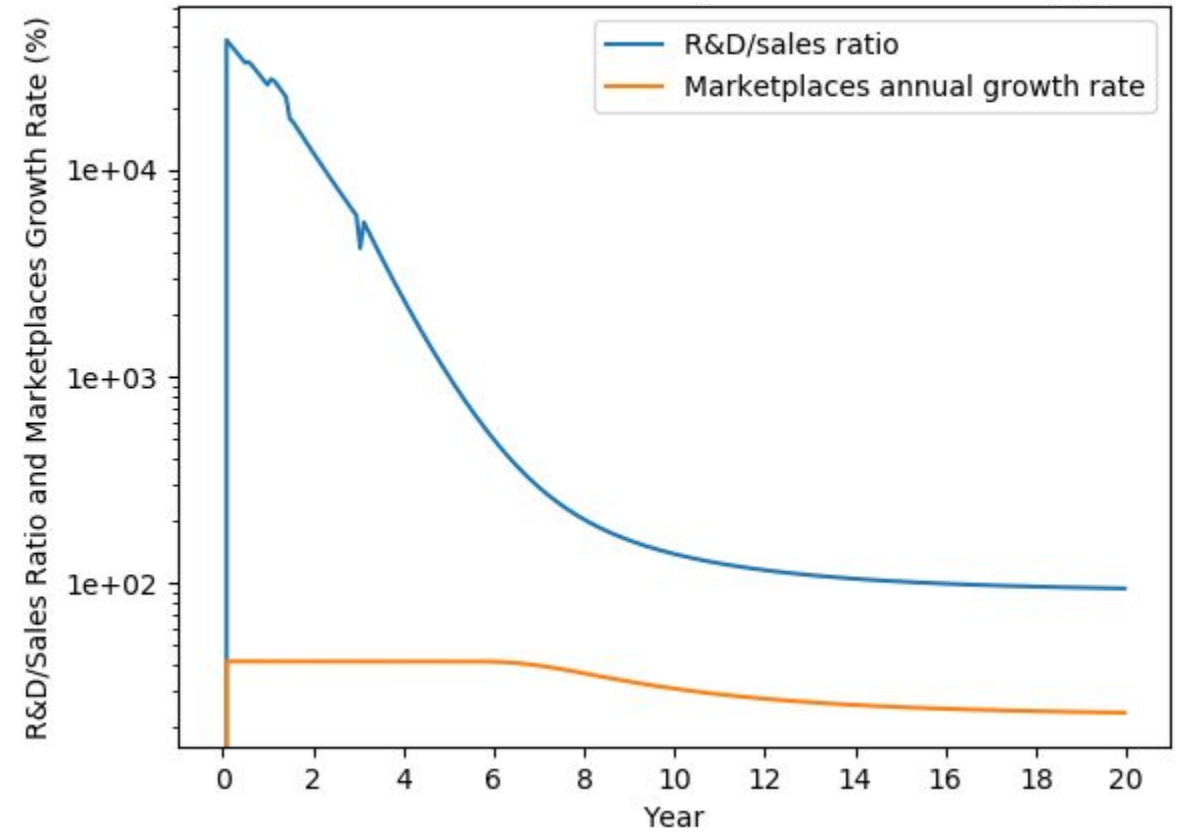


R&D/Sales Ratio, Marketplaces Growth Rate

R&D/Sales Ratio and Marketplaces Growth Rate (linear)

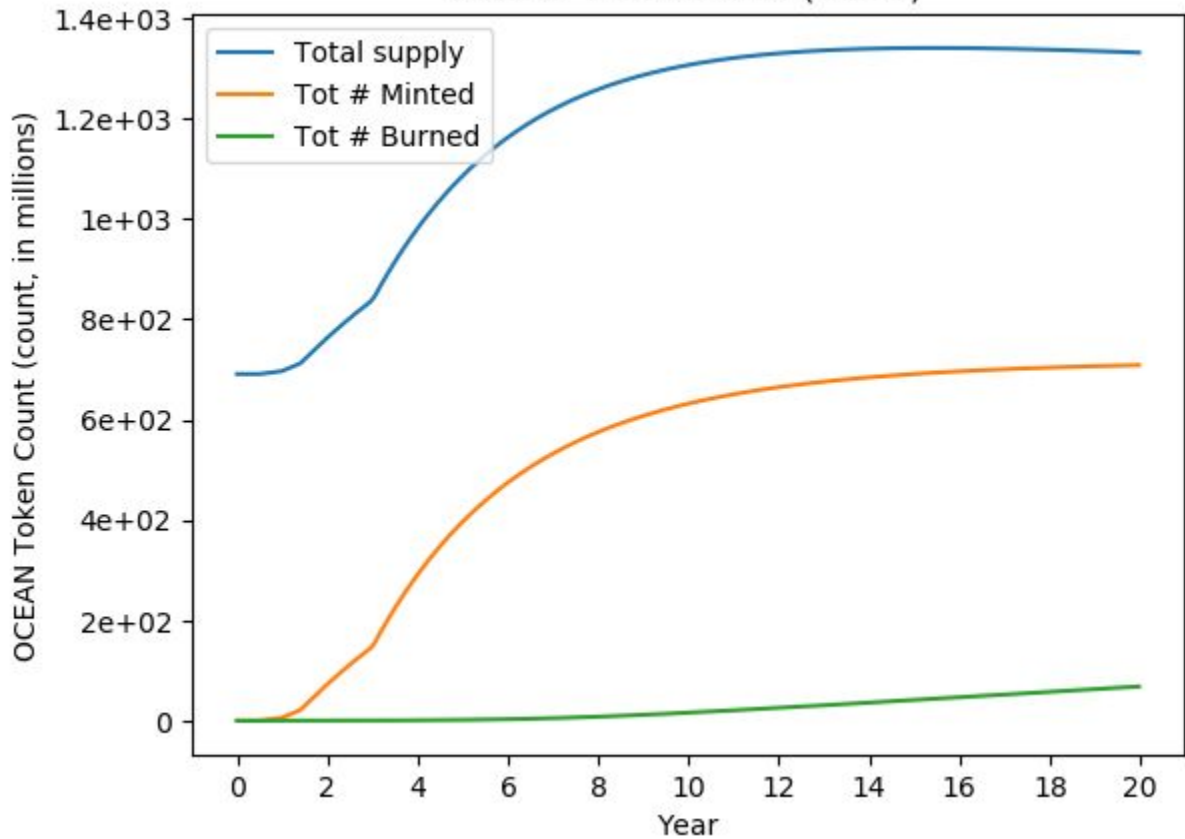


R&D/Sales Ratio and Marketplaces Growth Rate (log)

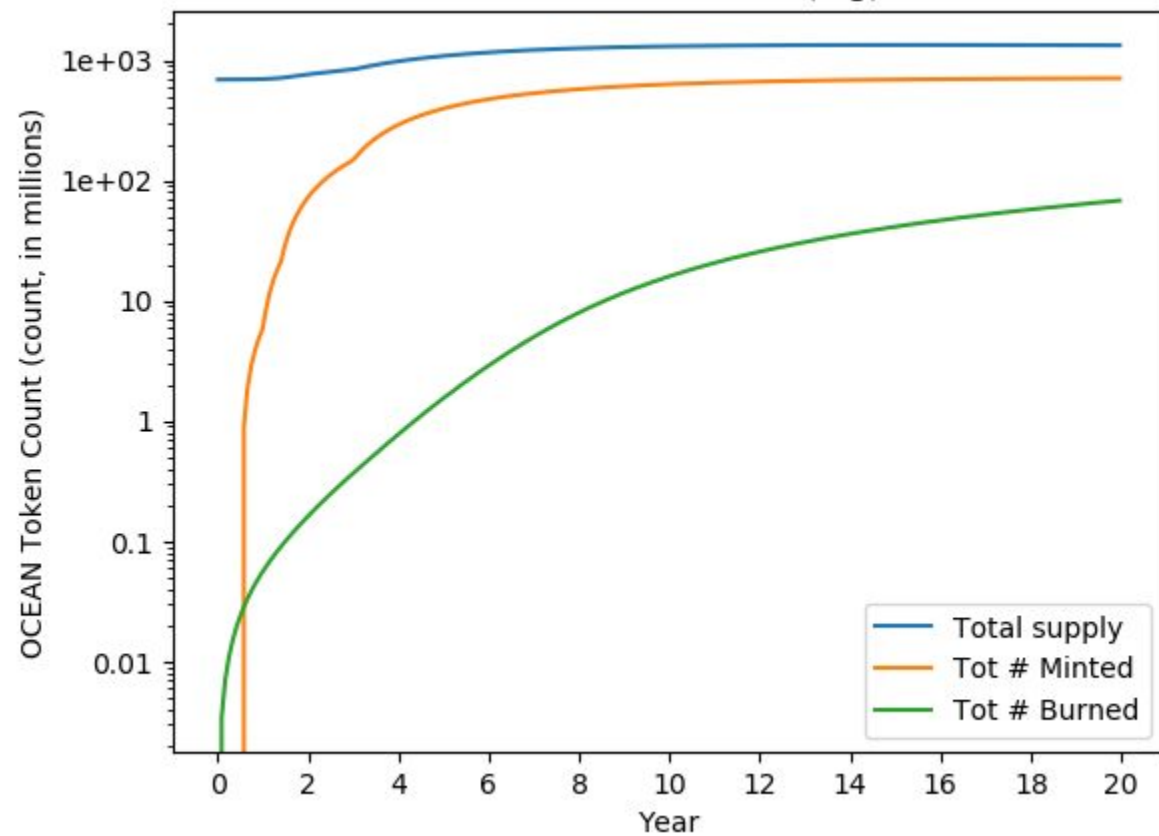


Token count

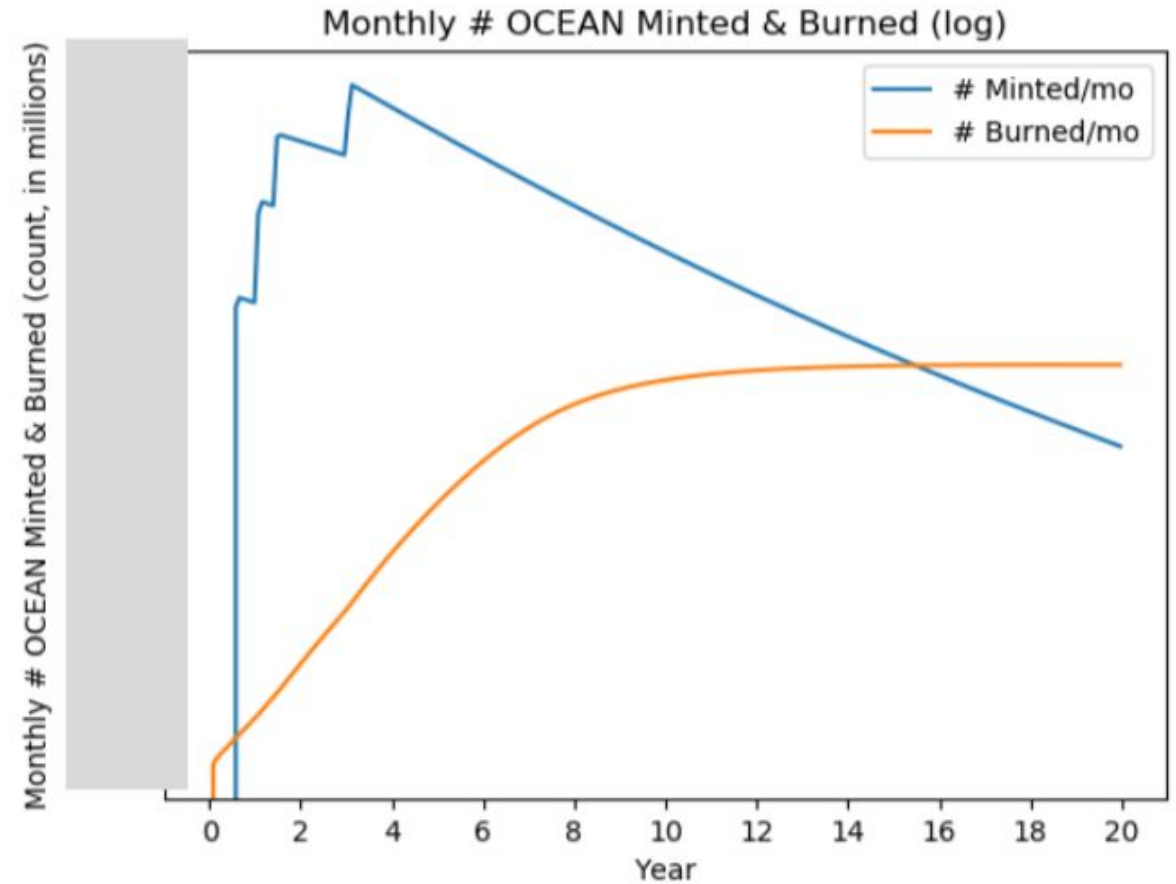
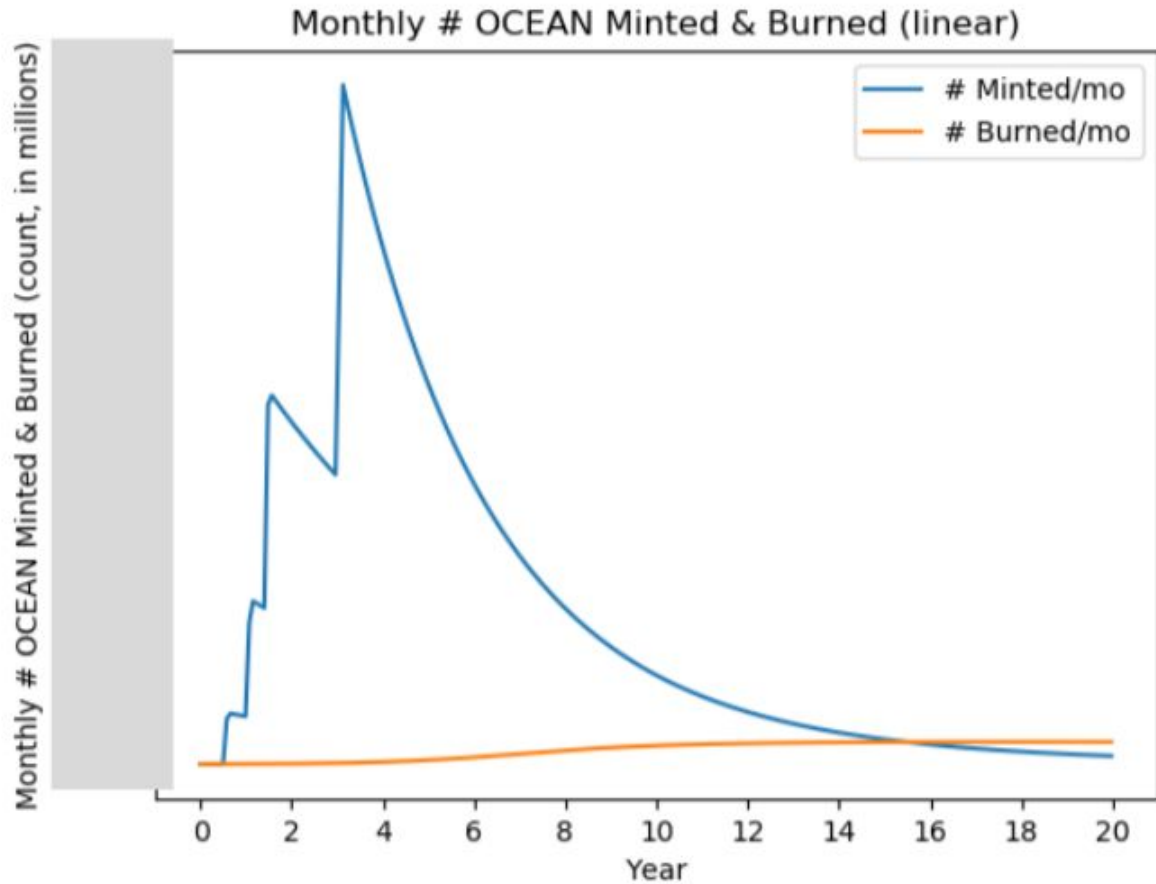
OCEAN Token Count (linear)



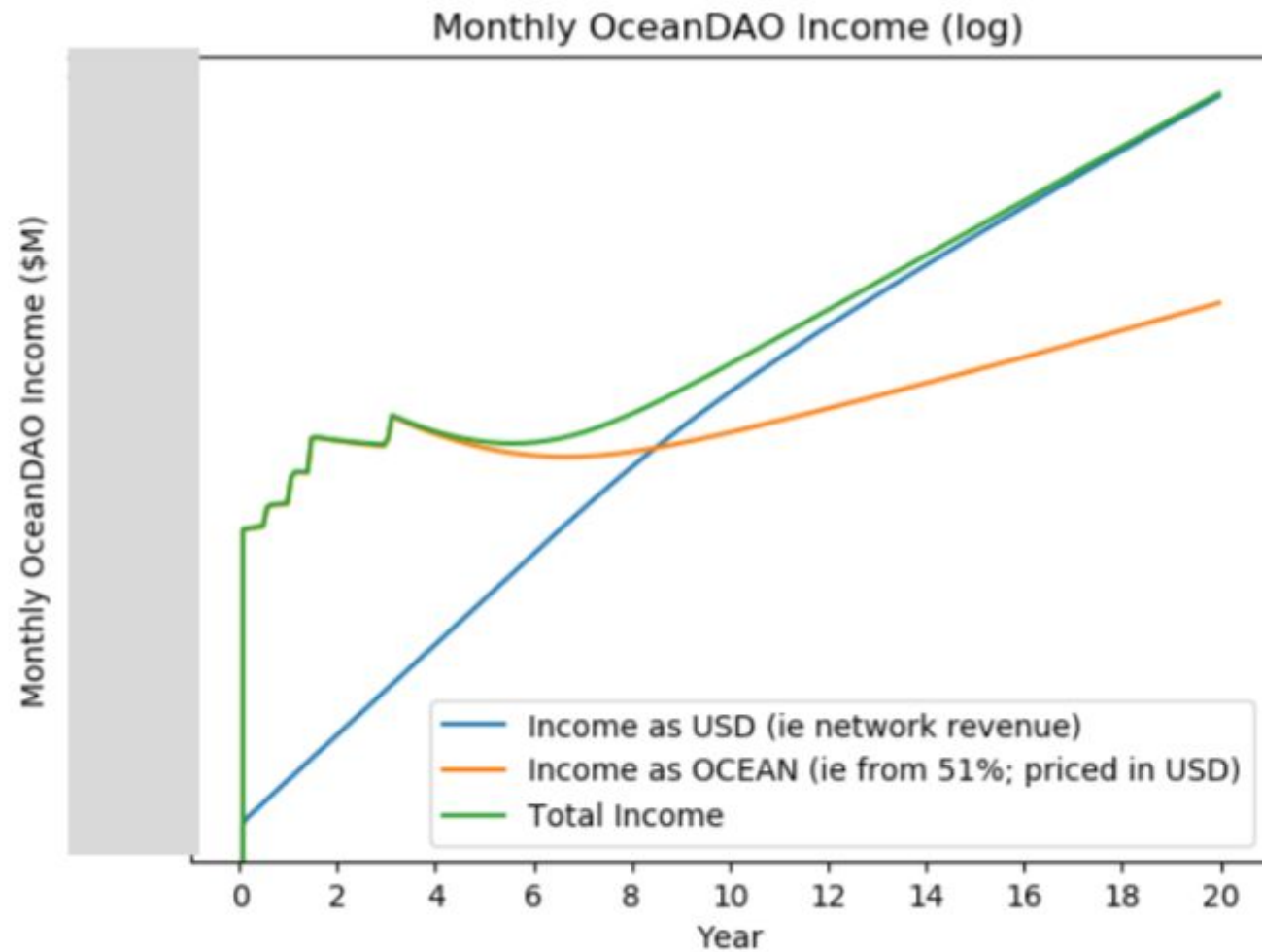
OCEAN Token Count (log)



Monthly # OCEAN minted & burned



DAO Income



**Questions,
With Answers from Modeling Experiments**



Q: Benefit of Worker-51% schedule?

- Q: How much do we need 51%?
- Results: 51% drives 30x more value to \$OCEAN
- Conclusion: use 51%

Note: this sheet is filled manually

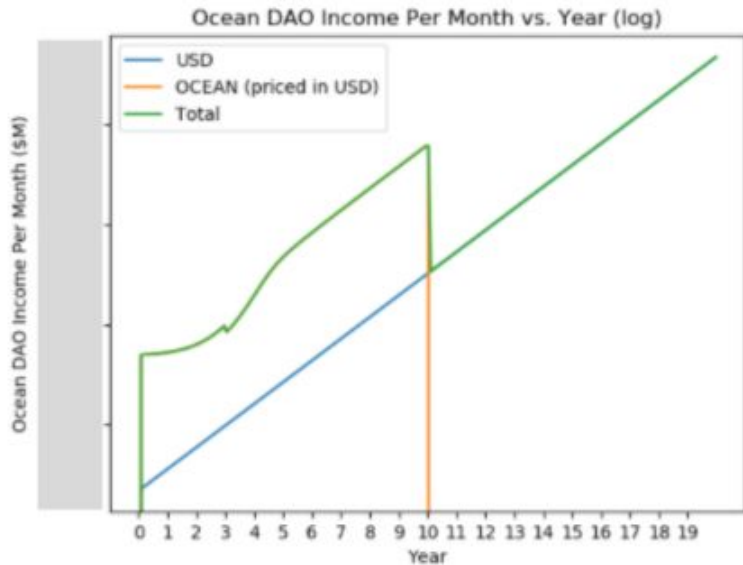
Tokens deployed in DAO	VALUATION IN 2028 (DCF)	VALUATION IN 2028 (P/E)	TOKEN PRICE IN 2028 (DCF)	TOKEN PRICE IN 2028 (P/E)
100M				1
200M				5
300M				3
400M				3
500M				3
600M				3
700M				3
800M				5
886M)



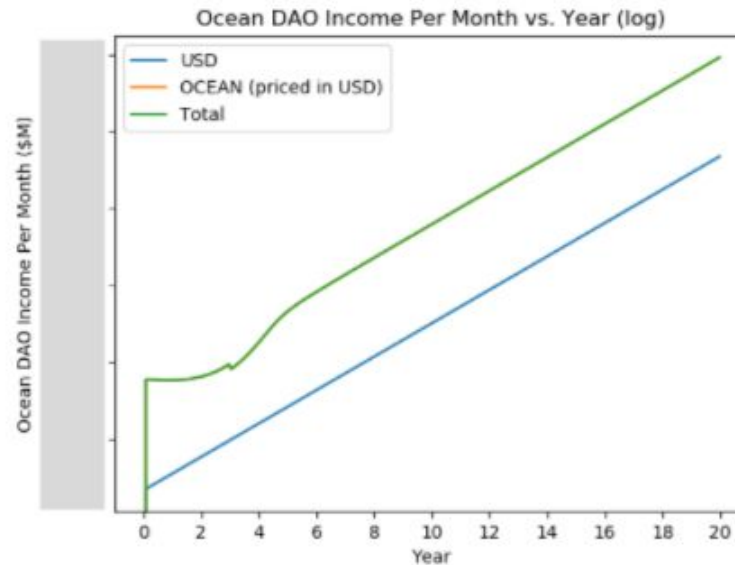
Q: Best Schedule for 51% distribution?

- Q: What is the best schedule for 51% distribution?
- Results:
 - Run 1: distribute uniformly over 10 years. Image below left.
 - Observed: funding dropoff is too sharp.
 - Run 2: Bitcoin-style exponential. Image below middle
 - Observed: it solves dropoff, but in early years too aggressive: much \$ & downwards \$OCEAN pressure.
 - Run 3: ratcheted exponential. Image below right
 - Observed: it solves dropoff, not aggressive in early years and allows “bake slowly” with manual intervention.
- Conclusion: Ratcheted exponential is best; use it.

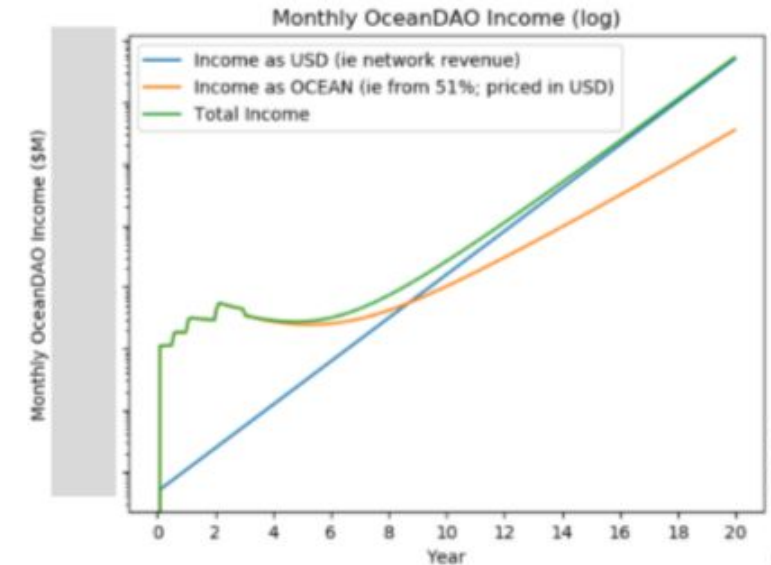
Run 1: uniform



Run 2: exponential



Run 3: ratcheted exponential



Ocean Market V3 TE

Ocean Market V3 TE: Process

- Goals
- Design
- Implementation
- Verification

Ocean Market V3 TE: Goals

3PM = Third Party Marketplace (e.g. dexFreight)

Main:

- Drives value of \$OCEAN: as mkt \$ vol goes up, \$OCEAN goes up
- Incentivizes people to “do work”, aka add value such as more datasets or curation
- Drives virality, i.e. incentivizes people to refer others to Ocean
- Basic design is simple to understand and to communicate. (2nd-order complexities are ok, if needed)
- Each 3PM can also get all the characteristics here. E.g. virality
- 3PMs drive data liquidity to Ocean Market: ie aiding discovery by OPM

Secondary (generally straightforward to solve, once main are solved):

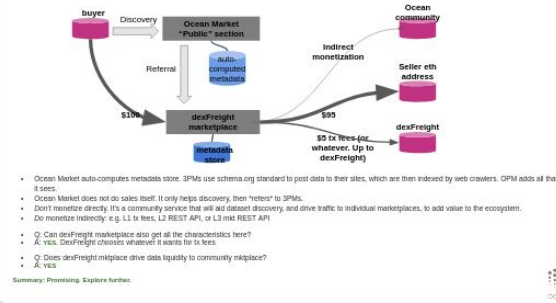
- Design accounts for Ocean Market, *and* for each 3PM
- Incentivizes people to learn about Ocean implicitly, via a more specific extrinsic incentive
- Actually does something useful. I.e. does not bolt on something useless
- Reasonable to implement & maintain
- For the live deployment, de-risk by ratchet up skin-in-game over time
- Accounts for our actual numbers: actual token supply, liquidity now, liquidity in future
- Avoid front running and flash staking. Eg. see a buy tx, flash stake, earn \$\$\$. A solution: need to stake for >24 h to earn.

Ocean Market V3 TE: Design exploration

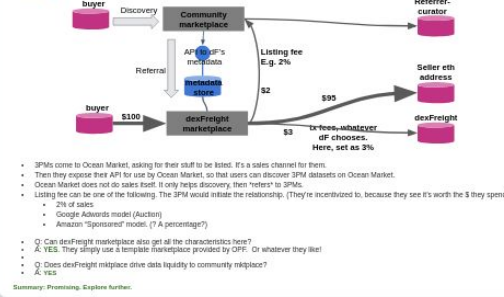
Many designs were explored against the criteria.

Appendix: Marketplace TE: Account for 3PM: Promising Designs

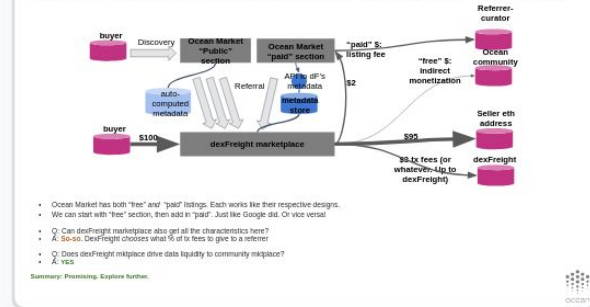
Cand A: Ocean Market as sales channel for 3PM (metasearch), "free" listing



Cand B: Ocean Market as sales channel for 3PM (metasearch), "paid" listing



Cand C: Ocean Market as sales channel for 3PM (metasearch), "free" + "paid" listing



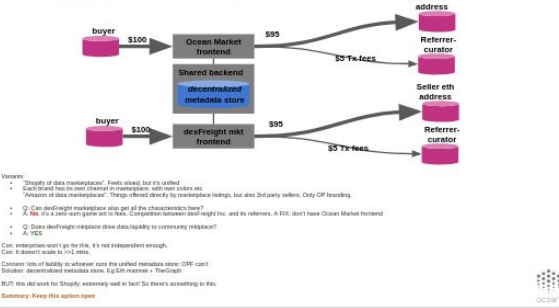
109

110

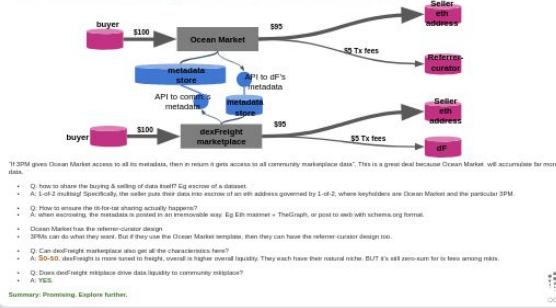
111

112

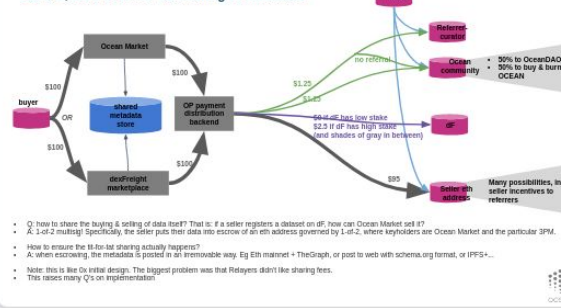
Cand D: Shared metadata & backend



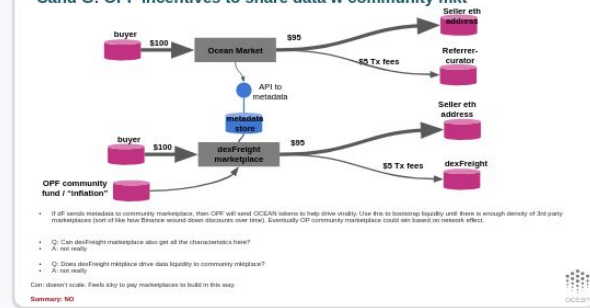
Cand E: tit-for-tat sharing of metadata Ocean Market -- 3PM



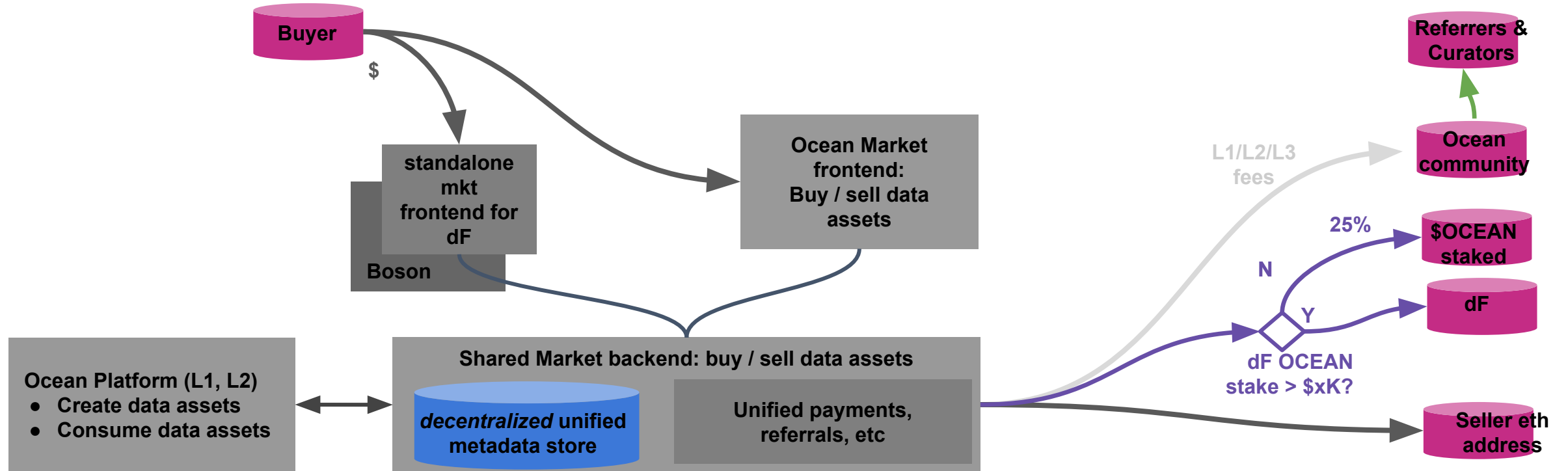
Cand E, cont'd: tit-for-tat sharing of metadata



Cand G: OPF incentives to share data w community mkt

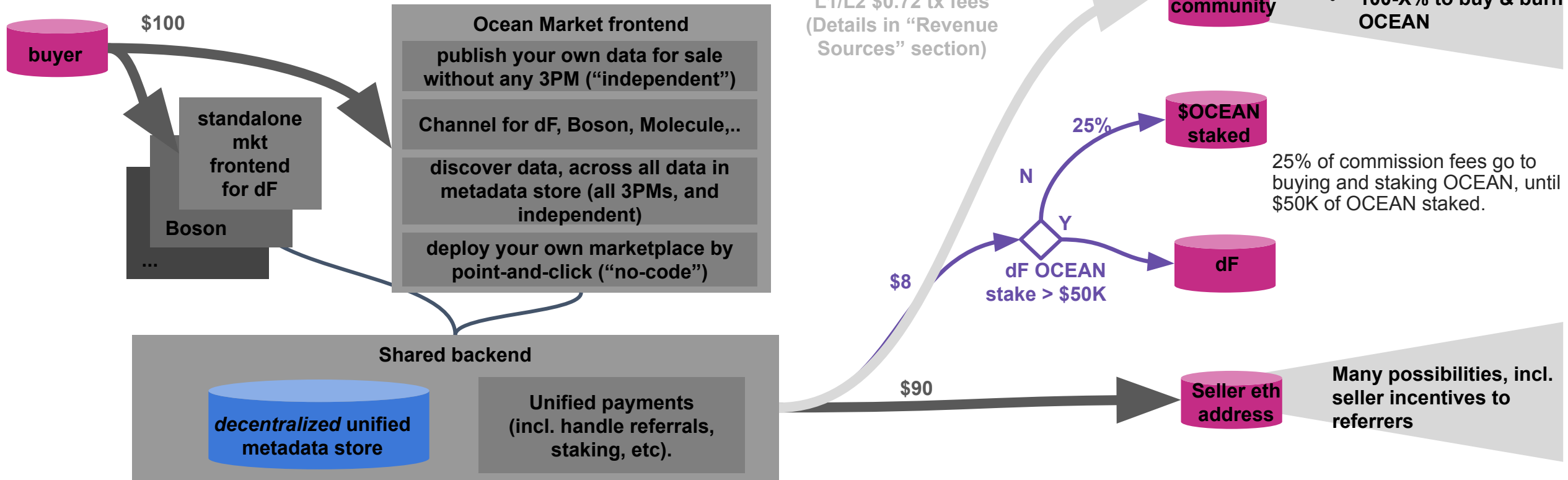


Ocean Market V3 TE: Chosen design



- As \$ volume goes up, it drives \$OCEAN.
- Gets “work” and skin-in-the-game by curators, referrers, third-party marketplace owners
 - If you’re doing referrals and you drive volume↑, for more rewards you need stake↑
 - Same for curation

Ocean Market V3 TE: Detailed version of chosen design



Concern: liability to whoever runs the unified metadata store; OPF can't. Solution: *decentralized* metadata store. Options: (1) Eth mainnet + TheGraph, (2) IPFS + pinning, maybe Filecoin (3) arweave + some search, (4) [maybe] publish on the Web, and then the Ocean Market crawler *only* crawls the websites of Publishers. All must follow schema.org schema.

Concern: for many 3PMs, give up too much control, so they won't use this. Answers: we can still capture revenue at L1, maybe L2 REST API, and maybe other L3 stuff like Oceansearch. And, we should make the Ocean Market frontend compelling enough for most people to stick around. E.g. more convenience, UX, liquidity. Just like Shopify did.

Top right: Curation is nice, referral is even more important. Because referral drives more traffic to the platform. So pay referral more. If you refer only, you earn a nice referral fee. If you refer & curate, you earn a bigger one. If you just curate, you earn a smaller one.

Daily referrer reward = $\max(\$1, \min(\sum_i P\% * \text{daily_sales}, 5\% * \text{amt_staked}))$

- Give referrers a little bit even if they haven't staked anything
- Only get big reward if big sales *and* big amt_staked.

• Reward for just curation: FIXME

Ocean Market V3 TE: Implementation

A realization: using AMMs implements the “Chosen TE design”, and meets “TE Goals”.

Design details:

- Datatoken-OCEAN AMMs. LPing = staking = curating. LPs get a % of swap volume.
- Store metadata on-chain
- Deploy to Ethereum mainnet
- Datatoken consume() sends a % to marketplace runner, and to Ocean community

How the Market design implements the “Chosen TE design”:

- As \$ volume goes up, it drives \$OCEAN.
 - [YES - as \$ volume up, more OCEAN is staked, driving \$OCEAN]
 - [YES - as \$ volume up, the \$ from % fees goes up, some of that goes to burning, driving \$OCEAN]
- Gets “work” and skin-in-the-game by curators, referrers, third-party marketplace owners
 - [YES - LP rewards] If you’re doing referrals and you drive volume↑, for more rewards you need stake↑
 - [YES - LP rewards] Same for curation

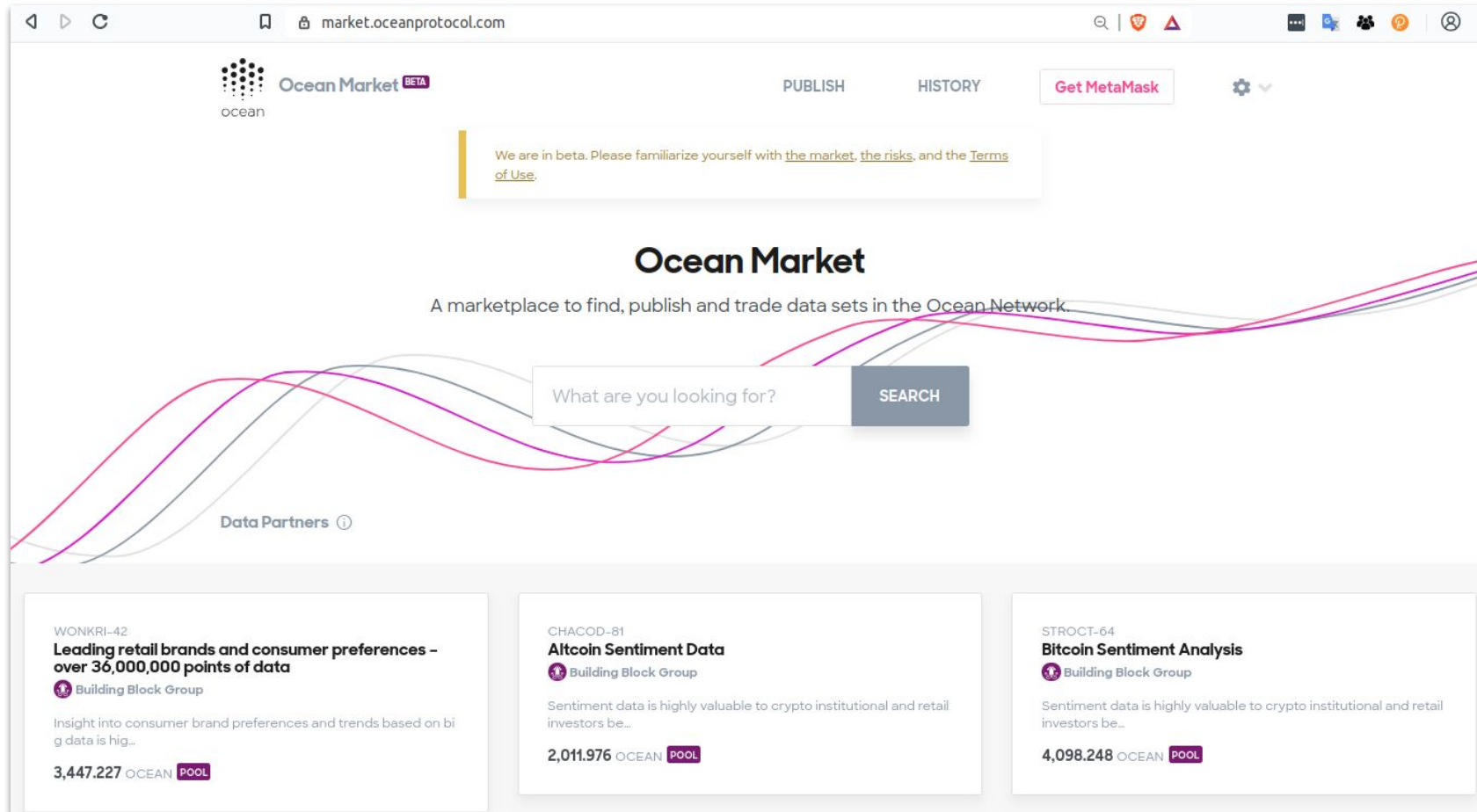
How it meets “TE Goals”:

- [YES] Drives value of \$OCEAN: as mkt \$ vol goes up, \$OCEAN goes up
- [YES] Incentivizes people to “do work”, aka add value such as more datasets or curation
- [YES] Drives virality, i.e. incentivizes people to refer others to Ocean
- [YES] Basic design is simple to understand and to communicate.
- [YES] Each 3PM can also get all the characteristics here. E.g. virality
- [YES] 3PMs drive data liquidity to Ocean Market: ie aiding discovery by OPM

Ocean Market V3 TE: Verification

1. **Humans.** Subjective discussions, with increasing # people. 1 → 2 → key stakeholders
 - Discussions among team, and Julien @ Fabric)
 - Discussions with Fernando @ Balancer
2. **Software modeling**, with increasing fidelity. Spreadsheet → agent-based sim
 - Built Py & JS drivers for Balancer, and make extensive unit tests
 - Did *not* do high-fidelity simulations of token dynamics. Why: (a) AMMs are already live (b) given the first point it wasn't worth the time commitment.
3. **Economic (live).** Can ratchet value-at-risk over time. People can choose risk/reward tradeoff.
 - Launched Ocean Market with lots of writings & caveats (e.g. “beta”). “Test in prod” ;)
 - People did choose risk/reward tradeoff. Some made \$, some lost, some simply tested.
 - Observed community response to Ocean Market, and token dynamics.
 - Made adjustments accordingly. Being live was key to rapid improvements in what mattered.
 - Most notable TE adjustment: 10/90 OCEAN/DT → 50/50 → 70/30. It helped a *lot*.
 - Further TE improvements identified around “Better IDOs” and more. For Ocean V4.1.

Ocean V3 (with Ocean Market) went live in the fall



The screenshot shows the Ocean Market website interface. At the top, the URL is market.oceanprotocol.com. The header includes the Ocean Market logo with a 'BETA' badge, navigation links for 'PUBLISH' and 'HISTORY', and a 'Get MetaMask' button. A beta notice states: 'We are in beta. Please familiarize yourself with [the market](#), [the risks](#), and the [Terms of Use](#).' The main heading is 'Ocean Market' with the subtitle 'A marketplace to find, publish and trade data sets in the Ocean Network.' Below this is a search bar with the placeholder text 'What are you looking for?' and a 'SEARCH' button. A 'Data Partners' link is also visible. The main content area features three data set listings:

ID	Dataset Name	Provider	Price (OCEAN)
WONKRI-42	Leading retail brands and consumer preferences - over 36,000,000 points of data	Building Block Group	3,447.227
CHACOD-81	Altcoin Sentiment Data	Building Block Group	2,011.976
STROCT-64	Bitcoin Sentiment Analysis	Building Block Group	4,098.248

<http://market.oceanprotocol.com>



Ocean Market V4.1 TE

Ocean Market V4.1 TE: Process

- Goals
- Design
- Implementation
- Verification
 - Highlight: **TokenSPICE with EVM-in-the-loop** on all of the Ocean smart contracts, including datatokens & factory, Balancer AMMs & factory, etc. So that I can model Ocean Market dynamics with high fidelity.

Ocean Market V4.1 TE: Goals

Simple

- easy to understand. Mental model plays well with existing
- smart contract SW simple: to implement, understand, maintain
- GUI SW simple: to implement, understand, maintain

Avoids large price swings when people just want to stake OCEAN

Solves price spikes at beginning

Solves price spikes in market equilibrium

No risk of exit scam after IDO

Good incentive for publisher to publish initially

Address risk of datatoken price decoupling from what people will pay for it to consume it
//
Incentivize for actual consumption



Ocean Market V4.1 TE: Status Quo V3 Design

Criterion	Ocean V3
Simple -easy to understand. Mental model plays well with existing -smart contract SW simple: to implement, understand, maintain -GUI SW simple: to implement, understand, maintain	✓
Avoids large price swings when people just want to stake OCEAN	✗
Solves price spikes at beginning	✗
Solves price spikes in market equilibrium	✗
No risk of exit scam after IDO	✗✗
Good incentive for publisher to publish initially	✓✓
Address risk of datatoken price decoupling from what people will pay for it to consume it // Incentivize for actual consumption	✗

Mental Model: Life Cycle of a Data Asset

1. **Publish DT contract.** No tokens minted yet. Token cap set.
2. **IDO // Burn-In Phase.** Initial tokens are distributed to a primary market, according to some rules.
3. **Equilibrium Phase.** Token's on the open market without rails. >0 secondary markets.

Sometimes 2 & 3 can blend together.

Ocean Market V3 Design, in context of Life Cycle

1. **Publish DT contract.**
2. **IDO // Burn-In Phase.**
 - Publisher creates a pool
 - Publisher mints 10-100 DTs into pool, alongside 70% OCEAN tokens s.t. price is ok
 - That's it!
3. **Equilibrium Phase.**
 - The initial pool tends to be the largest market
 - There can be secondary markets. E.g. some publishers have put DT pools on Uniswap.
 - Publisher is able to mint more DTs to inject into any market. They may have a *lot* of DTs.

Ocean Market V3 Design - Issues in Context of Life Cycle (there are other framings too)

Here's what's happening (**bold** = problems):

1. Publisher publishes a dataset. To not have to put in too much OCEAN stake for their initial price, they pick the minimum # DTs to publish
2. Stakers come along and stake. **Price skyrockets**. Low supply, high demand -> high price.
3. **Price is highly volatile**, because so few DTs
4. There's no GUI affordance for publisher to add more DT. If there was one, **publishers may dump DTs**.

We've seen this problem before. It's the question of how do you release a crypto token to the market, to get the token into many hands that want it, to get a decent price, to get decent liquidity, etc.

This is the realm of fund raising of any token project, with tools like vesting, ICOs, IEOs and more. This is what we call an "IDO".

Ocean Market V4.1 TE: Design Iterations 1/2 (w/ Manual Verification)

Criterion	Ocean V3	1SS	LB	1SS + LB	1SS + LB + Dutch:Pub	1SS + LB + Dutch:Pub + rICO	1SS + Dutch:Pub	1SS + Dutch:Pool	1SS + Dutch:Pool + Vested Premine
Simple -easy to understand. Mental model plays well with existing -smart contract SW simple: to implement, understand, maintain -GUI SW simple: to implement, understand, maintain	✓	✓	✓	✓	≈✓	✗	✓	✓	✓
Avoids large price swings when people just want to stake OCEAN	✗	✓	✗	✓	✓	✓	✓	✓	✓
Solves price spikes at beginning	✗	✗	✗	✗	✓	✓	✓	✓	✓
Solves price spikes in market equilibrium	✗	✓	✓	✓	✓	✓	✓	✓	✓
No risk of exit scam after IDO	✗✗	✓	✗	✗	✗	✓	✗	✓	✓
Good incentive for publisher to publish initially	✓✓	✗	✗	✗	✓	✓	✓	✗	✓
Address risk of datatoken price decoupling from what people will pay for it to consume it // Incentivize for actual consumption	✗	✗	✗	✗	✗	✗	✗	✗	✗

Ocean Market V4.1 TE: Design Iterations 2/2 (w/ Manual Verification)

Criterion	Ocean V3	1SS + Dutch:Pool + Vested Premine	1SS + Dutch:Pool + Vested Premine + Soft cap on ratio
Simple -easy to understand. Mental model plays well with existing -smart contract SW simple: to implement, understand, maintain -GUI SW simple: to implement, understand, maintain	✓	✓	≈✓
Avoids large price swings when people just want to stake OCEAN	✗	✓	✓
Solves price spikes at beginning	✗	✓	✓
Solves price spikes in market equilibrium	✗	✓	✓
No risk of exit scam after IDO	✗✗	✓	✓
Good incentive for publisher to publish initially	✓✓	✓	✓
Address risk of datatoken price decoupling from what people will pay for it to consume it // Incentivize for actual consumption	✗	✗	✓

Ocean Market V4.1 TE SW Verification

Outline

- Ocean System TE
- Interlude: On TE Verification
- Ocean System TE - SW Verification w TokenSPICE
- OceanDAO TE
- Ocean Market V3 TE
- Ocean Market V4.1 TE
- **Ocean Market V4.1 TE - SW Verification w TokenSPICE & EVM**
- Conclusion

Ocean Market V4.1 TE: Verification

Goal: TokenSPICE with EVM-in-the-loop

Motivations:

- Model Ocean Market V3 with high fidelity, avoiding error-prone translations
- Model Ocean Market V4.1 with high fidelity, trying out many variants quickly
- (Eventually) Set ourselves up to do what-if scenarios on live running contracts
- (Bonus) hardening of Ocean Market V4.1 smart contracts



Review: SW Simulators

1. **Spreadsheet-based agent-based system.** Each row is a different time step. Some columns are state variables; some are input variables; some are output variables. The next row's state variable values are a function of that row's input variables and the previous row's state variables. Output variables are a function of the current row's state variables and input variables.
2. **Custom software for agent-based modeling, with rough-grained models.** The rough grained models may be subroutines, differential equations or other “behavioral models”. This approach takes more up-front effort than (1), but offers more flexibility. Once that up-front effort is invested, it's also easier to maintain and test, towards building more complex models.
3. **Custom software for agent-based modeling, with fine-grained “smart contracts in the loop”.** This is even higher resolution than (2). Simulation time is longer but it starts to go with shades of gray into real-world behavior. It's akin to hardware-in-the-loop simulation.

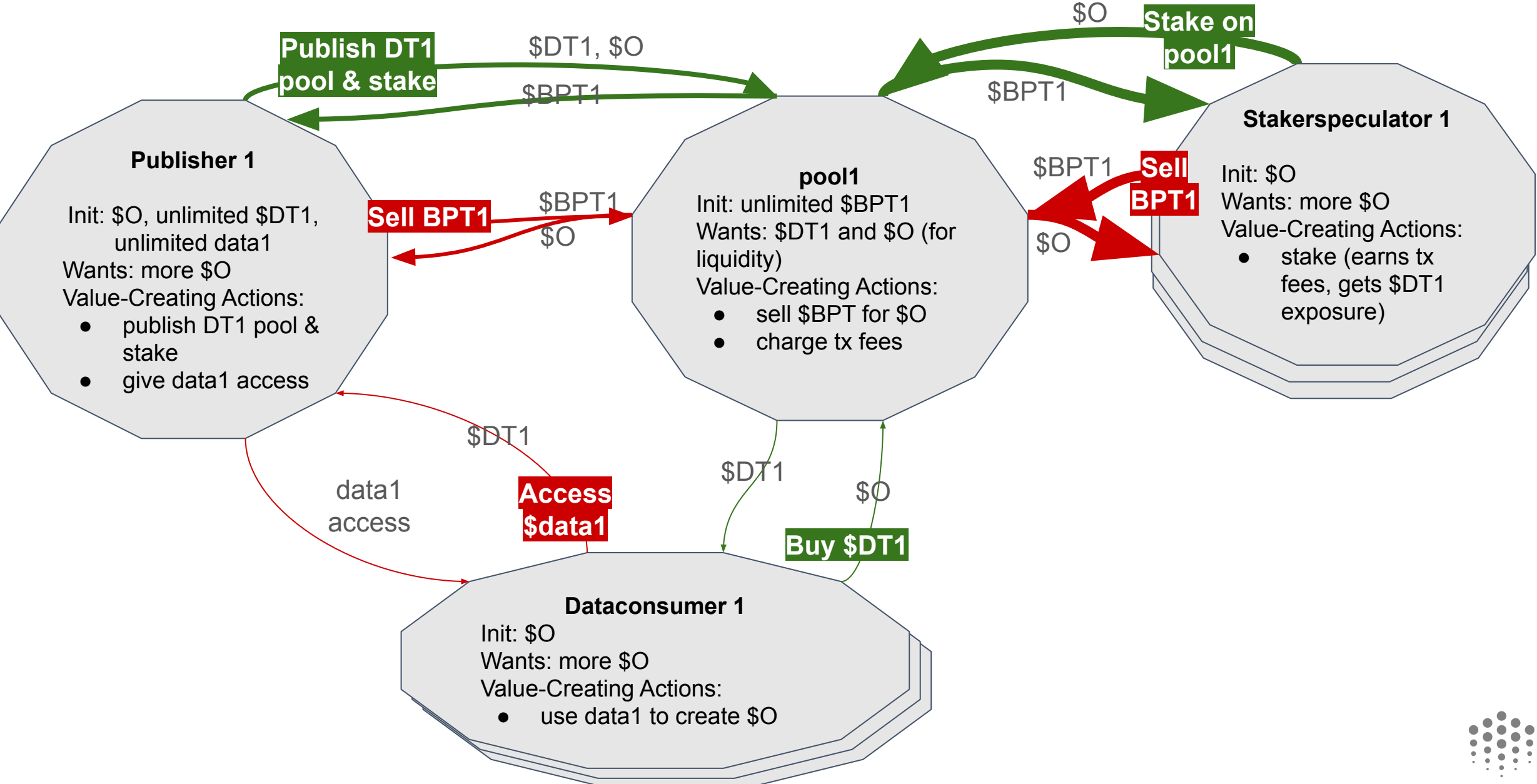
- Google Sheets
- Excel

- **TokenSPICE**
- cadCAD

- **TokenSPICE 2**
- cadCAD future?
- Gauntlet? (proprietary)

Block diagram: Ocean Market V3

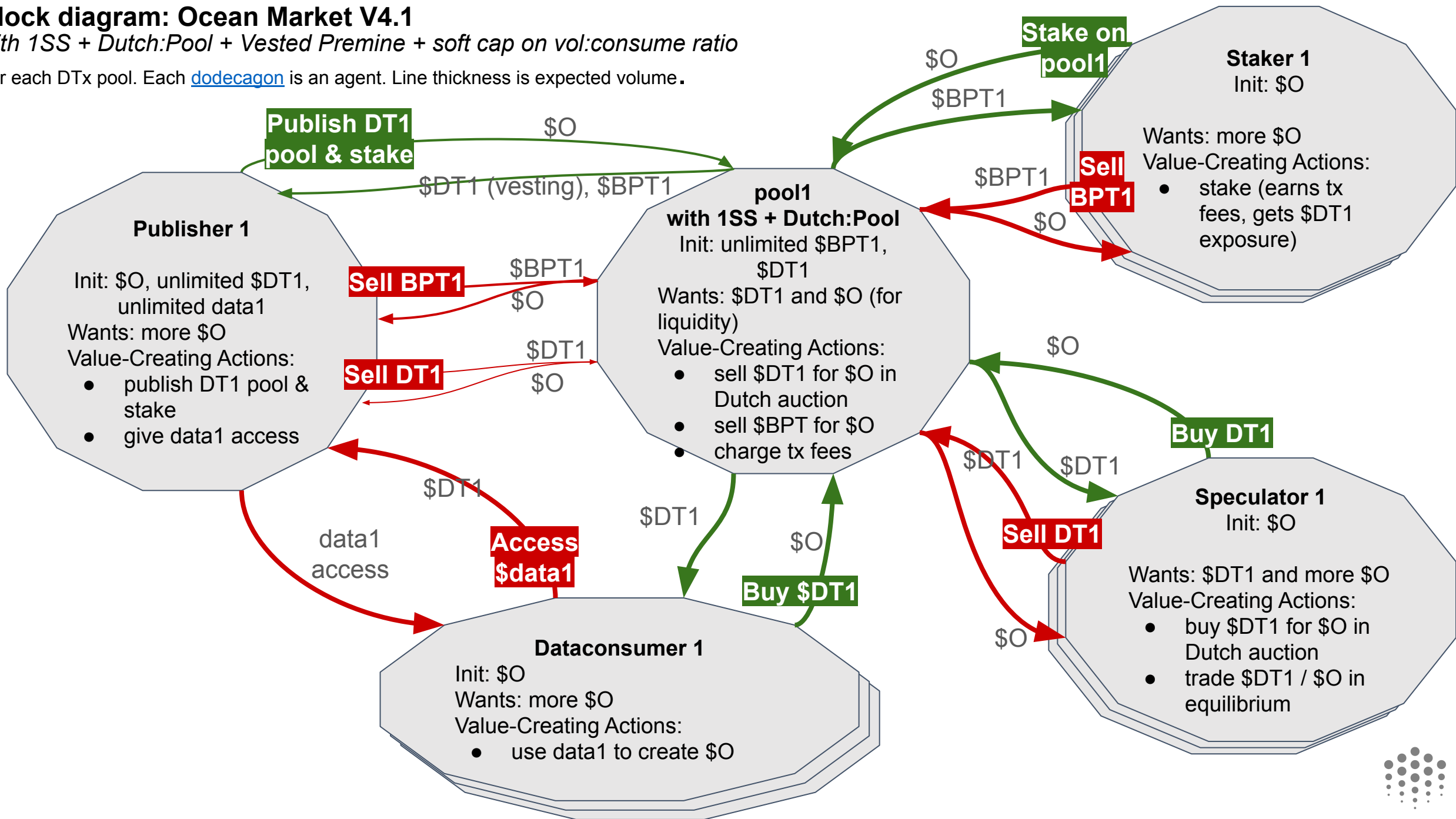
For each DTx pool. Each [dodecagon](#) is an agent. Line thickness is observed volume.



Block diagram: Ocean Market V4.1

with 1SS + Dutch:Pool + Vested Premine + soft cap on vol:consume ratio

For each DTx pool. Each [dodecagon](#) is an agent. Line thickness is expected volume.



TokenSPICE with EVM

Done so far:

- Wrote Ocean Market V4.1 smart contracts
- Drew schematics for V3 & V4.1
- Adapted TokenSPICE code
 - Run EVM end-to-end via ganache
 - Lets third-parties deploy to ganache, then uses at their ABIs
 - ABIs are wrapped as classes, which are inside agents.
 - Already include: Ocean datatokens, Ocean datatoken factory, Ocean friendly fork of **Balancer AMM**, Balancer AMM factory, etc. Have Unit tests for all.
 - Started writing Python-level agent behaviors

Still to do:

- Finish writing Python-level agent behaviors
- Replicate Ocean Market V3 dynamics: run simulations and tune as needed
- Observe Ocean Market V4.1 dynamics: point at Ocean V4.1 contracts and run!
- Iterate on sim and on design until satisfied
- ***THIS IS A LOT!***



TokenSPICE with EVM

Top-level agent architecture

- All agents inherit BaseAgent
- Controllable agents use EVM.
- Uncontrollable agents use pure Python. But each has EOA.
 - Therefore the core dynamics are still on-chain

Controllables

Controllable agents (structure):

- What agents: just Pool (incl. Strategies and Pool Controllers).
- The agent's state is stored on blockchain. Deployment is not in the scope of TokenSPICE right now. TokenSPICE just sees ABIs.
- PoolAgent.py wraps BPool.sol. Agent's wallet grabs values from BPool.sol
 - current design (.sol) is at oceanprotocol/contracts
 - new design (.sol) is at branch 'feature/1mm-prototype_alex'
 - how can PoolAgent see it? draw on btoken.py etc.

Controllable variables:

- Global design vars. E.g. schedule for token distribution.
- Design vars within controllable agents

TokenSPICE with EVM

Uncontrollables

Uncontrollable Agents:

- Uncontrollable agents use pure Python. But each has an Externally Owned Address (EOA) to interact w EVM. Implemented inside Wallet.
- What agents:
 - Status quo design: Publisher, Dataconsumer, Stakerspeculator
 - New design 1: Publisher, Dataconsumer, Staker, Speculator

Uncontrollable Variables (Env & rnd structure & params)

- Global rndvars & envvars.
- Rndvars and envvars within controllable agents
- Rndvars and envvars within uncontrollable agents
- Ranges for envvars, and parameters for rndvar pdfs, are in constants.py, etc.

Metrics

- These are what's output by SimEngine.py into the CSV, then plotted
- In the future, we could get fancier by leveraging TheGraph.



TokenSPICE with EVM

- Each Agent has an AgentWallet.
- Now, **AgentWallet is the main bridge between higher-level Python and EVM.**
- Each AgentWallet holds a Web3Wallet.
- The Web3Wallet holds a private key and creates TXs.

```
class AgentWallet:
    """An AgentWallet holds balances of USD, OCEAN, and DTs for a given Agent.
    It also serves as a thin-layer conversion interface between
    -the top-level system which operates in floats
    -the EVM system which operates in base18-value ints

    USD is stored as a variable internally. OCEAN & DTs are on EVM.
    """

    def __init__(self, USD:float=0.0, OCEAN:float=0.0):
        self._web3wallet = web3wallet.randomWeb3Wallet()

        #Give the new wallet ETH to pay gas fees (but don't track otherwise)
        self._web3wallet.fundFromAbove(toBase18(0.01)) #magic number

        #USD
        self._USD = USD #lump in ETH too

        #OCEAN
        globaltokens.mintOCEAN(address=self._web3wallet.address,
                               value_base=toBase18(OCEAN))
        self._cached_OCEAN_base = None #for speed
```

```
@property
def _address(self):
    return self._web3wallet.address

#=====
def BPT(self, pool:bpool.BPool) -> float:
    BPT_base = pool.balanceOf_base(self._address)
    return fromBase18(BPT_base)
```



Using TokenSPICE 1/4 [added 20210616]

<https://github.com/oceanprotocol/tokenspice>



TokenSPICE: EVM Agent-Based Token Simulator

TokenSPICE can be used to help design, tune, and verify tokenized ecosystems in an overall Token Engineering (TE) flow.

TokenSPICE simulates tokenized ecosystems using an agent-based approach.

Each "agent" is a class. Has a wallet, and does work to earn \$. One models the system by wiring up agents, and tracking metrics (kpis). Agents may be written in pure Python, or with an EVM-based backend. (The [original version](#) was pure Python. This repo supersedes the original.)

It's currently tuned to model [Ocean Market](#). The original version was tuned for the [Web3 Sustainability Loop](#). However you can rewire the "netlist" of "agents" to simulate whatever you like. Simply fork it and get going.

TokenSPICE was meant to be simple. It definitely makes no claims on "best" for anything. Maybe you'll find it useful.

[Documentation](#).



Using TokenSPICE 2/4

<https://github.com/oceanprotocol/tokenspice>

Contents

- 📖 Initial Setup
- 🔄 Updating Env
- 🧪 Do Simulations, Make Changes
- 🏗️ TokenSPICE Design
- 📅 Backlog
- 🌐 Benefits of EVM Agent Simulation
- 🏠 License

🏠 Initial Setup

Set up environment

Open a new terminal and:

```
#ensure brownie's *not* installed. It causes problems  
pip uninstall eth-brownie
```



Using TokenSPICE 3/4

<https://github.com/oceanprotocol/tokenspice>

Get Ganache running

Open a new terminal and:

```
cd tokenspice

#active env't
conda activate tokenspiceenv

#run ganache
./ganache.py
```

Note: you could run ganache directly, but then you have to add many special arguments. The script above does that for you.

Deploy the smart contracts to ganache

Open a separate terminal.

```
#Grab the contracts code from main, *OR* (see below)
git clone https://github.com/oceanprotocol/contracts

#OR grab from a branch. Here's Alex's V4.1 prototype branch
git clone --branch feature/1mm-prototype_alex https://github.com/oceanprotocol/contracts
```

Then, deploy. In that same terminal:



Using TokenSPICE 4/4: Kanban

<https://github.com/oceanprotocol/tokenspice>

oceanprotocol / tokenspice

Watch 10 Unstar 23 Fork 12

Code Issues 8 Pull requests Actions Projects 1 Security Insights Settings

MainProject
Updated 11 minutes ago

Filter cards + Add cards Fullscreen Menu

- 2 Backlog - longer term**
 - Implement Ocean V3 market dynamics, run simulations, tune as needed
0 of 3
#28 opened by trentmc
Type: Enhancement
 - Implement Ocean V4 market dynamics, run simulations, tune until satisfied
0 of 2
#29 opened by trentmc
Type: Enhancement
- 5 Backlog - near term**
 - Add Continuous Integration
#25 opened by trentmc
Status: Refined Type: Enhancement
 - Make code coverage more visible: put at top of README
#26 opened by trentmc
Type: Enhancement
 - Only allow code merges if unit test code coverage ratchets up
#27 opened by trentmc
Type: Enhancement
 - Be able to specify a netlist and run, without having to fork
#30 opened by trentmc
Type: Enhancement
 - Finish/fix DataConsumerAgent_test
#19 opened by trentmc
- 1 In progress**
 - Get *some* overall loop running that includes at least one EVM agent
#34 opened by trentmc
- 5 Done**
 - Rename n
#38 opened
 - Have non-pretty in
#35 opened
 - 1 linked pull req
 - plot_1.py uses Days
#36 opened
Type: Bug
 - Fix web3 getTransa
0 of 3
#32 opened
Type: Enh
 - 1 linked pull req
 - Automated as Dc

Benefits of EVM agent simulation [added 20210616]

TokenSPICE 2 and other EVM agent-based simulators have these benefits:

- **Faster and less error prone, because the model = the Solidity code.** Don't have to port any existing Solidity code into Python, just wrap it. Don't have to write lower-fidelity equations..
- Enables **rapid iterations of writing Solidity code -> simulating -> changing Solidity code -> simulating.** At both the parameter level and the structural level.
- Can quickly integrate Balancer V2 code. Then extend to model other AMMs. And other DeFi code. Etc etc.
- **Plays well with other pure Python agents.** Each agent can wrap Solidity, or be pure Python.
- Super **high fidelity simulations**, since it uses the actual code itself. Enables modeling of uncontrollable variables, both random (probabilistic) ones and worst-case ones.
- Can build **higher-level CAD tools**, that have TokenSPICE 2 in the loop:
 - 3-sigma verification - verification of random variables, including Monte Carlo analysis
 - Worst-case analysis - verification across worst-case conditions
 - Corner extraction - finding representative "corners" -- a small handful of points in uncontrollable variable space to simulate against for rapid design-space exploration
 - Local optimization - wiggle controllable params to optimize for objectives & constraints
 - Global optimization - "", with affordances to not get stuck
 - Synthesis - "" but wiggle code structure itself in addition to parameters
 - Variation-aware synthesis - all of the above at once. This isn't easy! But it's possible. Example: use MOJITO (<http://trent.st/mojito/>), but use TokenSPICE 2 (not SPICE) and Solidity building blocks (not circuit ones)
- **Mental model is general** enough to extend to Vyper, LLL, and direct EVM bytecode. Can extend to non-EVM blockchain, and multi-chain scenarios. Can extend to work with hierarchical building blocks.
- **Can also do real-time analysis / optimization / etc against live chains:** grab the latest chain's snapshot into ganache, run a local analysis / optimization etc for a few seconds or minutes, then do transaction(s) on the live chain. This can lead to trading systems, failure monitoring, more.

Research Qs

Summary: Research Q's

Basis:

- Ocean Market uses Balancer AMMs
- Doing TE to model, verify and optimize Ocean Market is highly useful on its own
- And it's well-defined subset of broader Balancer ecosystem; we can extend scope once we've got a handle on Ocean Market dynamics

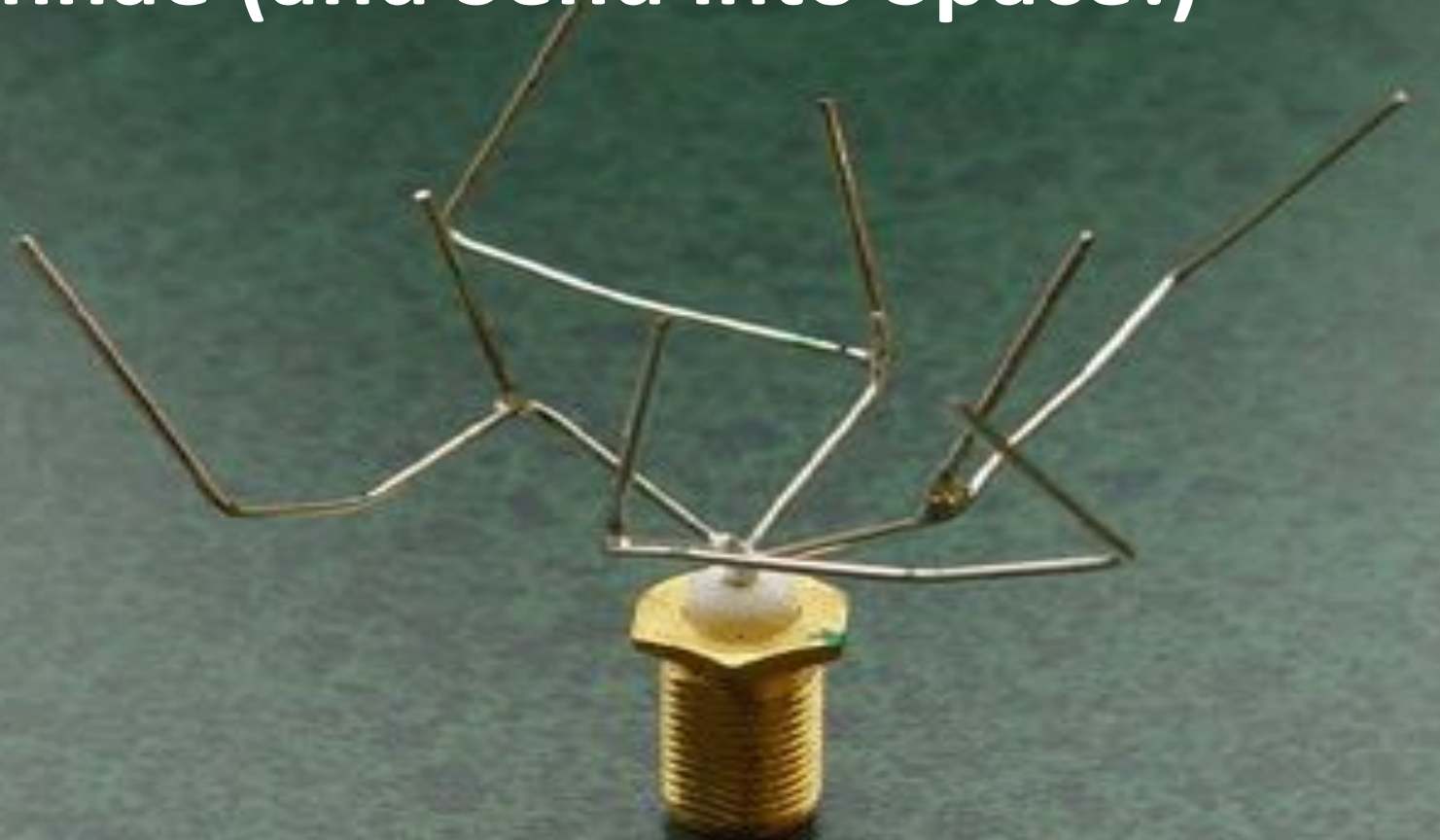
Research Q's

- Can we capture the dynamics of Ocean Market / data ecosystem for Ocean V3? (system identification problem). Includes capturing observed issues .
- Ocean V4.1 has new mechanisms, aiming to address the observed issues. How well do those mechanisms work?
 - One-sided staking bot?
 - Straight-up AMM vs Dutch auction vs LBP?
- Tool: use TokenSPICE with EVM Agent Simulation
<https://github.com/oceanprotocol/tokenspice>

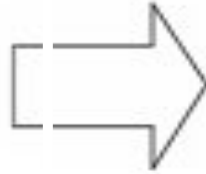
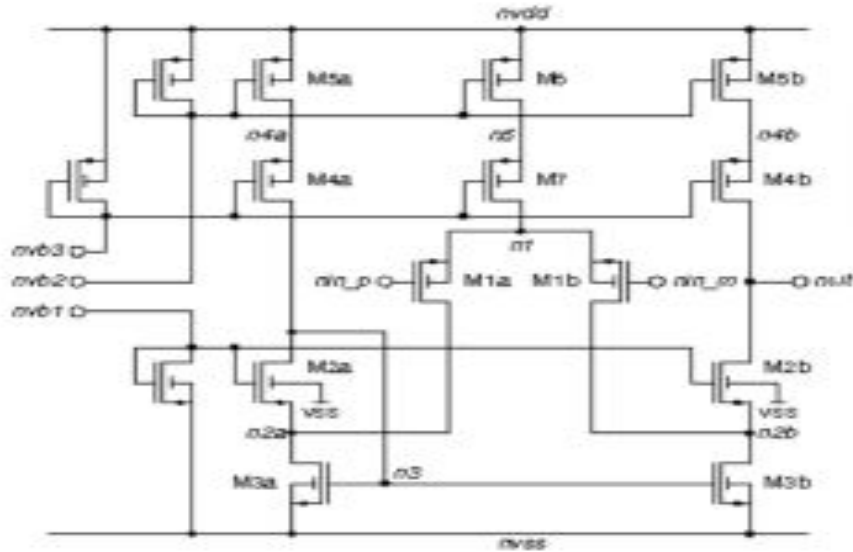


Evolution * TokenSPICE

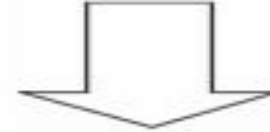
Evolve Antennae (and Send into Space!)



Evolve Whitebox Functions of Circuits



SPICE

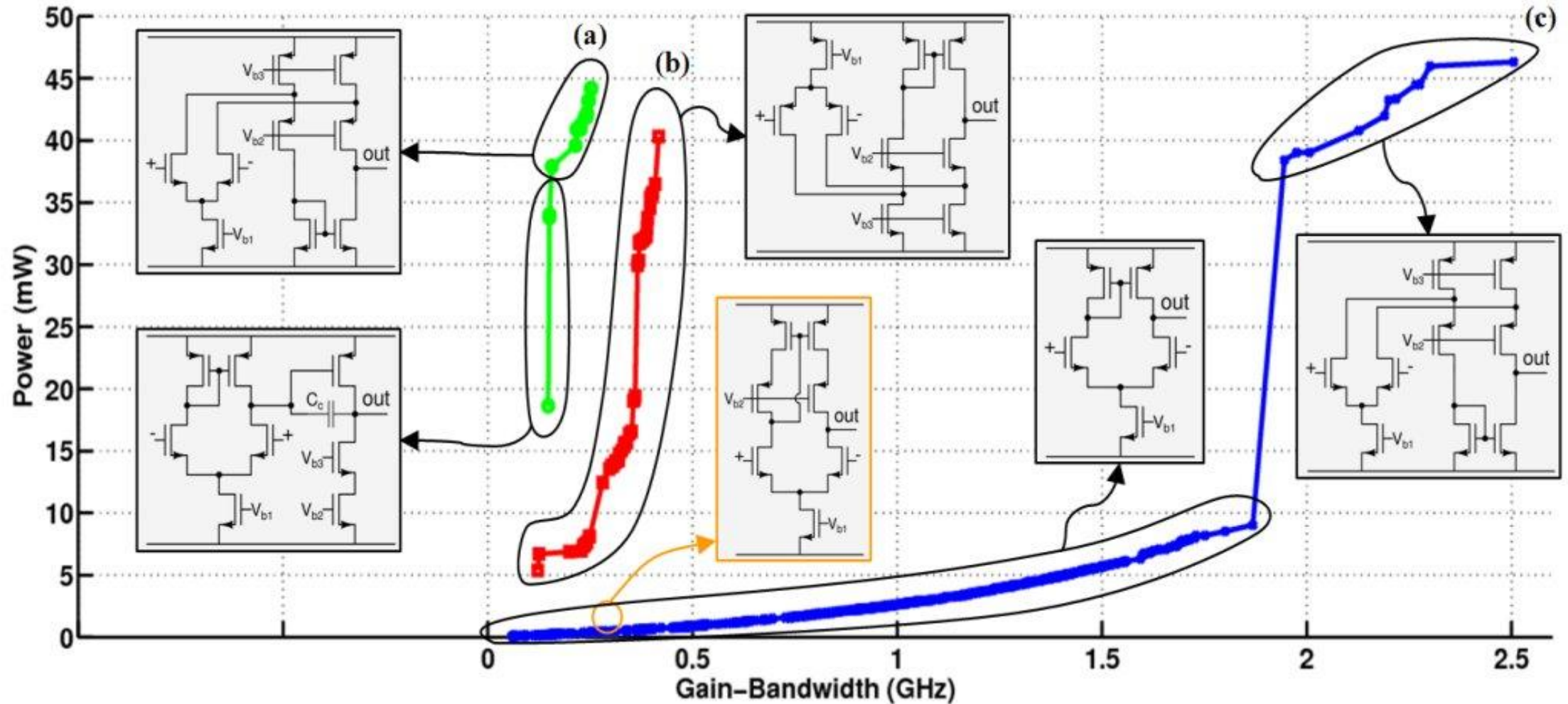


CAFFEINE



Perf.	Expression
A_{LF}	$-10.3 + 7.08e-5 / id1 + 1.87 * \ln(-1.95e+9 + 1.00e+10 / (vsg1*vsg3) + 1.42e+9 *(vds2*vds5) / (vsg1*vgs2*vsg5*id2))$
f_u	$10^{(5.68 - 0.03 * vsg1 / vds2 - 55.43 * id1 + 5.63e-6 / id1)}$
PM	$90.5 + 190.6 * id1 / vsg1 + 22.2 * id2 / vds2$
V_{offset}	$-2.00e-3$
SR_p	$2.36e+7 + 1.95e+4 * id2 / id1 - 104.69 / id2 + 2.15e+9 * id2 + 4.63e+8 * id1$
ΔK_n	$-3.72e+7 - 2.50e+11 * (id1-id2) / vgs2 + 5.53e+6 * vds2 / vgs2 + 109.72 / id1$

Evolve Analog Circuits



Extend TokenSPICE: Evolve Solidity

- Evolve Solidity smart contracts for whatever use case you want.
- Fitness function: Eth testnet running many agents
- Design space: a grammar with many Solidity code blocks

```
/// @title Voting with delegation.
contract Ballot
{
    // This declares a new complex type which will
    // be used for variables later.
    // It will represent a single voter.
    struct Voter
    {
        uint weight; // weight is accumulated by delegation
        bool voted; // if true, that person already voted
        address delegate; // person delegated to
        uint vote; // index of the voted proposal
    }
    // This is a type for a single proposal.
    struct Proposal
    {
        bytes32 name; // short name (up to 32 bytes)
        uint voteCount; // number of accumulated votes
    }

    address public chairperson;
    // This declares a state variable that
    // stores a `Voter` struct for each possible address.
    mapping(address => Voter) public voters;
    // A dynamically-sized array of `Proposal` structs.
    Proposal[] public proposals;

    /// Create a new ballot to choose one of `proposalNames`.
    function Ballot(bytes32[] proposalNames)
    {
        chairperson = msg.sender;
        voters[chairperson].weight = 1;
        // For each of the provided proposal names,
        // create a new proposal object and add it
        // to the end of the array.
        for (uint i = 0; i < proposalNames.length; i++)
            // `Proposal({...})` creates a temporary
            // Proposal object and `proposal.push(...)`
```



Extend TokenSPICE: Evolve EVM Bytecode

- Ethereum Virtual Machine has about 100 operators
- GP could evolve these directly

Runtime Bytecode	60606040525b600080fd00a165627a7a7230582012c9bd00152fa1c4
Opcodes	PUSH1 0x60 PUSH1 0x40 MSTORE PUSH1 0x18 PUSH1 0x0 SSTORE
Assembly	<pre>.code PUSH 60 contract MyContract {\n PUSH 40 contract MyContract {\n MSTORE contract MyContract {\n PUSH 18 (10 + 2) * 2 PUSH 0 uint i = (10 + 2) * 2 SSTORE uint i = (10 + 2) * 2 CALLVALUE contract MyContract {\n ISZERO contract MyContract {\n PUSH [tag] 1 contract MyContract {\n JUMPI contract MyContract {\n PUSH 0 contract MyContract {\n DUP1 contract MyContract {\n REVERT contract MyContract {\n</pre>



Evolve WASM Bytecode

- WebAssembly (WASM) is the future of smart contract VMs. Smart contracts in C, C++, Rust, ..
- WASM is already supported by major browsers

C input source	Linear assembly bytecode (intermediate representation)	Wasm binary encoding (hexadecimal bytes)
<pre>int factorial(int n) { if (n == 0) return 1; else return n * factorial(n-1); }</pre>	<pre>get_local 0 i64.eqz if (result i64) i64.const 1 else get_local 0 get_local 0 i64.const 1 i64.sub call 0 i64.mul end</pre>	<pre>20 00 50 04 7E 42 01 05 20 00 20 00 42 01 7D 10 00 7E 0B</pre>



Evolve Networks of Attackers & Defenders

- To improve security
- Ref. Una-May O'Reilly research @MIT

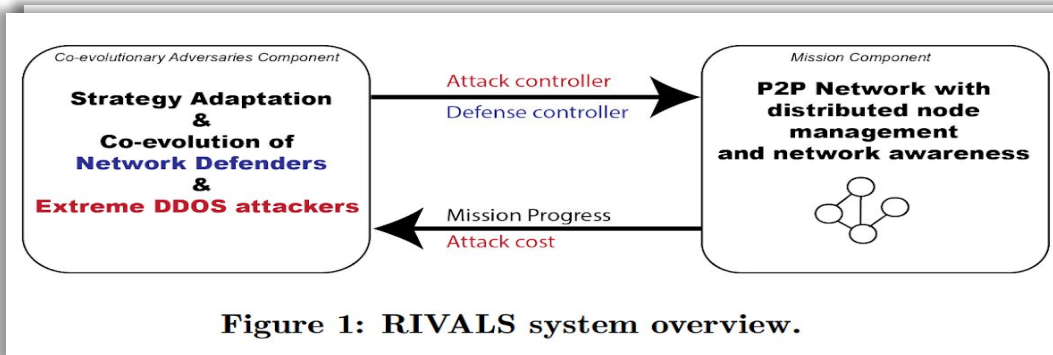


Figure 1: RIVALS system overview.

Developing proactive defenses for computer networks with coevolutionary genetic algorithms

Conference Paper · July 2017 with 12 Reads

DOI: 10.1145/3067695.3089234

Conference: Conference: the Genetic and Evolutionary Computation Conference Companion

[Cite this publication](#)



Anthony Erb Lugo



Dennis Garcia



Erik Hemberg

15.66 · Massachusetts Institute of Technology



Una-May O'Reilly

19.56 · Massachusetts Institute of Technology



ocean

Roadmap

Roadmap - this work

Via <https://github.com/oceanprotocol/tokenspice>

1. System identification: high-fidelity model of Ocean V3 (w/ Balancer V1); fit the model to observed on-chain dynamics
2. Verification: high-fidelity model of Ocean V4.1 (w/ Balancer V2) base design, and the efficacy of each proposed mechanism.
3. Design space exploration: tuning of Ocean V4.1 (w/ Balancer V2 design. Manual or optimization-based.



Roadmap - beyond this work

Via <https://github.com/oceanprotocol/tokenspice>

1. System identification: high-fidelity model of whole Balancer V1 ecosystem; fit the model to observed on-chain dynamics (up to when V2 released). Bring in uncontrollable variables (probabilistic & worst-case).
2. System identification: high-fidelity model of whole Balancer V1 & V2 ecosystem; fit the model to observed on-chain dynamics
3. Design space exploration: tuning of Balancer V2 Strategies to minimize IL and other objectives & constraints. Account for uncontrollable variables (probabilistic & worst-case).
4. Open-ended design space exploration: evolve solidity or EVM bytecode, go nuts. AI DAOs that own themselves. Fastest path = use <http://trent.st/mojito>, hook in TokenSPICE, add Solidity building blocks. This will be fun:). But one step at a time.

Conclusion

Outline

- Summary: Research Q's
- Intro
 - What's Ocean?
 - EE Simulation & Verification
 - TE Simulation & Verification
- Ocean System TE
 - Base
 - SW Verification w TokenSPICE
- Ocean Market TE
 - V3 base
 - V4.1 base
 - V4.1 SW Verification w TokenSPICE & EVM
- Roadmap - this research & beyond
- Conclusion

