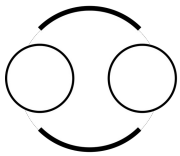
| | | oo.computer | | |

An explanation

Lee Thomas
lee@oo.computer



Rank ideas and make them real

Table of Contents

Version	Date	Description
0.0	6th May 2020	Initial draft (ooda.space)
1.0	7th July 2020	Update with ERC721 mention, compression, for SB.
1.1	18th July 2020	Expanded, for MTR
1.2	4th August 2020	Updates post site launch
1.3	11th December 2020	Re-orientation around hyper*
1.4	6th February 2021	Signal set added to appendix. Core principles included.
1.5	30th July 2021	Change to oo.computer brand
1.6	4th August 2021	Various minor updates to reflect new architecture
2.0	12th August 2021	Various minor updates, pre oo.computer going live

2.1	30th September 2021	Update to implementation to describe use of hyperbees in place of hyperdrives
2.2	24th November 2021	Typographical error corrections
2.3	27th November 2021	More options for profit-share mechanism added
2.4	14th January 2022	Profit Share Mechanism updated, introduction of PETs, new figures

Overview

The oo.computer (pronounced as in “boom” and “zoom”) is a transparent and accountable system to contextualise and rank ideas with others. It aims to merge the roles of investment, capital allocation and public value signalling by offering those who signal value in ideas and contexts a share of any profits that arise from them. It does so in a way that is censorship resistant, yet intuitively filterable. The overarching aim is to accelerate the revelation and realisation of valuable ideas.

Its operation is summarised thus:

1. New ideas and contexts are initiated.
2. Value, in ideas and contexts, is signalled through public commitments to pay the idea and context initiators.
3. The value signals are collated and used to rank ideas, in contexts. The rankings are filtered through selection and weighting of value signallers.
4. Independently verifiable profit share claim options are transparently accounted for.

Its features include:

- That every new idea creates a potential context in which to rank other ideas. A tree-like structure of linked contexts is formed and used to visualise ideas in a sequence of ranked lists. This is called the idea-value-tree.
- The ability to claim ownership of a new idea or context, and to transfer those claims to others. This offers the potential for a market for ideas and contexts using trade of verifiable idea ownership claims.
- A transparent and accountable code base. The system itself is built from ideas in the form of modular code components that are ranked and linked on the system. The user can see the paths of accountability associated with every code module.
- A highly scalable local-first architecture with Distributed Hash Table broadcast (built using the Hypercore protocol). Users create independently verifiable and indexable append only logs. This allows the user to broadcast signals in a way that is minimally censorable yet highly filterable.

Core principles

The following 6 core principles guide the decisions made during the creation and evolution of the system:

1. All **information filtering must be clearly defined, intuitive, and under the user’s control**.
2. Presentation of information must be **context oriented.**
3. **Contributors must share in profits** that arise from the system.
4. There must be **no means of persecution** via the system.
5. The system must be **transparent and verifiable** in its operation.
6. The system must be **easy to start and fun to use.**

The Idea Value Tree

An imaginary tree-like arrangement of ideas and contexts is central to the operation of the oo.computer. An outline representation of the idea value tree is shown in Figure 1. Every idea combines with its parent to create a unique context for other ideas to be added. The combination of an idea with its parent and child contexts creates an object called an “oo”. The schema for the idea-value-tree is shown in Appendix I.

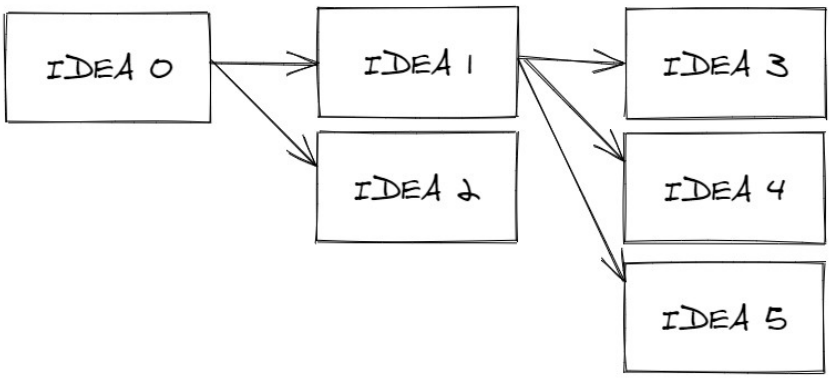


Figure 1: Outline representation of the idea-value-tree

The oo

A digital object called an oo (pronounced as in “moon” and “vroom”) embodies a digitally encoded idea in a context as well as a prospective context for other ideas. An oo has three components, each 256 bit words: A *context* (the left o of the oo), a *prospective context* (the right o of the oo) and an encoded *idea* formed by combining the *context* and *prospective context*. The oo.computer represents each oo iconographically, as shown in Figure 2.

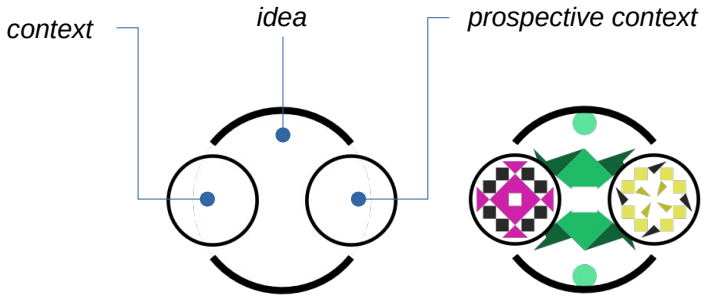


Figure 2: Iconographic representation of an oo. Left: an empty oo. Right: With “fingerprint” icons generated by Jdenticon

Icons are created for each word. By convention, zero (256 binary zeros) is translated to a white icon. Similarly, the maximum, $2^{256}-1$, (256 ones) is translated to a black icon. The remaining possible icons are generated using an icon generation tool, presently jdenticon. It is anticipated that different icon generation tools will be utilised in future as categories of idea encodings are deemed by users to warrant a distinct category of icons.

Chaining ideas and contexts

The oos are chained together in a sequence of contexts. After multiple ideas have been added, a tree like structure emerges. One can pick a route through the tree and represent it as shown in Figure 3. Importantly, each oo has a record of where it is in the chain; an integer representing the number of oos between it and zero. This value is referred to as *distance*, `d`.

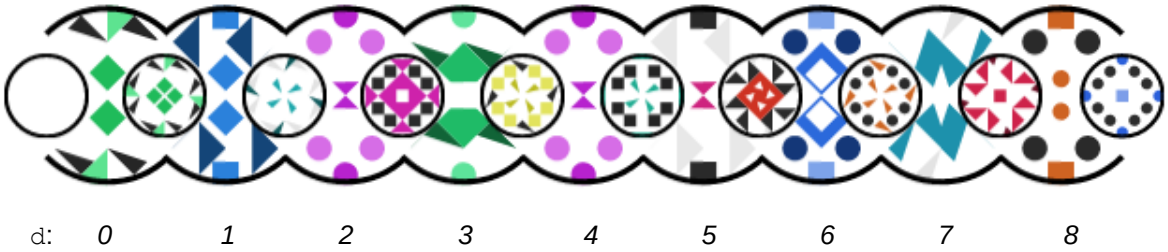


Figure 3: Iconographic representation of a chain of contexts

Relationship between encoded ideas and contexts

The *idea*, *context* and *prospective context* are interrelated as shown below and in Figure 4.

```
idea = XOR(ROTR(context,3*d),ROTL(prospective,3*d))
context = XOR(thing,ROTL(prospective,3*d))
prospective = XOR(thing,ROTL(context,3*d))
```

`XOR`: Exclusive or operation on two 256 bit words
`ROTR(word,n)`: a rightward bit rotate operation of n bits
`ROTL(word,n)`: a leftward bit rotate operation of n bits

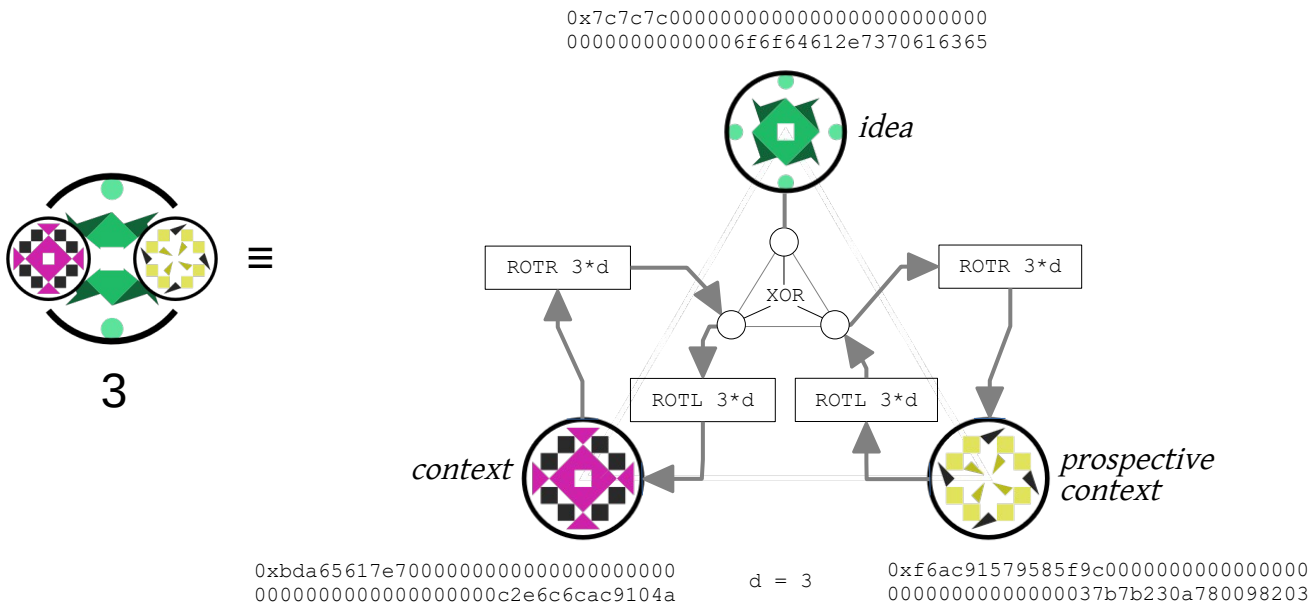


Figure 4: Diagrammatic illustration of the interrelationship between context, idea and prospective context

Ideas, by default, are contained in a box with a 2:1 aspect ratio. The user is able to "open" the box via the interface. An example series of ideas (interpreted as single UTF8 encoded words) is shown in Figure 5.

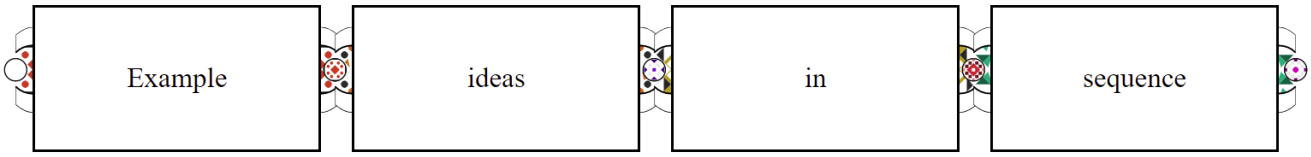


Figure 5: A series of single word ideas in 2:1 aspect ratio boxes

Interpretation of ideas

Ideas can have any number of interpretations. For example, they may represent a single word, a file, an account or a virtual computer. The conventions used by the system to interpret *ideas* are shown in Table 1.

Table 1: Idea encoding and presentation conventions

UTF Prefix	Hex Prefix	Encoding Convention	Presentation Convention
|||	7c7c7c	Interpret the remaining 232 bits as UTF8. This allows 29 character UTF8 units	Text in font according to browser preference and size scaled to fit.
MED	4d4544	Interpret the following 72 bits as the first 72 bits of the SHA1 hash of the media type (e.g. "text/html"). Interpret the final 160 bits as the SHA1 hash of a media file.	Present the content according to the media type.

In this document, a short hand is used to represent sequences of 29 character UTF8 text encoded ideas. Three pipe characters "|||" are used to delimit the ideas. Also, `0b0` and `0b1` are used represent the all zeros and all ones words. An example of a sequence of ideas (from Figure 5) represented in this way is:

```
|||Example|||ideas|||in|||sequence
```

The Signal Set

The oo.computer system operates using a set of pre-defined signals. These relate to the construction of the idea-value-tree, idea ownership claims, transfer of claims, and payments. Anyone can create a signal. To be valid, a signal must be independently verifiable (cryptographically signed) and stored in a publicly accessible place. A "signal chain" is where a user's signals are arranged into an append only log. A summary of the signal types is shown in Table 2, the complete definition is contained in a javascript object (see Appendix II). The version of the signal set is denoted by the SHA256 hash of the UTF8 encoded text defining the javascript object.

Table 2: The oo.computer signal set, summaries

Signal name	Description

INIT	Announcement of public key and intention to use a signal set version in relation to the idea-value-tree.
CLAIM	A claim of ownership of a word (idea or context).
TRANSFER	A declaration of transfer of an ownership claim to another key.
VALUE	A verifiable public commitment to pay the owners of an idea and contexts in an oo an equal amount, where the amount is discretionary but must be declared (unit of account is uBTC).
DISCARD	An announcement that an oo should no longer be considered valuable. Invalidates un-invoiced value signals.
INVOICE	A request for payment related to a value signal for an owned idea.
RECEIPT	A confirmation that a payment has been made with a relation to one or more value signals, with transaction id.
VERSION	A signal of change of signal set version.
HACKED	A signal that the private key relating to this account has been compromised.

Value signals allow the user to declare other information. Conventions for ideas value signalled in certain contexts have been established as follows:

Context	**Convention**
|||*Alts*	Alternative public keys and key algorithms that represent the signaller
|||*Currency*	Currency preference
|||*Filters*	A declaration that a particular word represents a filter context (any oo signalled with this context is considered to represent another value signaller).
|||*Names*	Human readable name(s) to associate with the key
|||*PaymentFacilitators*	Public keys of nominated third party payment facilitators, if undefined defaults to ooda space ltd
|||*Tree*	The public key (e.g. hyperdrive address) of the idea-value-tree derivable from the signal chain.

Value Signals

A value signal is the signal used for construction of the idea-value-tree. Value signals are:

- the way that users publicly signal that an idea, in a context, is valuable.
- the basis for profit share calculations.
- the means by which the ideas that a user sees in a context are filtered and ranked.

A value signal is a commitment to pay the initiators of the idea and contexts in an oo an equal amount, where the amount is discretionary but must be declared. A simplified representation of a value signal (value of 1 unit) is shown in Figure 6. The unit of account for value signals is uBTC. This is used because a consistent base currency is required so that value signals can be weighted without recourse to complex exchange rate calculations. The user is free to select the currency that the payment will be made in, however. The value signaller is free to select a payment facilitator, but must pay the transaction fee. The exchange rate used in the payment must be signalled by the receiver. This allows users to judge the reputations of value signallers and their payment facilitators.

The value signaller also declares who (which public keys) they deem to be the valid owner of the idea and contexts. If undeclared, the amount must be paid to all past value signallers via the oo.computer profit share system. In any case, 10% of the value signal payments must be distributed to all past value signallers via the oo.computer profit share mechanism.

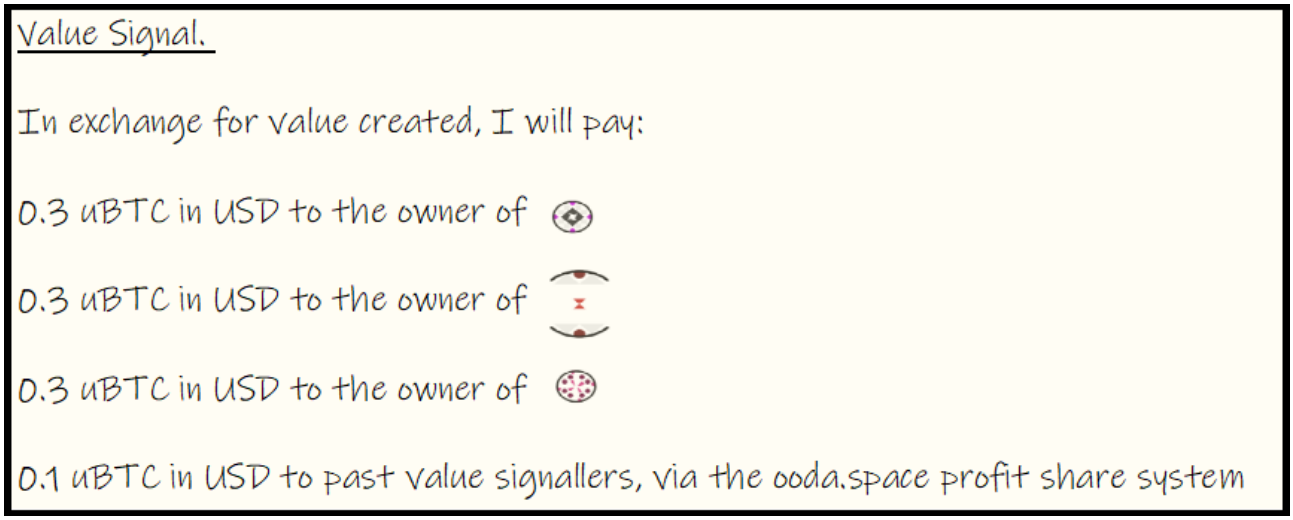


Figure 6: Simplified representation of a value signal

Filters

Filters are what users use to tell their computer which part of the idea-value-tree to retrieve, and in what rank order to display the ideas in each context. To create a filter, the user first creates a filter context by value signalling a filter name in the "Filters" context. This declares that a particular context in the idea-value-tree will be used to signal value in value signallers (called filterers in this context). The computer weights the value signals of each filterer accordingly and presents the idea-value-tree to the user based on these signals. Alternatively, the user can select other value signaller's filters (the term "filter-directors" is used to describe value signallers that create filters). Either way, the user chooses who chooses what they see, in a transparent process.

To denote their status, value signallers are given a rhombus shaped icon. The display of filters to the user is shown in Figure 7. Each filter is given a different colour and name. The presentation can be imagined as bar-graphs, with the height of each bar representing the normalised weighting of a filterer (in relation to the filterer with the highest value signals received).

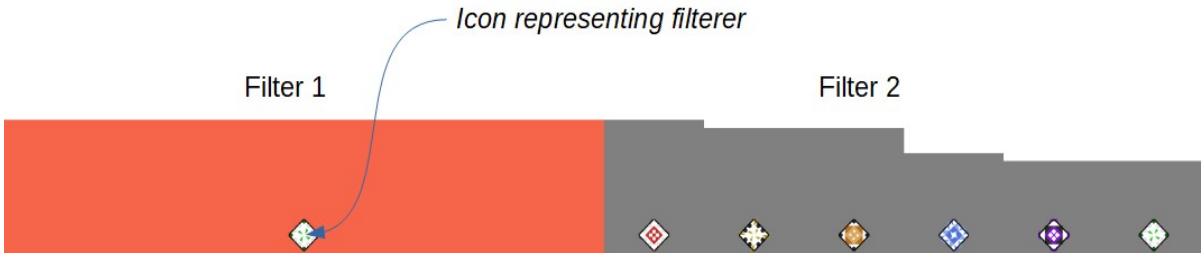


Figure 7: How filters are displayed to the user

Ranking ideas using value signals

The oo.computer will display the ideas in a given context in rank order, according to the cumulative value signals from each of the filterers in the user selected filters. If the user changes the weighting of a filterer, the rank ordering will change accordingly. If a new filterer is added, new ideas may be introduced. Through addition of colour to each oo, the user is given an indication of the relative amount of value that has been signalled from each user-selected filter.

Example derivation of ranking from value signals

Take an example filter where there are two filterers with value signals of 5 and 15 uBTC respectively.

In this case, the 1st address has 25% of the filter weight and the 2nd 75%

After this, the user selects a *context* to view. Let's say the *context* is "Transparent News"

Let's also say that the first filterer has value signalled 1000 uBTC to "Title 1" in the "Transparent News" context and that the second filterer has value signalled 500 uBTC to "Title 2" in the "Transparent News" context.

The filter modifies the amounts as follows

Title 1: 1000 * 0.25 = 250
Title 2: 500 * 0.75 = 375

As proportions of the filter, these are:

Title 1: 0.4
Title 2: 0.6

These values are then normalised against the maximum giving weightings of:

Title 1: 0.66667
Title 2: 1.0

These weightings are then used to calculate the rank order of the ideas. Therefore "Title 2" is brought to the front and "Title 1" pushed rearwards in the example "Transparent News" context. The example is summarised in Figure 8.

When there are multiple live filters the amounts from each filter are added together for each idea. A user configurable filter colour scheme is used to fill in the oo's wrapper in proportion to the value signals originating from the different filters. By default, each live filter has an equal overall weighting. The value signals for each filter are normalised with respect to the highest weighted value signals total received, from the filter's filterers, by any idea within a given context.

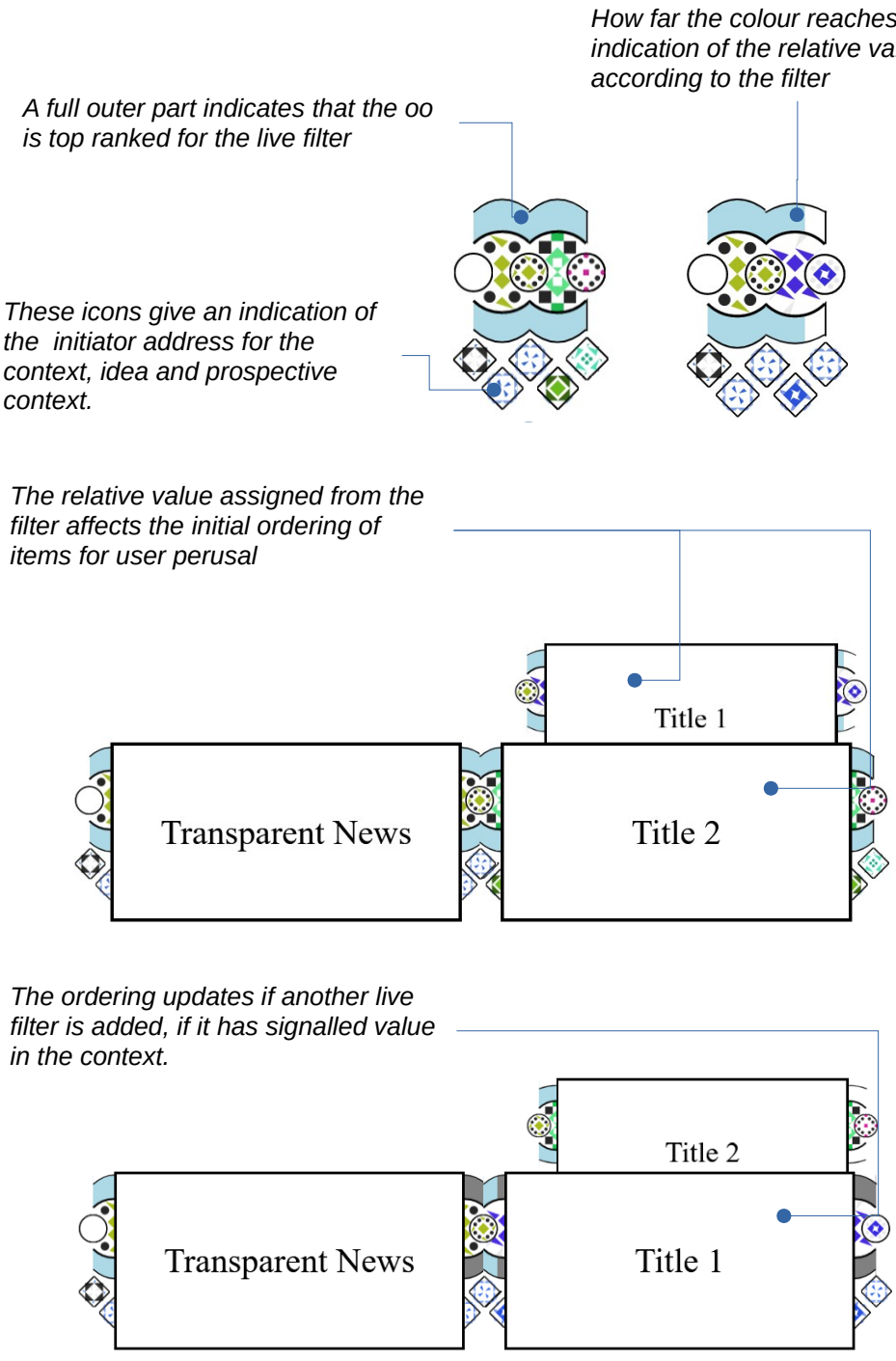


Figure 8: Simple example of the oo.computer ranking process

Profit share mechanism

A token based profit share mechanism has been designed. This will be implemented on a smart contract platform (yet to be selected) or as a new public blockchain. The aim of the profit share mechanism is to distribute the profits arising from the value signals to the past value signallers so that:

- Value signallers have a long-term profit motive for signalling valuable things.
- There is a gradual re-distribution of profit share entitlements from inactive to active users.
- The act of value signalling is a path for investment in the oo.computer itself.
- There is a general incentive to care about the healthy operation of the oo.computer.
- It is clear that no investors were given early access privilege.

The mechanism has a weekly cadence. An amount of centralised (conceptually only) processing occurs every week to apportion the profits and re-distribute profit entitlements for the following week. The processing occurs in a half hour window, referred to as a gate, at the end of each week. The holders of Profit Entitlement Tokens (PETs) are entitled to receive the profit from the system (10% of all value signals) in direct relation to the proportion of the token supply they hold. The token supply S is fixed; initially at 10^{50} though this may change to suit the available integer mathematics on the chosen smart contract platform (to confirm, the supply remains fixed after operation of the profit share mechanism commences).

To ensure that there are no investors with early access privilege, two sub-categories of PETs are used; Full Redistribution (FPETs) and Partial Redistribution (PPETs). At every gate, FPETs are taken from their holders and given to the Value Signallers for the prior week. In contrast, only 0.25% of PPETS are redistributed in this way. At initialisation, the entire token supply is FPETs. At every gate 0.0025 S FPETs are re-classified as PPETs. Therefore at week 400, all the PETs will be PPETS. This mode of operation essentially means that performing a value signal payment in week 0 is synonymous with a token crowdsale. But because the first week's entire supply are FPETS, the profit entitlement only lasts for a week. As the weeks go on, however, the earned profit share entitlements becomes longer lasting. The transition of FPETs to PPETS is illustrated in Figure 9.

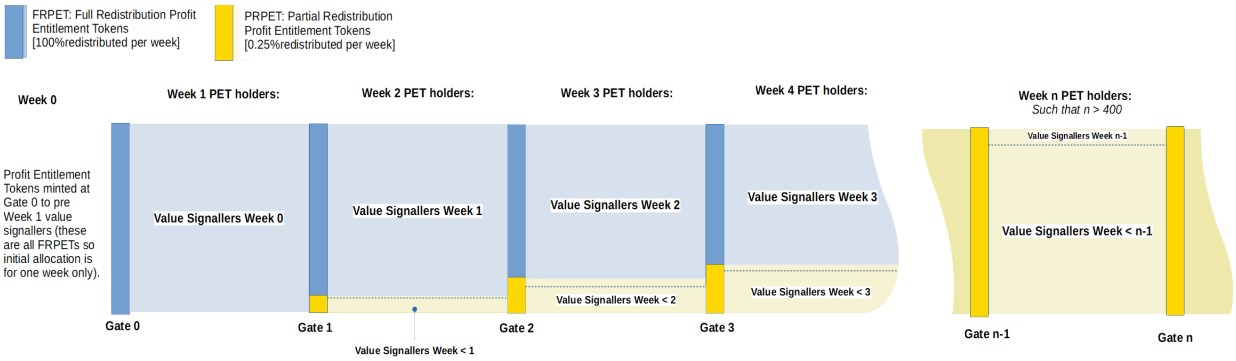


Figure 9: oo.computer Profit Entitlement Tokens; transition from full redistribution (FPETs) to partial redistribution (PPETs)

As an illustration, consider a user holding 0.01% of all PETs in week n (where n > 400, so they are PPETs). This user is entitled to 0.01% of the income from value signals for week n. Assuming the user

does no more value signalling, at week n 0.25% of the user's PPETs are redistributed to others. Therefore, for example, the user is entitled to ~0.0088% of the profit-share income from week n+52. Of course the user could lessen the rate of depletion (or even increase) their profit share entitlement by actively value signalling and making the value signal payments. However, assuming no further value signals are made, the per-unit gradual reduction in profit-share entitlement is shown in Figure 10.

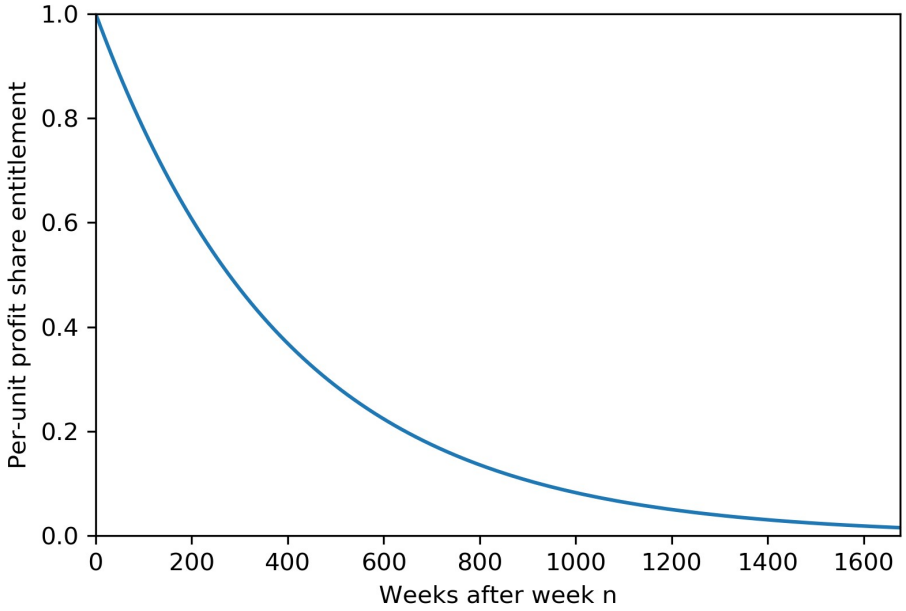


Figure 10: Per unit depletion of profit share entitlement for an inactive user

Note that, dependent on the integer mathematics of the selected smart contract platform, a user's profit share may drop to zero eventually. However, if there is sufficient resolution on the integer range, this should take a long time (~decades).

Figure 11 shows an example of the flow of value signal payments to context and idea initiators and to past value signallers (holders of PETs). To be valid, a value signal must contribute 10% of its amount to past value signallers via the mechanism. The figure considers an example value signal that takes place in week 2. It illustrates the recipients of an example value signal's payment, the PETs allocated to the value signaller in reward, the redistribution of the value signaller's PETs in the following weeks and the share of future value signal payments the example value signaller receives.

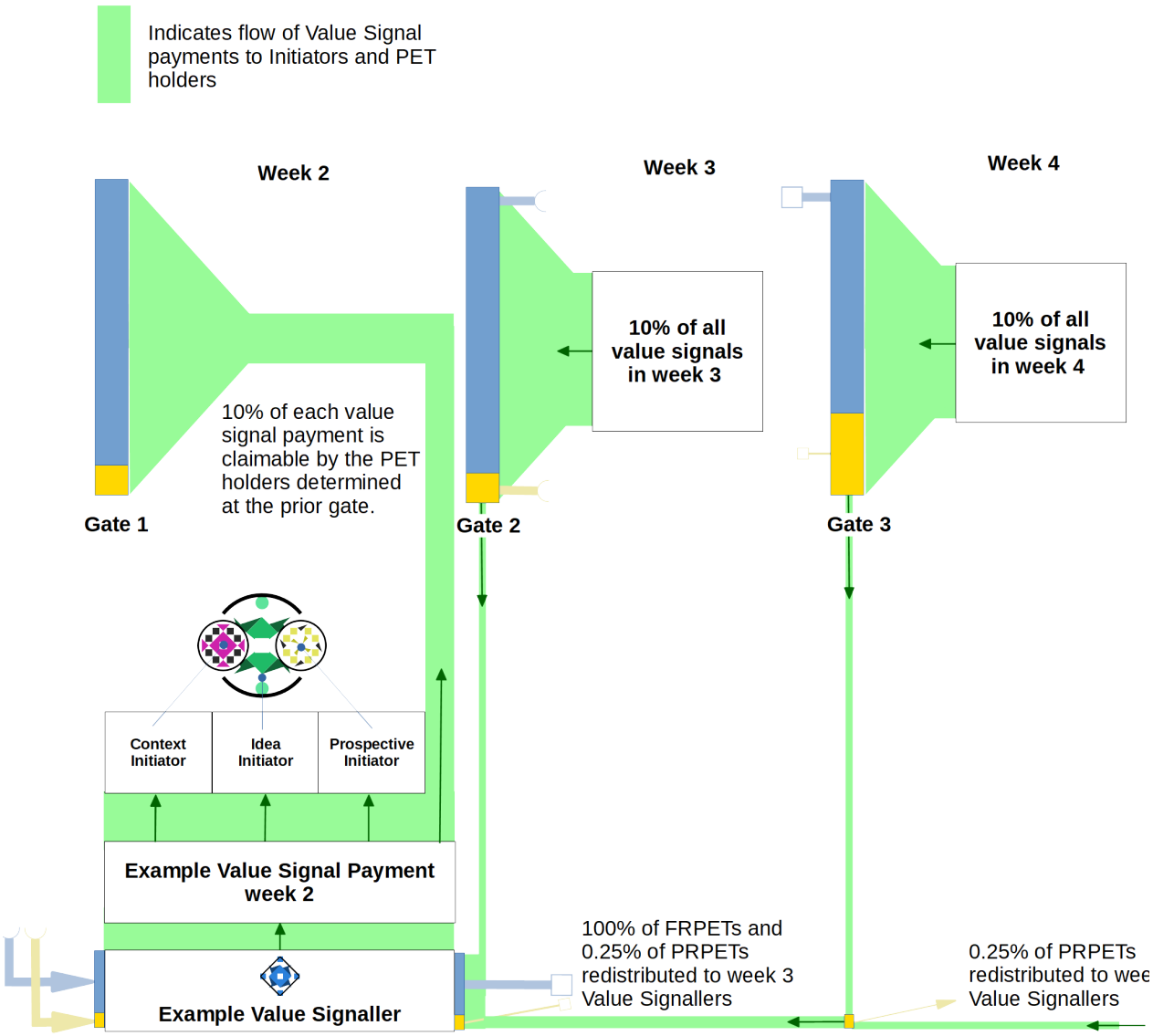


Figure ١١: An illustration of how Value Signals interact with the Profit Share Mechanism. An

Implementation

Construction of the idea-value-tree

The oo.computer is implemented in Javascript as a peer-to-peer web browser (based on Electron) using the hypercore-protocol (a suite of modules and abstractions that allow local-first verifiable construction and automated decentralised dissemination of custom data objects). Users store their signals in a hypercore, a local append-only log (blockchain) that allows for verification of transactions and transaction ordering; this is their "signal chain". From these signals, a more complex data structure is derived; the user's portion of the idea-value-tree (as per the schema in Appendix I).

To construct the idea-value-tree, a hyperbee (a b-tree key-value store where every edit is transactional and verifiable) was used. Every key is a hex-encoded context. Every value is a JSON (JavaScript Object Notation) string of containing the parent context, a list of the child contexts, its distance number, the current rank ordering and weightings for the child contexts, and the recognised ownership claim for the idea (see the schema in Appendix I). The idea-value-tree is derived from the user's signal chain. Others can check the validity of a user's idea-value-tree by re-creating it from their signal chain. The relationship between the signal chain, the idea-value-tree data structure and the user interface is indicated in Figure 12. An overview of the system's architecture is shown in Figure 13.

A separate pair of hyperbees are used to map ideas (256bit hex) to media (e.g. video, text etc) and content-type values. These are used if the idea represents a media object (i.e. where the opening 3 bytes are 0x4d4544, see Table 1).

When the user signals value in another signaller, the signal chain created by the signaller is retrieved using the hyperswarm-dht system. From this, the signals are used to update the user's idea-value-tree according to the schema (Appendix I). Where an idea represents some media, the files and content-type are requested as well.

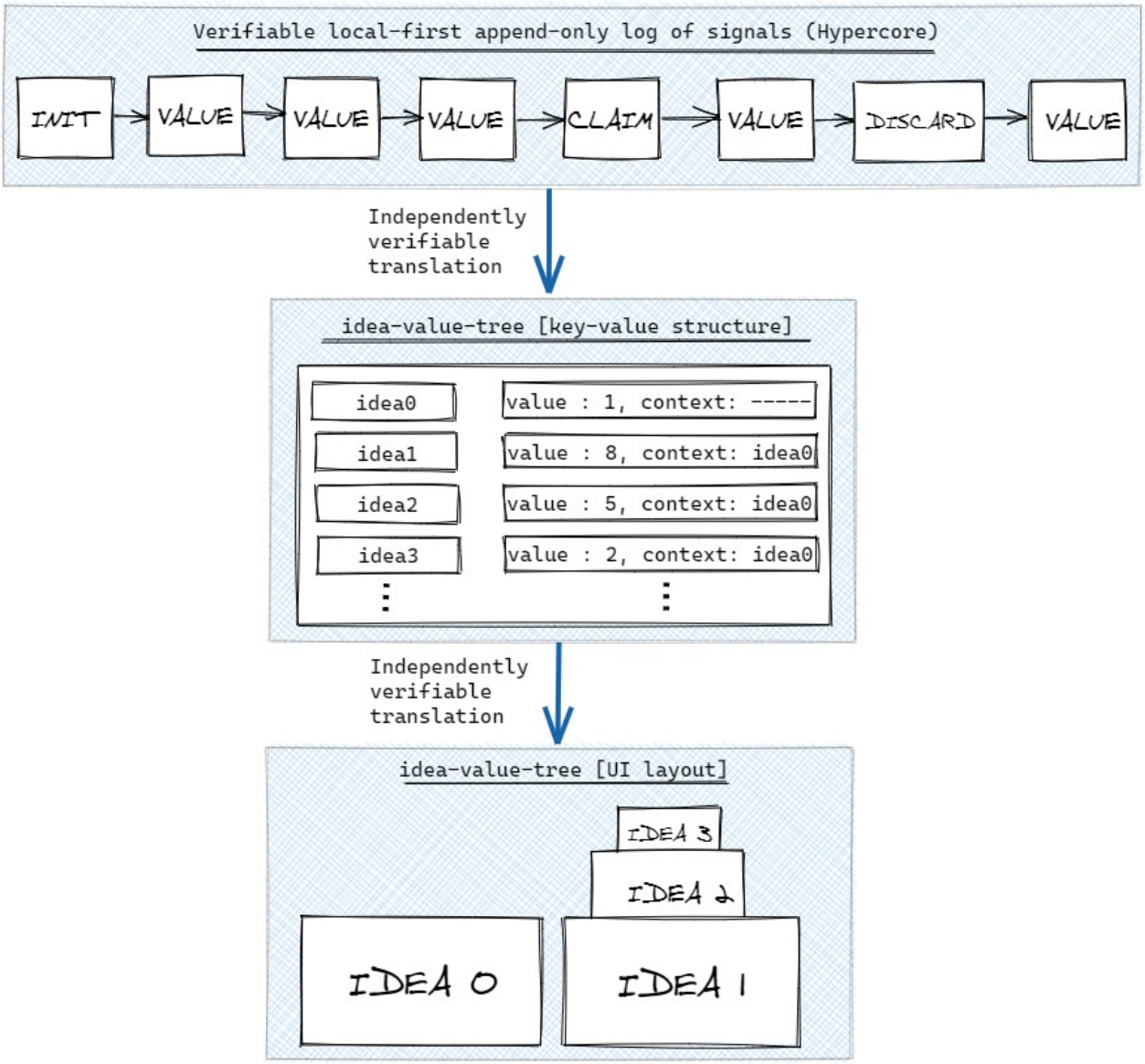


Figure 12: relationship between the signal chain, the idea-value-tree data structure (simplified) and the User Interface (UI)

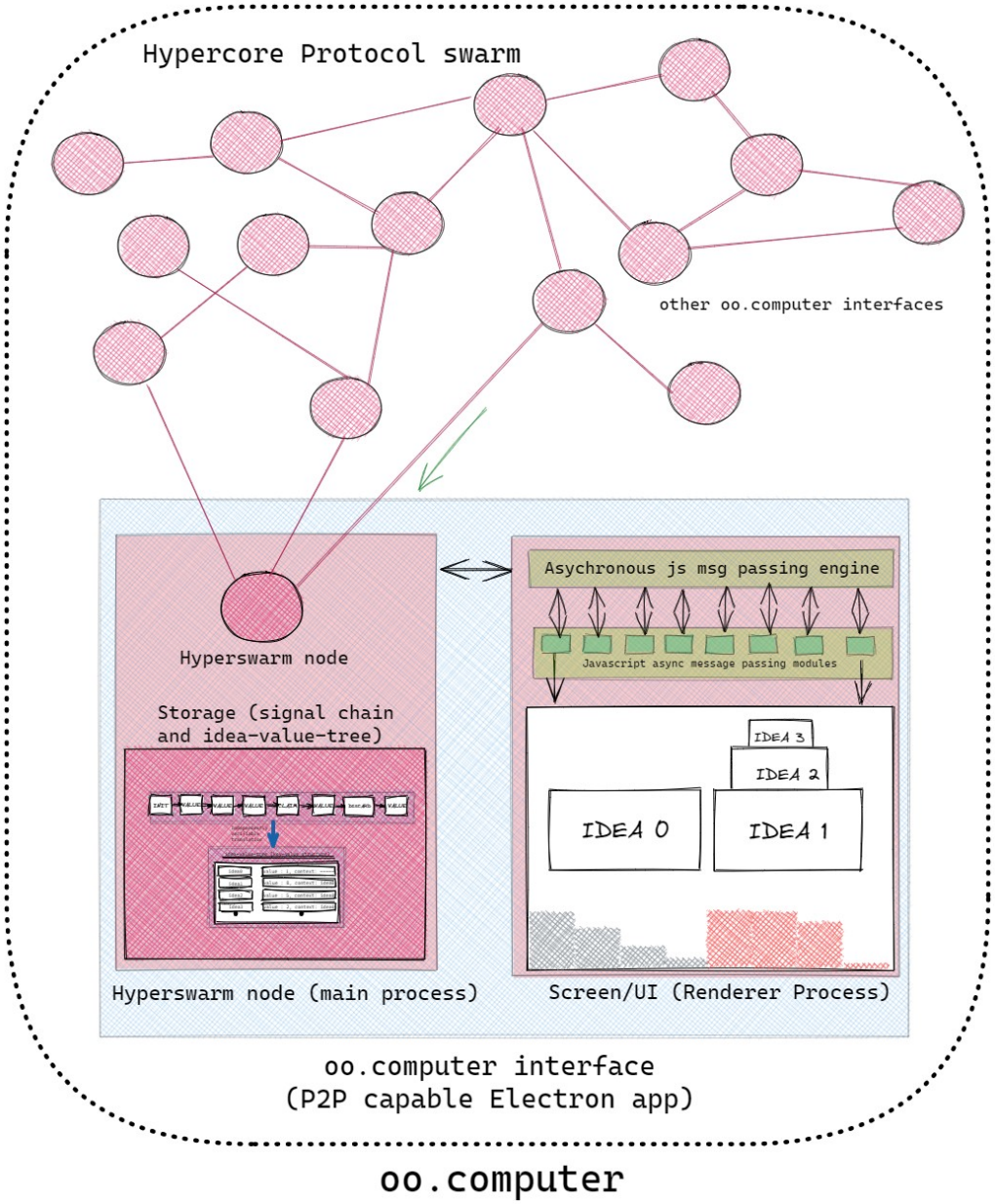


Figure 13: Overview of the present system implementation

Asynchronous Modular Architecture

To achieve transparency, the system itself is built from ideas in the form of modular code components that can be ranked and linked on the system. The intention is that a user can see the paths of accountability associated with every code module. A user can therefore, at any time, select trusted code authors/readers that vouch for particular components.

To facilitate this, an asynchronous modular approach was taken that seeks to emulate the software philosophies espoused by Joe Armstrong. As such, the oo.computer has at its core an asynchronous message passing and queueing mechanism. It is then implemented (to run in a web browser, initially as a peer-to-peer browser built on Electron) using a set of processes that manipulate self contained portions of the DOM (Document Object Model). All of the processes are based on a base template that includes a JSON-RPC message passing send function and receive case-switch message handler. A set of outer

templates is used to construct the required processes (see Table 3 for the set of templates and Table 4 for the additional Javascript libraries that were used). The index html page and the javascript process templates are each represented as ideas within the idea-value-tree. An aim here is that, eventually, javascript programmers will be able to suggest code improvements via the oo.computer itself and receive value-signal and profit share payments for doing so.

Table 3: Brief description of the processes required for the initial oo.computer user interface

Process Name	**Description**
ooSpawner	Controls the creation of new processes in the browser session
ooIdea	For presentation of an idea to the user. Manages the presentation of the icons and the value signals to the user.
ooInput	Enables the user to input ideas in a context
ooFilter	Represents a filter, handles value signals from its member filterers
ooFilterInput	An oo that enables the user to create a new filter
ooFilterMain	Manages the container for the filters.
ooContextManager	Handles value signals for a given context. It calculates the rankings, and co-ordinates the on screen positions for the oos in a the context.
ooPendingTxManager	Handles the collation and sending of signals.
ooIconCreator	Creates the icons for the iconographic display.
ooFilterCurator	Manages the creation and layout of filters
ooContextCurator	An oo that keeps track of the live contexts and routes value signals to context managers.
ooHyperManager	Manages the interface with the hypercore protocol
ooUserComms	Manages on screen user help prompts and menus

Table 4: Additional libraries included

Additional libraries	**Reason for inclusion**
bigInteger	Required for calculation of 256 bit word XOR and rotate functions.
jdenticon	Creates the icons
DOMpurify	Sanitizes (removes scripts) from ideas encoded as html
Rusha	Calculates the SHA1 hashes (to verify correct url retrieved)
web3	Utility functions

Payment facilitation, claim registry and profit-share mechanism

The next stage in the implementation is to build a trustworthy (no practical means of censorship or persecution) registry to allow users to prove when claims of idea initiation were made and to collect and distribute value signal payments. A tokenised version of this has been built as two connected smart contracts in the oo.computer predecessor, ooda.space (ideas.oodaspace.eth and profit.oodaspace.eth, see Appendix III), using the Ethereum blockchain. However, alas, at the time of writing, the large transaction fees make this solution insufficiently scalable. This realisation inspired the change in architecture to

local-first DHT distributed signal chains (using the Hypercore protocol). However, the new version does not yet include the profit share mechanism.

Therefore, in the next stage of implementation, the Ethereum smart contracts will be re-implemented using other systems, whilst maintaining backwards compatibility. Candidate options include Algorand, Arweave, Avalanche, Bitcoin, Bitcoin Cash, Bitcoin SV, Boba, Cardano, Chainlink, Chia, Counterparty, Dusk, Eth2, Fantom, Hedera Hashgraph, Holochain, Internet Computer, Moonbeam, Near, Obyte, Polkadot, Polygon, Secret, Shiden/astar, Skale, Solana, Stellar, Velas, Waves and doubtless many others. Whichever is chosen, it must be able to handle the INVOICE and RECEIPT signals within the oo.computer signal set (see Table 2). It is perhaps feasible that multiple implementations could exist in parallel (including those implemented by parties other than ooda space ltd). In the case where payment facilitators other than ooda space ltd are used, value signallers must indicate which other payment facilitators they will accept. Ultimately, users will be able to judge payment facilitators based on which other value signallers vouch for them.

Concluding thoughts

The primary aim of oo.computer is to accelerate the revelation and realisation of valuable ideas. The vision is that, as ideas progress from conceptual discussions towards reality, value signallers will signal value in contracts and software components to make the ideas real. A special class of ideas might emerge that results in the spawning of self similar entities (e.g. websites, companies, trusts, co-operatives) using the value signal system as their basis for operation. It is also foreseeable that the oo.computer might compete with the traditional patent system, such that inventors are incentivised to quickly disclose valuable ideas in the knowledge that they will be rewarded (both with intellectual credit and financially).

After this, owing to the nature of the oo.computer signal chains and the derived idea-value-tree, a verifiable log of all of the contributing value signals will exist. This will be used as a basis to distribute profits from spawned organisations to the contributors. An etiquette will be sought whereby when one receives value signals for a good idea, then one feels honour bound to treat a portion of the income as fuel for further value signals to bring the idea closer to reality. In any case, the participants will be acting as both investors and capital allocators, whereby the act of declaring an idea to be valuable is synonymous with investing in its eventual realisation.

Glossary

Term	Definition
claim	A claim of ownership of an idea (see CLAIM signal, Table 2).
context	A 256 bit word within an oo representing a context for an *idea*
distance	The number of oos between a given oo and the base oo (zero `0b0.` by default)
filter	An oo which for which child ideas (ideas signalled in the oo's prospective context) are interpreted as value signallers. The value signallers can then weighted by the proportion of value signal amounts and used to filter and rank the oos displayed to the user.
filterer	A value signaller (public key) that has been value signalled in the context of the filter
filter-director	The address which is the context for a filter and which signals value in the filter's filterers.
filter-name	The word used as the idea in a *filter oo*
Hypercore	The base data structure used in the Hypercore protocol (https://hypercore-protocol.org/) used to store the users signal chain in the oo.computer implementation
idea	A 256 bit encoding or fingerprint of an idea.
idea-value-tree	The imaginary tree-like arrangement of ideas and contexts central to the operation of oo.computer (schema in Appendix I).
oo	An iconographic representation of a context, idea and prospective context. When any two of the context, idea and prospective words are known the third can be derived. See Figure 4.
profit-share mechanism	The economic mechanism used to distribute a 10% profit-share portion of value signal payments to previous value signallers.
prospective context	A 256 bit word within an oo representing a context for child *ideas*
signal chain	An independently verifiable (instantiated as a Hypercore) ordered list of signals from the oo.computer signal set.
signal set	The set of signals that are used in the oo.computer system (see Table 2)
value signal	A public promise to pay an equal, non-zero, discretionary, amount to the recognised owner(s) of an idea and its context (see VALUE signal, Table 2). Ten percent is channelled into the oo.computer profit share mechanism.
word	General term for a 256 bit word used in the *idea-value-tree* (a *context, idea or prospective context).*
`0b0.`	`0x0000000000000000000000000000000000000000000000000000000000000000`
`0b1.`	`0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff`

References

Appendices

Appendix I – idea-value-tree schema

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "$id": "http://json-schema.org/draft-07/schema#",
  "title": "The oo.computer idea-value-tree schema",
  "description":"This is the schema for an idea-value-tree in the oo.computer system. This must be derivable from a user's signal chain (a series of independently verifiable oo.computer signals, e.g VALUE, DISCARD, CLAIM, TRANSFER). It defines an object indexed by 256 bit words. The words represent contexts and ideas in the oo.computer idea-value-tree. Each word contains the following:  a reference to its parent and children (if any); the user's recognised owner of the word; flags denoting if the word has been discarded, represents a filter context, or represents a signaller; an object containing the collated value signals of other value signallers; if the word is an idea (encoded as a hash-id), it may also include an object containing the base64 encoded data and content type that the word represents.",
  "type": "object",
  "patternProperties": {
          "0x[0-9a-f]{64}": {
                  "type": "object",
                  "properties": {
                        "parent": { "type": "string", "pattern": "0x[0-9a-f]{64}"},
                        "distance": { "type": "number", "multipleOf" : 1 },
                        "children": { "type": "array", "items": { "type": "string", "pattern": "0x[0-9a-f]{64}"}},
                        "owner": {"type": "string", "pattern": "0x[0-9a-f]{64}"},
                        "isDiscarded": { "type": "bool"},
                        "isSignaller": { "type": "bool"},
                        "isFilter": { "type": "bool"},
                        "signallers": {
                              "type": "object",
                              "patternProperties": {
                                      "0x[0-9a-f]{64}": { "type": "object",
                                                          "properties": {
                                                                  "total": { "type": "number" },
                                                                  "rankedList": { "type": "array", "items":
{ "type": "string", "pattern": "0x[0-9a-f]{64}"}},
                                                                  "totalsObj": {  "type" : "object",
                                                                                  "patternProperties": {
                                                                                        "0x[0-9a-f]{64}":
{ "type": "number", "multipleOf" : 1}
                                                                                  },
                                                                                  "additionalProperties": false
                                                                                },
                                                                  "propsObj": {  "type" : "object",
                                                                                  "patternProperties": {
                                                                                        "0x[0-9a-f]{64}":
{ "type": "number", "multipleOf" : 1}
                                                                                  },
```

```
                                                                                    "additionalProperties": false

                                                                                }

                                                                },

                                                                "required": [ "total","rankedList", "totalsObj",
"propsObj"]

                                        },

                                    "additionalProperties": false

                                    }

                            },

                            "media" : { "type": "object",

                                        "properties": {

                                              "prefix" : { "type": "string", "pattern": "0x[0-9a-f]{64}"},

                                              "content" : { "type": "string", "contentEncoding":
"base64","contentMediaType": "[a-z]\/[a-z]"},

                                              "contentMediaType": {"type": "string", "pattern": "[a-z]\/[a-z]"},

                                              "ipfs":  { "type": "string","pattern": "Qm[1-9A-HJ-NP-Za-km-z]{44,}|
b[A-Za-z2-7]{58,}|B[A-Z2-7]{58,}|z[1-9A-HJ-NP-Za-km-z]{48,}|F[0-9A-F]{50,}|f[0-9a-f]{50,}"},

                                              "arweaveTx": { "type": "string","pattern": "[0-9A-Za-z-_]{43}"}

                                          },

                                          "required": [ "prefix","content", "contentMediaType" ]

                                    },

                            "additionalProperties": { "type": "string" },

                            "required": [ "parent","distance", "children", "owner", "isDiscarded", "isSignaller",
"isFilter", "signallers" ]

                        },

                    "additionalProperties": false

                }

            }

}
```

Appendix II – oo.computer signal set Javascript object

```
let oodaSpaceSignalSet = {

          man: function (){return new Object(
            {
            $SIGNALLER : `The ed25519 public key representing the signaller's oo.computer signal chain (a
hypercore), in hex format.`,
            $OTHERSIGNALLER : `The ed25519 public key representing oo.computer signal chain (a hypercore) of a
signaller other than $SIGNALLER, in hex format`,
            $IDEATREE : `The ed25519 public key representing the signaller's oo.computer idea-value-tree (a
hypercore protocol url), in hex format.`,
            $CONTEXT : `A 256bit word represent a context in an oo. An oo is three interrelated 256bit words with
an integer as defined in the oo.computer system.`,
            $CONTEXTOWNER : `The public key of the owner of $CONTEXT as recognised by $SIGNALLER.`,
            $IDEA : `A 256bit word represent a context in an oo. An oo is three interrelated 256bit words with an
integer as defined in the oo.computer system.`,
            $IDEAOWNER : `The public key of the owner of $IDEA as recognised by $SIGNALLER.`,
            $DISTANCE : `The distance of the referenced oo. An oo is three interrelated 256bit words with an
integer (distance) as defined in the oo.computer system.`,
            $PROSPECTIVECONTEXT : `A 256bit word representing a prospective context in an oo. An oo is three
interrelated 256bit words with an integer as defined in the oo.computer system.`,
            $PROSPECTIVECONTEXTOWNER : `The public key of the owner of $PROSPECTIVECONTEXT as recognised by
$SIGNALLER.`,
            $AMOUNT : `An amount referenced in the signal. Must be a positive number representable in
javascript.`,
            $PREVSIGNALHASH : `The hash of the parent signal, as selected by $SIGNALLER`,
            $SIGNALNO : `An integer reference for the signal from $SIGNALLER - typical use is for the sequence
number`,
            $CURRENCY : `An indication of the currency. It is suggested to use an ISO4217 code (e.g. USD) or
similar (e.g. BTC). Metric prefix used as required (e.g. uBTC, dGBP, kJPY);`,
            $WORD : `A 256bit word referenced in the signal.`,
            $WORDS : `A list of 256bit words referenced in the signal.`,
            $OTHERSIGNAL : `Another signal referenced in the transaction.`,
            $OTHERSIGNALS : `A list of other signal (identifed with hash and signature) referenced in the
```

```
signal.`,
            $SIGNALSETVERSION : `The SHA256 hash of a protocol object (utf-8 representation of a javascript
object)`
          })},

          INIT: function(signaller='$SIGNALLER',signalsetversion='$SIGNALSETVERSION',ideatree='$IDEATREE') {
            let signal =`INIT:

I will create an oo.computer signal chain using the oo.computer signal set on the hypercore with address hyper://$
{signaller}.

I will use VALUE signals to indicate the value of digitally encoded ideas in contexts. I will use these to create
an oo.computer idea tree at hyper://${ideatree}

I will use DISCARD signals to indicate that previous value signals, for a given idea in context, should be
discarded.

I will use CLAIM and TRANSFER signals to claim ownership of ideas or contexts and to transfer ownership claims to
others.

I will use the INVOICE signals to request value signal payments.

I will use the RECEIPT signals after invoiced payments have been received.

I will use the VERSION signal to indicate any change the to the set of signals I use. Any prior valid signals
shall continue to be valid.

I will use the HACKED signal to indicate that my private keys have been compromised.

I give permission for all of my signals to be freely duplicated and stored.

The oo.computer signal set is defined in the javascript signal set object which, when written in UTF-8 encoded
text has the SHA256 hash of 0x${signalsetversion}.

I will create an oo.computer idea tree with the address hyper://${ideatree} such that it can be verifiably derived
from the above signal chain.

`
let signalHash = ''//hash of signal
            return new Object(
                                {
                                          SIGNALTYPE: 'INIT',
                                          SIGNALLER : signaller,
                                          SIGNAL :  String(signal),
                                          SIGNATURE : '',
                                          setSignature : function(signature){this.SIGNATURE = signature;},
                                          SIGNALNO : 0,
                                          SIGNALHASH : signalHash,
                                          PREVSIGNALHASH : "0x00"
                                });
          },

          VALUE:
function(signaller='$SIGNALLER',context='$CONTEXT',contextOwner='$CONTEXTOWNER',idea='$IDEA',ideaOwner='$IDEAOWNER
',distance='$DISTANCE',prospectivecontext='$PROSPECTIVECONTEXT',prospectiveContextOwner='$PROSPECTIVECONTEXTOWNER'
,amount='$AMOUNT',currency='$CURRENCY',signalNo='$SIGNALNO',prevSignalHash='$PREVSIGNALHASH') {
            let signal =`VALUE:

I hereby signal a value of ${amount} uBTC in the following oo within the oo.computer value tree:

context: ${context}
idea: ${idea}
distance: ${distance}

with a resulting prospectiveContext of: ${prospectivecontext}

I recognise that:

${contextOwner} is the owner of context
${ideaOwner} is the owner of idea
${prospectiveContextOwner} is the owner of prospectiveContext

I will pay the owner* of context 0.3 x ${amount} uBTC in ${currency}  within 1 month of a valid invoice signal.
I will pay the owner* of idea 0.3 x ${amount} uBTC in ${currency}  within 1 month of a valid invoice signal.
I will pay the owner* of prospectiveContext 0.3 x ${amount} uBTC in ${currency}  within 1 month of a valid invoice
signal.
I will pay future value signallers** 0.1 x ${amount} uBTC in ${currency} for faciliation of the payment.

*If I am the owner, then I will instead pay future value signallers** 0.3 x ${amount} uBTC in ${currency}  within
1 month of a valid invoice signal.
** via the ooda space ltd profit share system

I will store the data from which the idea is derived (if any) and track its ownership and transfer signals.

If idea is a value signaller, I will store a copy of their value and discard signals.
If context is a value signaller, I will store a copy of their value and discard signals that have a context of $
{prospectivecontext}.
```

```
This is signal number ${signalNo} in my set of signals.

I reserve the right to invalidate this signal using a signed discard signal.
`
let signalHash = ''//hash of signal
            return new Object(
                                {
                                        SIGNALTYPE: 'VALUE',
                                        SIGNALLER : signaller,
                                        CONTEXT : context,
                                        IDEA : idea,
                                        DISTANCE : distance,
                                        PROSPECTIVECONTEXT : XORcontextidea(context,idea,distance),
                                        AMOUNT : amount,
                                        CURRENCY : currency,
                                        SIGNAL :  String(signal),
                                        SIGNATURE : '',
                                        setSignature : function(signature){this.SIGNATURE = signature;},
                                        SIGNALNO : signalNo,
                                        SIGNALHASH : signalHash,
                                        PREVSIGNALHASH : prevSignalHash
                                });
            },




            DISCARD:
function(signaller='$SIGNALLER',context='$CONTEXT',idea='$IDEA',distance='$DISTANCE',prospectivecontext='$PROSPECT
IVECONTEXT',signalNo='$SIGNALNO',prevSignalHash='$PREVSIGNALHASH') {
            let signal =`DISCARD:

I am ${signaller}

I hereby invalidate all my prior value signals in the following oo within the oo.computer value tree:

context: ${context}
idea: ${idea}
distance: ${distance}

I will honour any invoiced value signals but I will no longer make any uninvoiced payments.

            `
            let signalHash = ''//hash of signal
                        return new Object(
                                        {
                                                SIGNALTYPE: 'DISCARD',
                                                SIGNAL : signal,
                                                SIGNALLER : signaller,
                                                CONTEXT : context,
                                                IDEA : idea,
                                                DISTANCE : distance,
                                                PROSPECTIVECONTEXT : prospectivecontext,
                                                SIGNAL :  signal,
                                                SIGNATURE : '',
                                                SIGNALNO : signalNo,
                                                SIGNALHASH : signalHash,
                                                PREVSIGNALHASH : prevSignalHash
                                        });
            },

            INVOICE:
function(signaller='$SIGNALLER',otherSignaller='$OTHERSIGNALLER',amount='$AMOUNT',currency='$CURRENCY',idea='$IDEA
',words='$WORDS',otherSignals='$OTHERSIGNALS',signalNo='$SIGNALNO',prevSignalHash='$PREVSIGNALHASH') {
            let signal =`INVOICE:

I request payment of ${amount} ${currency} from ${otherSignaller} in relation to:

${words}

in the following value signals:

${otherSignals}

            `
            let signalHash = ''//hash of signal
                        return new Object(
                                        {
                                                SIGNALTYPE: 'INVOICE',
                                                SIGNAL : signal,
                                                SIGNALLER : signaller,

                                              OTHERSIGNALLER : otherSignaller,
                                                AMOUNT : amount,
                                                CURRENCY : currency,
                                                WORDS : words,
                                                OTHERSIGNALS :  otherSignals,
```

```
                                                SIGNATURE : '',
                                                SIGNALNO : signalNo,
                                                SIGNALHASH : signalHash,
                                                PREVSIGNALHASH : prevSignalHash
                                        });
        },

        VERSION:
function(signaller='$SIGNALLER',ideatree='$IDEATREE',signalsetversion='$SIGNALSETVERSION',signalNo='$SIGNALNO',pre
vSignalHash='$PREVSIGNALHASH') {
        let signal =`VERSION:

From now on I will be using the oo.computer signal set object which, when written in UTF-8 encoded text has the
SHA256 hash of 0x${signalsetversion}.

I will use these signals to create an oo.computer idea tree at hyper://${ideatree}

        `
        let signalHash = ''//hash of signal
                        return new Object(
                                        {
                                                SIGNALTYPE: 'CHANGE_SIGNALSET',
                                                SIGNAL : signal,
                                                SIGNALLER : signaller,
                                                SIGNALSETVERSION : signalsetversion,
                                                SIGNATURE : '',
                                                SIGNALNO : signalNo,
                                                SIGNALHASH : signalHash,
                                                PREVSIGNALHASH : prevSignalHash
                                        });
        },

        CLAIM:
function(signaller='$SIGNALLER',word='$WORD',signalNo='$SIGNALNO',prevSignalHash='$PREVSIGNALHASH') {
        let signal =`CLAIM

I initiate and claim ownership of ${word}.

        `
        let signalHash = ''//hash of signal
                        return new Object(
                                        {
                                                SIGNALTYPE: 'CLAIM',
                                                SIGNAL : signal,
                                                SIGNALLER : signaller,
                                                WORD : word,
                                                SIGNATURE : '',
                                                SIGNALNO : signalNo,
                                                SIGNALHASH : signalHash,
                                                PREVSIGNALHASH : prevSignalHash
                                        });
        },

        HACKED: function(signaller='$SIGNALLER',signalNo='$SIGNALNO',prevSignalHash='$PREVSIGNALHASH') {
        let signal =`HACKED:

My private key has been compromised.

        `
        let signalHash = ''//hash of signal
                        return new Object(
                                        {
                                                SIGNALTYPE: 'CLAIM',
                                                SIGNAL : signal,
                                                SIGNALLER : signaller,
                                                WORD : word,
                                                SIGNATURE : '',
                                                SIGNALNO : signalNo,
                                                SIGNALHASH : signalHash,
                                                PREVSIGNALHASH : prevSignalHash
                                        });
        },

        TRANSFER:
function(signaller='$SIGNALLER',word='$WORD',otherSignaller='$OTHERSIGNALLER',signalNo='$SIGNALNO',prevSignalHash=
'$PREVSIGNALHASH') {
        let signal =`TRANSFER:

I hereby transfer ownership of ${word} to ${otherSignaller}.

        `
        let signalHash = ''//hash of signal
                        return new Object(
                                        {
                                                SIGNALTYPE: 'TRANSFER',
                                                SIGNAL: signal,
                                                SIGNALLER : signaller,
                                                WORD : word,
```

```
                                                OTHERSIGNALLER : otherSignaller,
                                                SIGNATURE : '',
                                                SIGNALNO : signalNo,
                                                SIGNALHASH : signalHash,
                                                PREVSIGNALHASH : prevSignalHash
                                            });
          }
}
```

Appendix III – oodaspace.eth smart contract

```
pragma solidity ^0.6.0;

import "https://github.com/OpenZeppelin/openzeppelin-solidity/contracts/token/ERC721/ERC721.sol";
import "https://github.com/OpenZeppelin/openzeppelin-solidity/contracts/token/ERC1155/ERC1155Burnable.sol";
import "https://github.com/OpenZeppelin/openzeppelin-solidity/contracts/token/ERC1155/ERC1155Receiver.sol";
//import "https://github.com/OpenZeppelin/openzeppelin-solidity/contracts/math/SafeMath.sol";


// © (c) Copyright 2020 ooda space ltd

/// @author ooda space ltd
/// @title  oo.computer Tokens for Profit Share (ToPS)
/// @notice mints ERC1155 tokens that have possibility of being exchanged for a share in ooda space ltd profits.
/// @notice ERC1155 tokens are batched by week number


contract oodaspaceltdToPSContract is ERC1155, ERC1155Receiver {

    address public minterAddress;
    address private oodaspaceltd = 0xC66b74e8Ad6672EfD8F7ed91D7A37CEf44Db3Cef;
    uint256 public currentWeekSec = now;
    uint256 public currentWeekNo = 1;
    uint256 totalSupply = 0;
    bool transferPausedForProfitShare = false;
        mapping(uint256 => uint256) public supplyPerWeek;
        mapping(address => uint256) public balanceOfETH;
    constructor() public ERC1155("https://oo.computer/api/week/{id}.json") {
        /*minterAddress = msg.sender;
        _mint(oodaspaceltd,0,1,'');
        supplyPerWeek[0] = 1;*/
    }

    bool internal isInitialised = false;
    function init(address minter) external {
        require(!isInitialised);
        minterAddress = minter;
        _mint(oodaspaceltd,0,1,'');
        supplyPerWeek[0] = 1;
        isInitialised = true;
    }

    /// @notice updates the address that represents the company
    /// @param newAddress The new address
    function setCompanyAddress(address newAddress) external {
                require(msg.sender == oodaspaceltd);
                oodaspaceltd = newAddress;
        }

        /// @notice mints a new ERC1155 token
        /// @notice the tokens are minted in per-week batches, this is to allow a future contract to perform
    /// @param receiver the address that will control the minted token.
    /// @param amt the number of tokens to mint
    function mint(address receiver,uint256 amt) external {
        require(msg.sender == minterAddress);
        if (now >= currentWeekSec + 7 days){
            currentWeekNo++;
            currentWeekSec = currentWeekSec + 7 days;
            supplyPerWeek[currentWeekNo] = 0;
        }
        supplyPerWeek[currentWeekNo] += amt;
        _mint(receiver,currentWeekNo,amt,'');
    }

        /// @notice sets the token uri
    /// @param newuri the uri
    function SetURI(string calldata newuri) external {
        require(msg.sender == oodaspaceltd);
        _setURI(newuri);
    }

    /// @notice This function calculates the ETH and tokens that can be exchanged for a given week's tokens
    /// @param weekNo the week number id (token Id) of the ToPS ERC1155 tokens to be exchanged
    /// @param amt the number of ERC1155 tokens to be exchanged
    /// @return weekNoETH the amount of ETH exchangable with the ToPS ERC1155 tokens, nextTknAmt the number of the
next week's ERC1155 ToPS exchangeable
```

```
    function calcProfitShare(uint256 weekNo,uint256 amt) public view returns(uint256,uint256){
        uint thisWeekPortion = supplyPerWeek[weekNo];
        uint priorWeekPortion = supplyPerWeek[weekNo] * 415; // for 100% redistribution over 8 years
        uint numeratorETH = (amt * supplyPerWeek[weekNo]) * 1000000000000000000000000000000000000000;
        uint numeratorToPS = amt * 100000000000000000000000000000000000 * supplyPerWeek[weekNo+1] * 415;
        uint denominator = ((thisWeekPortion * 100000000000000000000000000000000000) + (priorWeekPortion *
1000000000000000000000000000000000000));
        uint weekNoETH = numeratorETH.div(denominator);
        uint nextTknAmt = numeratorToPS.div(denominator);
        return (weekNoETH, nextTknAmt);
    }

    /// @notice This function is called on reciept of ERC1155 tokens.
    /// @notice It exchanges the received token for ETH and ToPS for the following week)
    /// @param operator - the operator contract address of the ERC1155s (this contract!)
    /// @param sender - the account sending the ERC1155 tokens
    /// @param weekNo - the weekNo of the ToPS tokens received
    /// @param amt - the amount of ToPS received
    /// @param data - unused parameter (required for ERC1155 standard)
    /// @return this function signature (required for ERC1155 standard)
    function onERC1155Received(address operator, address sender, uint256 weekNo, uint256 amt, bytes calldata data)
external override returns (bytes4) {
        require(operator == address(this));
        require(weekNo < currentWeekNo-1);
        uint256 amtETHThisWeek;
        uint256 amtofNext;
        (amtETHThisWeek, amtofNext) = calcProfitShare(weekNo,amt);
        _burn(sender,weekNo,amt);
        balanceOfETH[sender] += amtETHThisWeek;
        _mint(sender,weekNo+1,amtofNext,'');
        return bytes4(keccak256("onERC1155Received(address,address,uint256,uint256,bytes)"));
    }

    /// @notice This function is called on reciept of ERC1155 tokens.
    /// @notice It exchanges the received token for ETH and ToPS for the following week)
    /// @param operator - the operator contract address of the ERC1155s (this contract!)
    /// @param sender - the account sending the ERC1155 tokens
    /// @param weekNos - the week numbers of the ToPS tokens received
    /// @param amts - the amounts of ToPS received, per weekNo
    /// @param data - unused parameter (required for ERC1155 standard)
    /// @return this function signature (required for ERC1155 standard)
    function onERC1155BatchReceived(address operator, address sender, uint256[] calldata weekNos, uint256[]
calldata amts, bytes calldata data) external override returns (bytes4) {
            require(operator == address(this));
            uint i;
            for (i = 0;i < weekNos.length;i++){
            require(weekNos[i] < currentWeekNo-1);
            uint256 amtETHThisWeek;
            uint256 amtToPSofNext;
            uint256 next = weekNos[i] + 1;
            (amtETHThisWeek, amtToPSofNext) = calcProfitShare(weekNos[i],amts[i]);
            _burn(sender,weekNos[i],amts[i]);
            balanceOfETH[sender] += amtETHThisWeek;
            _mint(sender,next,amtToPSofNext,'');
            }
        return bytes4(keccak256("onERC1155Received(address,address,uint256,uint256,bytes)"));
    }

    // withdrawer
    /// @notice This function allows users to withdraw ETH
    /// @param withdrawer the address withrawing ETH
     function withdrawETH(address withdrawer,uint256 amt) external {
         require(balanceOfETH[withdrawer] >= amt);
        balanceOfETH[withdrawer] -= amt;
        msg.sender.transfer(amt);
        }
}

/// @author ooda space ltd
/// @title The oo.computer smart contract version 0
/// @notice The main contract that handles value/discard signals and mints tokens when new ideas or contexts are
initiated
contract oodaspacev0 is ERC721 {

   address private oodaspaceltd = 0xC66b74e8Ad6672EfD8F7ed91D7A37CEf44Db3Cef;

    constructor() public ERC721("oodaspace", "◎") {//◌◌◌◌◌◌ \◌◌ / //o‿o‿d‿a‿.‿s‿p‿a‿c‿e
        //oodaspaceltdToPSContractAddress = new oodaspaceltdToPSContract();
        //_mint(oodaspaceltd, 0);
    }

    event ValueSignal(address indexed signaller, uint256 indexed context, uint256 indexed prospectiveContext,
uint256 amount, int256 distance);
        event DiscardSignal(address indexed signaller, uint256 indexed context, uint256 indexed
prospectiveContext, uint256 amount, int256 distance);
        event Found(address indexed initiator, uint256 indexed word);

        // Word based
```

```
        mapping(uint256 => uint256) public balanceOfETH; //
        mapping(bytes32 => uint256) public notificationBlockNo;            // mapping used to prevent front-
running
    mapping(uint256 => uint256) public initBlockNo;
    mapping(uint256 => address) public initiators;

        // internal

        // the contract location that handles the Tokens for Profit Share (ToPS)
        oodaspaceltdToPSContract public oodaspaceltdToPSContractAddress;
        bool contractSet = false;

    /// @notice updates the address that represents the contract that handles ERC1155 profit share claim tokens
    /// @param contractAddress The new address
        function initToPSAddress(oodaspaceltdToPSContract contractAddress) external {
                require(msg.sender == oodaspaceltd);
                require(!contractSet);
                oodaspaceltdToPSContractAddress = contractAddress;
                _mint(oodaspaceltd, 0);
                _mint(oodaspaceltd, 0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff);
                contractSet = true;
        }

    // address setters
    /// @notice updates the address that represents the company
    /// @param newAddress The new address
        function setCompanyAddress(address newAddress) external {
                require(msg.sender == oodaspaceltd);
                oodaspaceltd = newAddress;
        }

        /// @notice sets the token uri
    /// @param newuri the uri
    function setBaseURI(string calldata newuri) external {
        require(msg.sender == oodaspaceltd);
        _setBaseURI(newuri);
    }

        /// @notice sets the token uri
        /// @notice unused, included for future optionality
    /// @param newuri the uri
    function setTokenURI(uint256 word, string calldata newuri) external {
        require(msg.sender == oodaspaceltd);
        _setTokenURI(word,newuri);
    }

    function exists(uint256 word) external view returns (bool) {
        return _exists(word);
    }

    //bit manipulation funcs
    /// @notice Rotates a 256 bit word right by n*3 bits
    /// @param value The input word to be bit rotated
    /// @param n The word is rotated right by 3*n
    /// @return the bit rotated word
        function rotr(uint256 value, int256 n) public pure returns (uint256) {
        int256 shift = (n * 3) % 256;
        if (value == 0) {
          return value;
        }
        return (value >> shift) | (value << (256 - shift));
        }

    /// @notice Rotates a 256 bit word left by n*3
 bits
    /// @param value The input word to be bit rotated
    /// @param n The word is rotated left by 3*n
    /// @return the bit rotated word
        function rotl(uint256 value, int256 n) public pure returns (uint256) {
                int256 shift = (n * 3) % 256;
                if (value == 0) {
                        return value;
                }
                return (value << shift) | (value >> (256 - shift));
        }

    /// @notice Performs a bitwise XOR on two words after bitwise rotating them left and right respectively by 3*n
bits
    /// @param a first input word, this is bitwise rotated right by 3*n bits
    /// @param b first input word, this is bitwise rotated left by 3*n bits
    /// @param n The words are rotated by 3* the number of bits
    /// @return the bitwise XOR of the bit rotated words
    function xorwithshift(uint256 a, uint256 b, int256 n) public pure returns (uint256) {
        return rotr(a,n) ^ rotl(b,n);
    }

    // value and discard signals, notifications and claims
```

```
    /// @notice Front-running protection: the user can first notify un-initiated ideas and contexts using this
function
    /// @notice this allows them a window in which to claim any front-ran tokens (and should act as a
disincentive)
    /// @dev The resultant mapping is checked in the signalValue function before minting of ERC721 tokens
    /// @param notificationHashes an array of sha256 of the initiator address, the claimed word and a user defined
salt
        function notify(bytes32[] calldata notificationHashes) external {
            uint i;
            for (i = 0;i < notificationHashes.length;i++){
                if (notificationBlockNo[notificationHashes[i]] == 0){
                    notificationBlockNo[notificationHashes[i]] = block.number;
                }
            }
        }

    /// @notice This function allows the front-end to reliably create valid claim hashes for a given salt
    /// @param word - the word (context, idea, or prospectiveContext) to be claimed
    /// @param senderAddr the address that will be used to send the claiming value signal
    /// @param salt a user defined word to make the claimed word harder to guess for an adversary
        function notificationHashCreator(uint256 word,address senderAddr, uint256 salt) external pure returns
(bytes32){
            return keccak256(abi.encodePacked(senderAddr,uint256(word),uint256(salt)));
        }

    /// @notice This is where the user can, at their option claim their ERC721 tokens
    /// @notice the tokens must be owned in order to be able to withdraw value signal payments
    /// @param words an array of the words (ideas or contexts)
    /// @param salts - the salts used in the notification for the words only needed if within 7200 blocks of first
value signal
    function claimTokens(uint256[] calldata words, uint256[] calldata salts) external {
        uint i;
            for (i = 0;i < words.length;i++){
                bytes32 notificationHash =
keccak256(abi.encodePacked(msg.sender,uint256(words[i]),uint256(salts[i])));
                if(block.number < initBlockNo[words[i]] + 7200){
                    // allows ~1 day for claim to be made
                   if(notificationBlockNo[notificationHash] > 0 && notificationBlockNo[notificationHash] <
initBlockNo[words[i]]){
                       _transfer(address(this),msg.sender,words[i]);
                       delete notificationBlockNo[notificationHash];
                       delete initiators[words[i]];
                   }
            }
            else if(block.number > initBlockNo[words[i]] + 8000000){
                    // if unclaimed in > ~3 years can be claimed by company
                    require(msg.sender == oodaspaceltd);
                    _transfer(address(this),msg.sender,words[i]);
                    delete notificationBlockNo[notificationHash];
                    delete initiators[words[i]];
            }
            else {
                // tokens transferred to initiator
                _transfer(address(this),initiators[words[i]],words[i]);
                delete notificationBlockNo[notificationHash];
                delete initiators[words[i]];
            }
             }
    }

    /// @notice This function is called by a user to signal value in an "oo" (an idea in context)
    /// @notice if the context or idea words are new, and properly claimed, an ERC721 token is minted.
    /// @notice value signal payments go to the holder of the ERC721 token
    /// @param context the context word
    /// @param prospectiveContext the prospective context word - this is combined  with the context and n to give
the idea
    /// @param n this is the "distance" that modulates how the idea is derived
         function signalValue(uint256 context, uint256 prospectiveContext, int256 n) external payable {
         require(msg.value >= .0001 ether,'A value signal must be >= .0001 ether');
        uint256 idea = xorwithshift(context,prospectiveContext,n);

        if (!(_exists(context))){
                        initBlockNo[context] = block.number;
                        _mint(address(this), context);
                            emit Found(msg.sender, context);
        }
        if (!(_exists(idea))){
                        initBlockNo[idea] = block.number;
                _mint(address(this), idea);
                     emit Found(msg.sender, idea);
        }
        if (!(_exists(prospectiveContext))){
                        initBlockNo[prospectiveContext] = block.number;
                    _mint(address(this), prospectiveContext);
                             emit Found(msg.sender, prospectiveContext);
        }

        // 30% of value signal amount goes to holder of idea, context and prospectiveContext tokens, 10% (plus
```

```
remainder) goes to oodaspaceltd
        uint256 amtr = msg.value % 10;
        uint256 amttenth = msg.value / 10;
        uint256 amt3 = amttenth*3;
         uint256 amtoo = amttenth + amtr;
        balanceOfETH[0] += amtoo; // 0 is owned by oodaspaceltd
        balanceOfETH[context] += amt3;
        balanceOfETH[idea] += amt3;
        balanceOfETH[prospectiveContext] += amt3;

        emit ValueSignal(msg.sender, context, prospectiveContext, msg.value, n);

        // this mints profit share tokens for the value signaller
        oodaspaceltdToPSContractAddress.mint(msg.sender,amtoo);
         }

    /// @notice This function is called by a user to signal that an "oo"  should be discarded(an idea in context)
    /// @param context the context word
    /// @param prospectiveContext the prospective context word - this is combined  with the context and n to give
the idea
    /// @param n this is the "distance" that modulates how the idea is derived
         function signalDiscard(uint256 context, uint256 prospectiveContext, int256 n) external payable{
             require(msg.value >= .0001 ether,'A discard signal must be >= .0001 ether');
             uint256 idea = xorwithshift(context,prospectiveContext,n);
         require(_exists(context) && _exists(idea) && _exists(prospectiveContext));
                  balanceOfETH[0] += msg.value; // 0 is owned by oodaspaceltd
                  emit DiscardSignal(msg.sender, context, prospectiveContext, msg.value, n);
    }

    // withdrawer
    /// @notice This function allows users to withdraw ETH
    /// @param tokenIds the tokenIds from which to withdrawETH
         function withdrawETH(uint256[] calldata tokenIds) external {
             uint i;
             for (i = 0;i < tokenIds.length;i++){
                 require(msg.sender == ownerOf(tokenIds[i]));
                 require(block.number > initBlockNo[tokenIds[i]] + 10000);
                 balanceOfETH[tokenIds[i]] = 0;
                 msg.sender.transfer(balanceOfETH[tokenIds[i]]);
             }
         }
}
```