Indra's Pearls
Blockchain to Block Lattice

John Small

May 20, 2019
Version 0.0.7

Abstract

The elegance and simplicity of Satoshi's original design of Bitcoin has proved its worth in the last decade. It does however suffer from a scaling problem.

Preserving as much of that elegance and simplicity as we should, we propose sharding a blockchain into a block lattice to ameliorate the scaling problem. The shards in the lattice are mapped to an abstract space equiped with a simple Euclidian metric which defines the concept of distance between shards. Therefore shards have nearest neigbours. When a new block is created in each shard then as well as using the hash of the previous block it uses the hashes of the previous blocks of its nearest neighbours to form a merkle root of hashes which is included in the new block. This binds all the shards together such that reversing the work of any shard requires reversing the work of all shards. Each block receives confirmations when its hash is included in the next block in the same shard, and also in the next block in neighbouring shards. This rapidly adds confirmation depth, increases the entropy, and so achieves finality more quickly.

In a sharded system it's hard to maintain the abstraction of account state, but relatively easy to work with transactions as in a UTXO system. Therefore it's a UTXO based system.

We also take the implicit assumptions in existing UTXO systems, make them explicit and then generalise. This makes it possible to have most of the features of fungible (ERC20) tokens and non-fungible (ERC721, ERC1155) tokens and to exchange them in transactions. The conservation rules of UTXO handle the accounting by default.

We introduce a new consensus method, a modified DPOS system which partitions the orthogonal vectors implicit in present DPOS systems. It also uses logarithms of the vectors which favours smaller producers and hence reduces centralisation. We also describe a way to allow anyone to claim stakes by presenting a proof that a block is

invalid.

The next stage will include ideas to extend the type of records from simple scalar quantities to non-scalars, e.g. durations and geographic areas. This makes the system ideal for recording and exchanging property inside virtual worlds and in the long run in the real world.

This is a draft. It's a work in progress. The latest version will always be available at
`https://www.indras-pearls.net/about-us/white-paper`

Contents

1 Introduction

This paper outlines a natural extension of blockchains into block lattices. Moving from a blockchain to a block lattice is much easier if it's UTXO based. Being UTXO based allows us to extend the UTXO model. Being sharded forces us to ensure fast finality, which in turn requires a restricted POS/DPOS system. We identify the new possibilities that open up and the problems we're going to face. The project is in an active research and development. We have a working proof of concept which has actually been running on up to 15 shards. As the design evolves some ideas will be discarded and others put in their place. I've taken the view that perfection is the enemy of done and so while it's not perfect it does work.

Some of the idea introduced have been developed or have been suggested by other groups, however the idea to combine them in one package is unique. Also the method of probalistiticaly allocating the right to produce a block according the logarithms of independent vectors forming 'stake' appears to be completely novel.

2 Basic concepts

2.1 Standard blockchain

(Nakamoto, 2008) introduced the notion of a chain of blocks of electronic transactions where each block timestamps the transactions and thus prevents double spending. The basic elements of a block chain are represented in this figure. Time goes from left to right, each block takes a hash from the previous block, data for that block and a nonce to create a new block hash, which is used in the next block.

2.1.0.1 Blockchains are a thermodynamic process. Each block depends on a hash of the previous block and therefore any change in any previous block requires rewriting the entire chain of blocks. To create a new block each participant must prove they've peformed computational work by

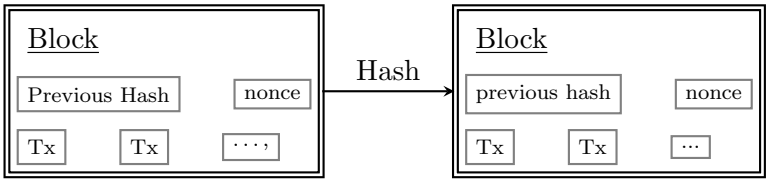

Figure 1: The essential elements of a block chain from (Nakamoto, 2008)

solving a cryptograhic puzzle that depends on the block they're proposing to create. The effort required to do the work consumes lots of energy which is dissipated as heat. Therefore the records in the blockchain are considered to be irrevocable. From a high level view of the process the fact that it's hard to alter previous records is at root because of the Second Law of Thermodynamics. The Second Law of Thermodynamics accounts for the asymmetry between past and future and this is why blockchains can create a sequence of timestamped records.

In this sense mining Bitcoin, or any other cryptocurrency, is a type of refrigerator that takes a source of low entropy, electricty, and produces something even lower entropy, a highly ordered stream of electrical signals and records stored in computer memory, and expels waste heat. The total entripy always increases, and this is the fundamental reason why it's hard to reverse records written into the blockchain. The progressive extension of the blockchain approximates the increase in entropy by summing the amount of computational work that went into creating each block. When the chain splits, clients decide which chain to follow by comparing the total amount of work, and taking the branch with the most accumulated work, i.e. the greatest increase of entropy that has occured as a result of the work, and therefore the hardest to reverse. But...

Entropy can also be increased by expansion in space. That's why heat engines work, they heat a gas, the heated gas tries to expand and that expansion is converted into mechanical work. Going from a hot gas in a confined space to a cooler gas in a larger space increases the entropy. Therefore it seems obvious and natural to extend the concept of a blockchain as a progression in a single dimension of time, to a lattice of blocks that increase entropy by expanding outwards in space.

2.2 Prior work

Since this is such an obvious idea, it's been suggested before, but rather surprisingly there seems to be only one active project pursuing development of it Kadena (Will Martino, Monica Quaintance, 2018). The idea of merge mining between shards was suggested in the early stages of Ethereum but dismissed under the heading 'What are some trivial but flawed ways of

solving the problem?' (Ray and Buterin, 2019), though the reasons don't apply for this design. Also the notion of exchanging hashes between shards is making a re-appearence in Ethereum 2.0. The Kadema team were very dilgent in their background research and dug up one comment from 2014, now only avaialble in the Web Archive about a braided rope of blockchains (Marc-André Bélanger, 2014). In addition the IOTA cryptocurrency introduces the concept of a 'tangle' (Popov, 2017). But that's not a blockchain so it's not directly comparable.

2.3 Constructing a block lattice

To construct a block lattice we have to introduce the concept of space. The notion of 'expanding outwards' also requires metric on that space so that the notion of 'outwards' has meaning. The simplest way to do this is to divide up the blockchain into many blockchains, called shards (as in shards of glass), and map them to a space equiped with a metric such that 'distance' between shards has a consistent meaning Wikipedia, 2009. With a notion of distance in place then each shard has closest neighbours. We now require that each new block depends on the hash of the previous block and also on the hash of its neighbours. The consequence is that any change to any data in a block not only propagates forwards in time but also expands outwards in space, which inceases the entropy much faster and makes things much harder to reverse.

This is illustrated by removing the details inside the blocks and focusing on the hash transferred from one block to the next, labeling the block number with B_j and the hash with H_j we have

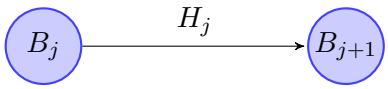

Figure 2: Blockchain showing only the flow of the hashes from one block to the next block in the sequence

We then set up many blockchains B_i, and label the jth block in the ith chain $B_{i,j}$ and the jth hash in the ith chain $H_{i,j}$

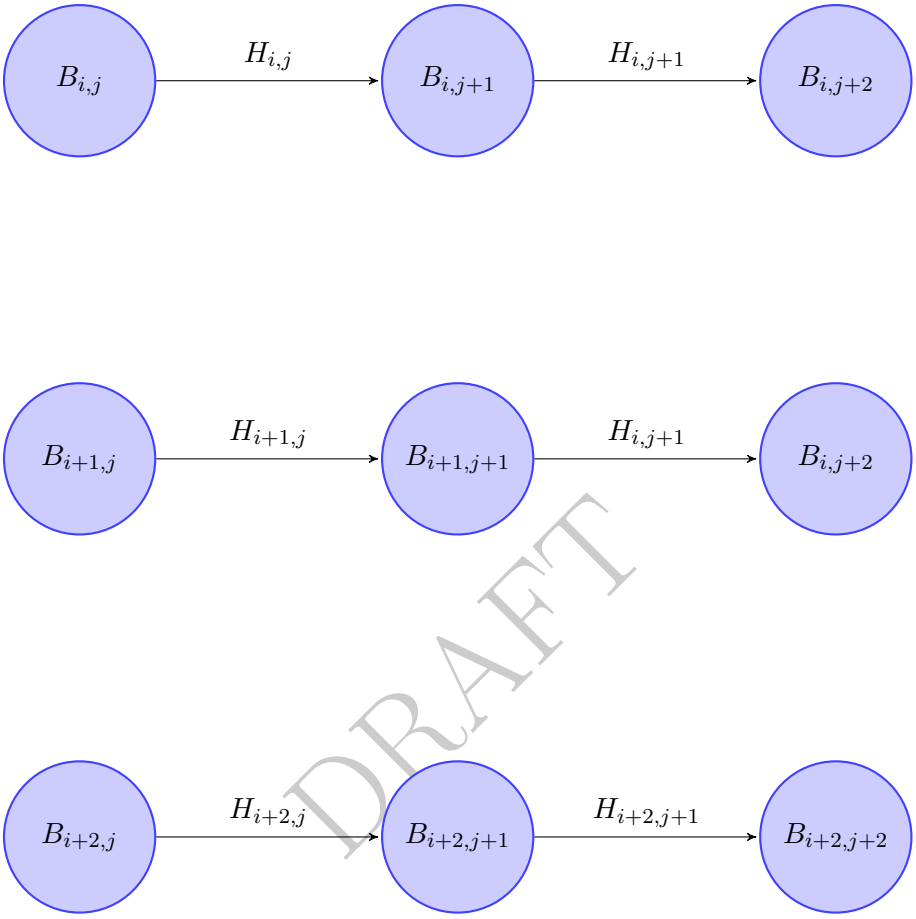

Figure 3: A set of three separate block chains

Then we arrange for each block to take not only on the hash from the previous block but also on the hashes from the previous blocks of its nearest neigbours. It then gets the Merkle root of these and uses that in place of the traditional previous block hash. This way the shards are tied together. Notice we've introduced the concept of "nearest neighbour", which implies a space of some sort.

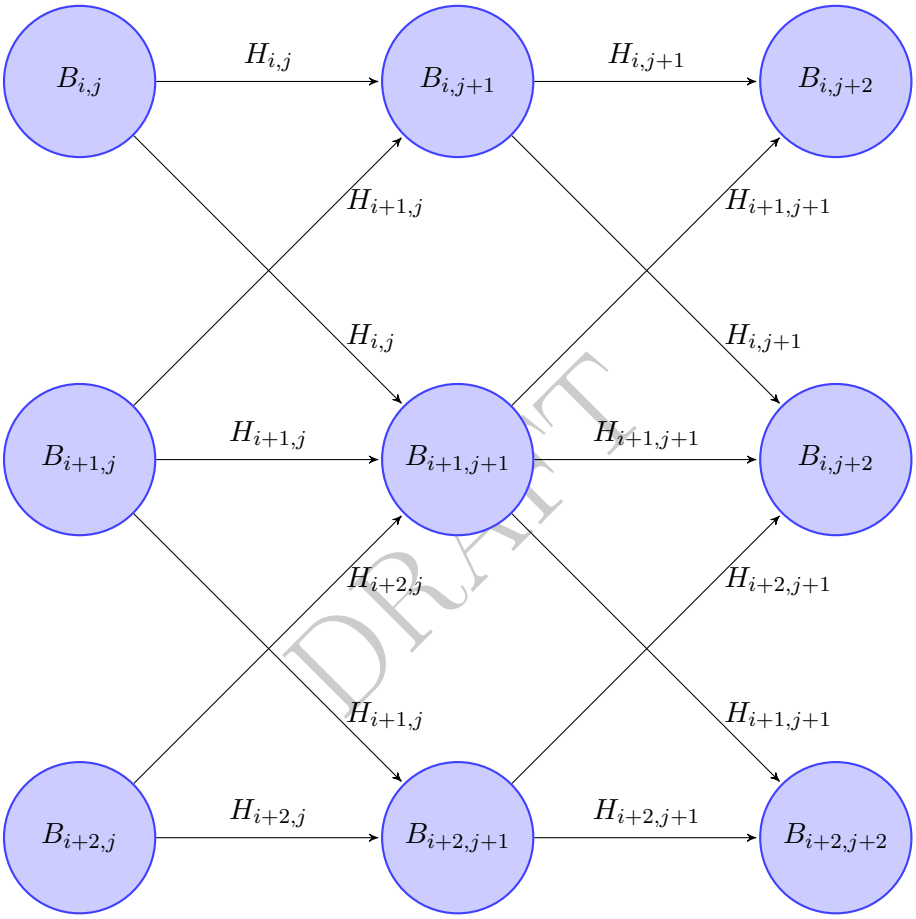

Figure 4: The chains are now interdependent and form a **block lattice**

2.3.0.1 Estimating the work done. With these modifications in place we can now estimate how the calculation of the total work done to produce a block is changed. In the case of a simple chain of blocks the work done (the difficulty) is simply the work done in this block plus the sum of the work done in all previous blocks. Let W_n be the total work accumulated so far to the nth block, let BW be additional work done to produce the new

block and $W_(n+1)$ be the new total work at the $n+1$th block then we have

$$W_{n+1} = W_i = n + BW \tag{1}$$

Without too much loss of generality we can make the simplfying assumption that a block lattice is infinite in extent (no edges). Considering an example where the shards are mapped to a simple 2 dimensional square lattice. Then each shard will have 8 neighbours. This means that each new block will accept the hash from 9 previous blocks (its own shard, plus 8 neighbours). That means the total work done to produce a block has to be summed over all 9 previous blocks. If we assume that producing a block requires 1 unit of work $BW = 1$ then equation (1) becomes

$$W_{N+1} = (9 \times W_n) + BW \tag{2}$$

The so that if $W_0 = 0$ the genesis block, then the first block after that $W_1 = 1$, and in the next block $W_2 = 9 + 1 = 10$ which results in this sequence

1	1
2	10
3	91
4	820
5	7381
6	66430
..	..
17	2084647712458321

After a few iterations the equation converges to

$$W_{n+1} = 9^n \tag{3}$$

which increases exponentially

This method of calculation is used in (Will Martino, Monica Quaintance, 2018) where they assign 'Merkle mass' to each layer and comment

> Known as Merkle mass..., it is the sum of the mass of the sequential layer intersections of the cone and it increases at a nearly exponential rate

But the above method is double counting. Another way to add up the work is to simply add up only the work added by the blocks a block depends on. This gets a different figure

1	1
2	$1 + 3^2 = 10$
3	$10 + 5^2 = 35$
4	$35 + 7^2 = 84$

Let n be the count of the number of the block height, d be the dimension of the lattice then we have

$$W = \sum_{1}^{n} (1 + 2n)^d \tag{4}$$

When $d = 2$ it increases only by adding the squares, since we're adding up the 'area' of each preceding layer. Which is sub-exponential.

The two methods seem to be adding up different things, and I'm not sure which is the correct calculation to use when calculating 'chain with most work'. Which is required for Proof of Work consensus.

The total entropy, i.e the number of possible combinations, is increasing exponentially and therefore in a block lattice it quickly becomes very hard to reverse out any records. Any change in a block in a shard propagates outwards and become incorporated into the hashes of more and more blocks in other shards until all shards are affected. The measure of the total work done is an indirect count of the number of states that have to be explored to find that partcular state. In a block lattice that number increases exponentially because each new block depends on blocks from more and more distant shards and the number of possible states rapidly increases.

2.4 Why is it called Indra's Pearls?

When deciding on a name for the project I thought it should be something to do with space, but other projects had taken those names e.g. Chainspace (Al-Bassam et al., 2017), the Latin word for space "Ether" is taken, the Sanskrit word for space "Akasha" is used by another crypto project. Then I remembered a nice description of a net of pearls where each one reflects all the others, which is what happens in a block lattice as described above.

> In the heaven of the great god Indra is said to be a vast and shimmering net, finer than a spider's web, stretching to the outermost reaches of space. Strung at each intersection of its diaphanous threads is a reflecting pearl. Since the net is infinite in extent, the pearls are infinite in number.
>
> In the glistening surface of each pearl are reflected all the other pearls, even those in the furthest corners of the heavens. In each reflection, again are reflected all the infinitely many other pearls, so that by this process, reflections of reflections continue without end.
>
> (Mumford et al., 2002, Indra's Pearls, Cambridge Universtity Press)

Which is a fairly good analogy of what happens in a block lattice, each shard in a block lattice is reflected in and is reflected by all the other shards. Hence the name *Indra's Pearls*

2.4.0.1 The consequences for finality. In (Nakamoto, 2008) to preserve the integrity of the blockchain each miner must check the validity of the previous block before the start mining the current block. Using the hash of the previous block in a current block is recognised as sign that the previous block has been checked and found valid. After approximately 6 rounds of blocks the accumulated work is regarded as being so great it would be impossible to redo and rewrite the history and the block is accepted as 'final'. In a blockchain the dependency on hashes only carries forwards in time, but in a block lattice the dependencies spread out through the lattice as well. The same rule applies, to use the hash of a preceding block in the same shard and in its neighbours, you have to have validated the block. In a simple block chain, after n rounds you have n confirmations. But in a block lattice each round propagates the hash into $(1+2n)^d$ blocks, where n is the number of rounds and d the number of dimensions in the lattice. In our simple 1D block lattice, after 1 step we have 3 confirmations and after 2 rounds $3+5=8$ confirmations. For a 2D lattice this becomes $3^2=9$

confirmations after the first round and $3^2 + 5^2 = 34$ confirmations after the second round.

In general the total number of confirmations C after n steps in a d dimensional lattice is

$$C = \sum_{1}^{n} (1 + 2n)^d \tag{5}$$

Which is the same equation as 4. The result is that a new block has to be validated by more miners or block producers than in a standard blockchain, and those block producers can be independent of each other. So the integrity of records is greater and we get finality more quickly.

2.4.0.2 Preventing 51% attacks. Any Proof of Work (POW) or Proof of Stake (POS) consensus system which allows anonymous mining or block production, it's possible for an anonymous malicious agent with sufficient computing power to rewrite the history. Re-writing the history allows double spending. For example they can branch off a new chain but keep it private, in the main chain they make a transaction where Alice sends tokens to Bob, while in their hidden chain Alice sends tokens to Cynthia. They wait for a while waiting for enough blocks to be produced on the main chain for the transaction to Bob to be confirmed and Bob to withdraw the tokens into fiat or another crypto. Once that's happened they publish their hidden chain and because they have more computing power than everyone else their newly published history has the greatest accumulated work, so it overrides the main chain and the transaction sending tokens to Bob is wiped out. This has happend in a number of smaller cryptocurrencies e.g. Verge and Ethereum Classic (Gareth Johnson, 2019; Reporter, 2018, see) But in a block lattice that's very hard to do because no one chain is isolated from the others. To overpower one shard you have to overpower its neighbours, and the neighbours of the neighbours and so on to the entire lattice.

(Nakamoto, 2008) includes a calculation of the probability that adversary creating a competing chain can catch up with the main chain. This calculation has to be modified to work with block lattices. The Kadema system (Quaintance, Io, and Martino, 2018) proposes a PoW model with linked shards, similar to Indra's Pearls. They've already extended this calculation to work with a shared system, the proofs are quite complicated but their summary is;-

> ...the proof is analogous to the probabilistic double-spend attack analysis in the Nakamoto Bitcoin paper (Nakamoto, 2008) expanded to the additional dimension of multple chains under the Chainweb protocol. **We present three versions of the**

proof in successive levels of strictness that demonstrate the viability of Chainweb as an extremely secure protocol. (my bolding)

Even so, a PoW system which permits anonymous mining can only ever achieve probabilistic finality. There is always a small but non-zero probability that someone will turn up having mined a complete chain which is longer and has more work. The critical thing is to limit the number of people capable of producing blocks, that way we can ensure all potential block producers are accounted for, and hence ensure that a block is final. This is discussed in more detail in 6

2.4.0.3 Problems with this approach. Because it's so hard to rewrite the history in a block lattice that causes a problem if a network split ever occurs. We think that network splits ought to be less frequent or even impossible because every block in a shard has to refer previous blocks in neighbouring shards, so the entire lattice is tightly tied together. But that argument is not a proof and it does need to be proved. If a network split happens and one section of the lattice gets two possible histories, then they'll be so tightly tangled up with the rest of the lattice that sorting it out will be impossible. This is an inherent problem with all consensus mechanisms that don't limit the number of potential block producers. There's always a non-zero probability that someone can turn up a sequence of blocks that has more work or more stake and hence require the history to be re-written. In a block lattice the probability of this happening is much lower than in a simple block chain, but it is still probabilistic finality. This is a problem with all chain based finality systems. The alternative is to use some variant of Byzantive Fault Tolerance consensus systems to assure instant finality. They have their own issues but by combing the chain (or lattice) based approach we can get the best of both. See 6

2.4.0.4 The topology of practical block lattices. Real space has no edges because it's (probably) infinite but that's impossible in a finite computing system. Yet we can't have edges since that means some nodes won't have the full complement of neighbours. The simplest thing use toroidal topology. For a lattice with 1 dimension of space, we create circle by joining the ends. In 2 dimensions we create a torus and so on to higher dimensions. Then we can give each shard a unique location on the torus by assigning an integer address and setting the size of each dimension to be a prime number. Then the location is defined by the modulus of the address according to the prime number used for each dimension. So for example an address of 9 in a 5×7 torus would be 4 along the 5 axis and 2 along the 7 axis. Under a Euclidean metric its closest neighbours have addresses $3, 8, 10, 15, 23, 24, 29, 30$

The Kadema team in (Will Martino, Monica Quaintance, 2018) use a construction based on a Peterson graph which provides maximal connections across a minimum span, which is the best solution if restricted to a flat global topology. It seems much easier just to use a toriodal global topology.

2.4.0.5 Increasing connectedness to enhance security. In (Quaintance, Io, and Martino, 2018), the Kadema team analysed cases where an attacker takes over progressively larger and larger sections of the lattice as cone of control marching forwards in time. However that proof assumes only local connections between shards. The security of a block lattice relies on its interconnectedness. A simple way to increase the connectedness of a graph is to add a few long distance connections. Therefore we should add one long range connection to another shard to be used along with the nearest neighbours when calculating the merkle root of the previous block hashes. There are many ways to do this, the long range connection could either be permanent or calculated on the fly. If it's fixed then the attacker would know in advance that they also have to overwhelm another shard and the shards that depend on its blocks in future. So we'll choose a dynamic method. The dynamic method will select another shard with deterministic randomness as follows.

```
gather the previous block hashes from self and neighbours
calculate the merkle root from the hashes
find the remainder modulo the size of the lattice
choose the shard with address == remainder
add that shard's block hash at the same height
recalculate the merkel root
```


That makes it impossible (in truth very very hard) for an attacker to take over any one section of a lattice. They have to take over the entire lattice. The Kadema proofs in (Quaintance et al., 2018) need to be extended to allow for long range connections, this is a research project.

Locating shards on a torus is the easy part, a more serious problem is that it takes time for information to travel from one shard to another and this causes an issue with state or account based systems.

3 UTXOs vs Account state

In all computer systems that have parallel processing there is a time delay moving data from one process to another which makes it hard to agree on the state of something 'now'. The state of an account in process A might be X, but another process B running in parallel might alter the state of the

account to Y, but process A only gets to find out about it after a delay. The state of something is the total of all past events that affect that thing summed up to 'now'. In a parallel system there is no universal 'now'. We have to introduce ways to create a fake 'now'. The usual way to do this is via locking schemes but the UTXO system avoids all this, it's inherently parallelisable since it doesn't maintain states of accounts

In a single blockchain this doesn't cause a problem because within each block there is a consistent 'now' so we can generate state which rolls up all past transactions and therefore have a state 'now'. That allows us to have smart contract code that says

```
if (balanceOf[address] > 200) {
 doSomething;
}
```

But that's not possible in sharded system because the balance of the *address* might be held in different shard to the smart contract. You can't roll up all past transactions, because you might not know about some in other shards

3.1 Relativity of simultaneity

The problem is most clearly seen if we rotate diagram 4 anti-clockwise 90°. It then matches the standard format for space time diagrams used in physics. In spacetime it takes time for signals to travel between points separated in space, in the same way it takes time for signals to travel from one computing process to another.

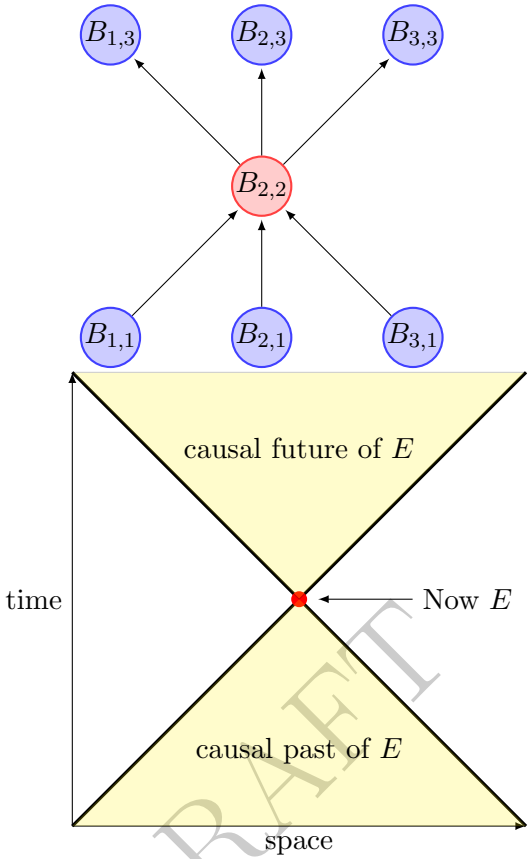

Figure 5: Propagation of hashes in a block lattice is analogous to a space time diagram

In special relativity there's a name for the problem. It's called ***Relativity of simultaneity*** Article, 2019 Now, this is critical for understanding the difference between stateful blockchains such as Ethereum and UTXO based systems such as Bitcoin.

In a state based system the totality of all past transactions come together to create the current state of an account, and blocks update the states of accounts. That's analogous to all the information from the past coming together into the central point of the above diagram 'Now'. State based systems rely on being able to know what the state of an account is at that moment of 'now', inside the block being generated. In spacetime there is no universal 'now', events that appear simultaneous to one observer might not be simultaneous for another observer. That's why it's called *The relativity of simultaneity*, which is the physicists way of saying "there is no universal

now". The analogous situation in a sharded blockchain is that there is no universal 'now' for all accounts because they might be in different shards so you can never write smart contract code like;-

```
if (balanceOf[address] > 200) {
    doSomething;
}
```

Because the account might not be in the same shard as the smart contract and therefore the value is unknown. Which neatly brings us to the next point.

3.2 Transactions are fundamental

Although this is only an analogy it does illustrate some useful features. Transactions in a UTXO system map to events in spacetime. The notion of accounts which hold state are abstractions from the history of past transactions and are therefore a synthesised 'now'. In a single blockchain it's relatively easy to manipulate states of accounts because the transactions happen in the same chain and the block being created defines a 'now'. So we can have a consistent concept of 'now' and therefore we can maintain a consistent abstraction of 'account state'. When transactions that affect the state of an account happen in many different shards it's very hard to maintain that abstraction, which is why Ethereum 2.0 requires re-engineering Ethereum to retrofit a transaction layer and has to add layers of complexity to re-synthesise 'now'

We adopt the approach that we should work with UTXOs not accounts as that simplyfies things enormously. In some ways each transaction is analogous to an event between sub-atomic particles which we can visualise with a Feynman diagram. The basic conservation laws, what goes in must come out, and the only things that can come out must have the same quantities as went in. Which is the basic logic of a UTXO transaction

3.2.0.1 Sharding in other account based systems. To comment adequately on each approach and dig out the problems would take us too far away from the intent of this paper. This has to be a separate paper. Suffice to say that each approach uses different and very complex methods to maintain a synthetic view of the state 'now'. The core fundamental issue is usually buried away in the small print and says effectively "Oh and by the way this extra layer holding the state assumes extemely fast connections between the nodes, and by the way it's always going to be out of date".

3.2.0.2 Synthesising 'now' in your brain. As an aside, your brain peforms a similar trick to synthesise the sense of 'now' from all the different events going on and messages it's getting from your senses, and has a similar delay problem. In the same way that a sharded account based system constructs a synthetic 'now' from all the different transactions going on in different shards and is behind the time, that sense of 'now' you experience is actually behind the time. What you experience as 'now' is actualy about half a second behind reality (refs to be provided)

4 An obvious extension to UTXO

Building a UTXO system from scratch gives us the freedom to extend the concept. Our general philosophy is make explicit what is implicit and then generalise. For example a Bitcoin transaction contains the implicit understanding that it is Bitcoin that is being transferred. In the set of UTXO records all values are assumed to be refer to the native token, e.g. Bitcoin.

Txn-hash	position	script to claim the output	value
txn-hash	5	<some script>	6

We make that explicit by giving the thing being transferred an id.

Txn-hash	position	script to claim the output	token_id	value
txn-hash	5	<some script>	<some token id>	6

4.1 Elegance in action

We've generalised the UTXO method to handle almost any number of different tokens. The basic logic of UTXO, inputs must equal outputs, also means we get these features for free.

1. The basic logic of UTXO means we can create transactions that exchange different tokens between parties.
2. It's a distributed exchange by default.
3. It treats fungible (ERC-20) and non-fungible tokens (ERC-721) in the same way.
4. No smart contract code is required to maintain the ledger for a token.

A simple design which accomplishes a lot is *Elegance in action*.

4.1.0.1 Prior art. The idea that the basic UTXO protocol of BTC, ZEC and similar needs to be extended to include more than the base value of whatever chain has a substantial history starting with the notion of 'colored coins', (Charlon, Flavien; Andreev, Oleg; Gundry, Devon; azuchi, Shigeyuki;Dorier, 2014) but that relied on using the existing UTXO structure and adding extra information into the unlocking script on the output via an OP_RETURN operation. That has problems because it's using the script to do something it's not intended for, and also unless carefully coded, the OP_RETURN data won't be transferred on to the next transaction, so a coin can easily lose its color.

Cardano (Jones, 2019) have proposed a similar idea for multiple currencies to be handled at the base UTXO layer in a future version of Cardano instead at a higher level in smart contracts. Their idea is a bit different because it requires a central store to track created token ids and they haven't actually implemented it yet, but we'll be looking and learning from what they're doing. The Nxt (Contributors, 2016) blockchain and its derivatives Waves (Platform, 2016) and Ardor (Staff Ardor, 2017) also implement multi-currencies at the base layer, but by adding attachments to the transactions rather than simply adding a token_id field.

4.2 Token creation

Allowing new token ids does however mean we must have a way to create new tokens in a way that does not allow them to be created more than once. An obvious way to do this is that the token id should depend in some way on the present or previous block hash in the block that creates the token. When a token is created we have to create the entire token supply at the same time. So we'll have a special type of transaction to create a new token id along with the full supply. The logic of UTXO, sum of outputs cannot be greater than the sum of inputs and because the input value is zero, the output value must also be zero. To keep things elegant we have to create a negative amount of a new token to balance out creating a positive amount. That means when we create tokens A_+ we also create anti-tokens A_-. Negative amounts are obviously useful when you want to burn a token, but it also has deep impications which need to be thought through. Nxt also creates an equal negative amount of its base token (Contributors, 2016)

Games developers have already reached the limit of what can be done with ERC-721 tokens. The token economy in games typically require tens of thousands of non-fungible tokens, which is hard to do in smart contracts. Exchanging the tokens isn't straight forwards either. Therefore some games developers have had to create a new standard ERC-1155 ((Radomski, 2019)) to deal with the problem. But in the Indra's Pearls system there is no prob-

lem. You can create as many tokens as there is space for unique identifiers in the token_id field, which is essentially infinite. Once created there is no need for a smart contract to manage account balances, the core logic of UTXO takes care of transfers.

4.3 Non-scalar values

The other thing implied in standard UTXO systems is that the quantity being exchanged is a positive scalar value. in the previous section we mentioned that token creation should generate an equal quantity of negative value to balance out the positive value. That can be useful when creating financial products. We can also extend the idea to allow other types of asset to be exchanged, duration and areas. This allows creating transactions that might be used to record the rent of a property for a length of time. This initial use case for tokens with richer data will be in the gaming world. It'll take decades before the law catches up with cryptocurrency world to allow blockchains to be used in legal contracts.

4.4 Rights

The last implication in UTXO that should be made explicit is that when an asset is transferred we're actually transferring the right to make onward transfers and that right has a value. There could be transactions for an asset that don't confer the right to onward transfer. And example might be the representation of an intangible asset such as reputation gained by completing a valid block. Reputation is a valuable asset, but it cannot be transferred. This comes in useful when building a consensus system that assigns a value to trustworthyness, see 6

4.5 Scripting language

Most smart contracts are written to maintain a central ledger for a token. E.g. All ERC-20 based ICO tokens. But since we can get most of the utility of fungible and non-fungible tokens, which is simple exchange of value, without recourse to smart contracts do we really need a Turing complete smart contract language? A domain specific, expressive, non-Turing complete scripting language would do for most purposes. The current proof of concept for Indra's Pearls is using a very cut down emulation of Bitcoin script. The next version will include a functional or declarative program based domain specific scripting language

Turing complete smart contract languages are great for programmers to play with, but they're dire for security and performance. Most use cases can actually be handled by non-Turing complete languages, which can be more secure becuase they're simpler. However the scripting language in Bitcoin is hard to read in it's native form, and it's hard to code complex cases. There is a growing demand for a scripting language that is more expressive than Bitcoin script, but not as expressive and dangerous as the EVM, EOS, Tron or NEM scripts. Waves has its own non-Turing complete scripting language RIDE (Begicheva and Smagin, 2018), both Kadema 'Pact' (Will Martino, Monica Quaintance, 2018) and Cardano 'Plutus' (Cardano, 2019) are using functional scripting languages because they're easier to read, and most importantly, easier to prove correctness through formal methods. In addition functional languages, in which functions take inputs and produce outputs map quite well to the UTXO model. However it's also worth considering the declarative paradigm as in most cases the rules we want to apply in the script can be reduced to a few simple declarations where the intent is obvious.

From this point onward in this white paper it is assumed that we will be working with a domain specific scripting language sufficiently expressive to be capable of handling the tasks required of it and not be Turing complete, although it's not present in the initial proof of concept it will be in the next proof of concept. One important feature this language will have is the capacity to express facts about the entire transaction the output is contained in, and also to express facts about the entire block the transaction is contained in. For example an unlock script for an output we want to be able to express that some fact about the entire transaction the output is in must be fullfilled, e.g. that there must be a certain other input or other output. Which neatly brings us to a method for creating a multi-party fair exchange protocol.

4.6 Multi-party fair exchange protocol

One of the features we can give an expressive scripting language is the capacity to check the features of the rest of the transaction the script is embeded in. It should be able to draw data from outside the script and access the entire transaction, and in some cases the entire block. That way we can write unlocking scripts that require the user to present a transaction in a certain form. So we can write script to claim the output that states in pseudocode;-

```
if this txn transfers X of token Y to address Z then
you can claim this output for token P
```

So we get a multi-party fair exchange protocol. This isn't the same as a coin swap between two parties, because anyone can claim tokens P. But notice that this is an inadvertent American style call option. If the price moves in the wrong way it's not worth making the transfer, if it moves in the right way, then the transaction can yield a profit. We'll extend this idea by creating a token which can be traded independently and is required to be presented to unlock the output

4.6.1 Example: Call Options

Because UTXOs don't have to be sent to accounts or addresses, they can be claimed by anyone that can provide a valid script, then you can write quite complex financial instruments. For example, we want to create a call option **C** that gives the owner the right to buy 100 units of token **A** for a price of 1000 units of token **B** (obviously at a certain time, but let's not make things too complicated). We create an initial supply of the call options **C**, and create transactions with output 100 units of token **A**. The verification script on output releasing token **A** requires you to exercise 1 call option **C**. The pseudocode might be written as;-

```
if this txn transfers 1000 of token B to address Z
and this txn has an input of 1 unit of token C then
you can claim the this output for token A
```


The problem here is that the transaction would have an input for the call option **C**, but no output for it, which under UTXO rules means the block producer or miner can claim it, and then use it to excercise the option again, or the owner of the call option could simply output it to an address they own and use it again. We need a way to terminate tokens. This is where negative values become useful, see 4.2. Following the rules of UTXO, inputs must be greater than or equal to outputs, that requires we generate an equal supply of negative values when we create positive value supply for a token. With negative values at our disposal the above script can be changed to;-

```
if this txn transfers 1000 of token B to address Z
and this txn has an input of 1 unit of token C
and this txn has an input of −1 unit of token C then
you can claim the this output for token A
```

The obvious thing is to create a transaction with two outputs, 100 units of the token **A** to be claimed, and -1 unit of the ancillary token for the call option **C**. That way the call option token **C** gets burned when it's exercised. Also, rather neatly, the supply of the call options **C** has to be backed by the right amount of the underlying **A**. Because it's a public block lattice,

anyone can verify that the call options **C** are backed by assets **A**. If we put a time lock into the unlocking rules so that the transaction is only valid between certain times then it becomes a European style option instead of an American style option. Using combinations of options it's possible to create almost any financial instrument. This becomes useful for block producers who need to hedge against price volatility in the underlying token used for rewards see 6.6

4.6.2 Merging and splitting tokens. FANIN & FANOUT

If we're going to use tokens to track physical objects then obviously we need a way to merge many tokens into one, and split one into many. For example, a car is made of many components, each can be tracked through the supply chain with a token. But we never buy a box of bits, we buy a whole car. Therefore we need a way to take a collection of tokens and replace them with one token, a FANIN operation. The that token needs to be able to reverse the FANIN operaion, and perform a FANOUT. Then we can use the single token to recover all the tokens that went into the FANIN.

Similar to the way we've done a call option above 4.6.1, we create a new token A_+ along with its anti-token A_-, and then we create a transaction from all the other tokens B_n we want to merge together, the outputs for each B_n have unlock scripts that only allow it to be claimed in a transaction which unlocks all the other B_n and includes the inputs A_+ and A_-

This allows us to take a box of bits, each represented by a token, and replace that with a single token which can be transferred to other parties as a single unit. Then when the need arises, we can use that token to recover the tokens for each item in the box of bits, burning the single token in the process. The Indra's Pearls system is therefore a good fit for supply chain tokenomics, and also for security token exchanges because we can deal with corporate actions such as splitting and combining securities.

4.7 Principle token for rewards 'IPRL'

In BTX, ZEC, ETH, EOS, new values are created out of nowhere to distribute to the people creating blocks and therefore incentivize them. But it seems inelegant to have a different creation rule for Indra's Pearls base token,which we'll call IPRL. Therefore for all tokens to follow the same creation rule, we'd have to create the total supply of the IPRL once and then distribute it, which also means creating a balancing negative supply. This is the approach taken in IOTA (Popov, 2017) and also Nxt (Contributors,

2016). There are advantages and disadvantages to this. The advantage is that everyone can see the total supply, the disadvantage is that the total supply has to be divided up into smaller units for distribution to block producers. This actually turns out to be an advantage when looking at dynamic block rewards, see 6.4

5 Inter-shard transfers

Obviously the logic of inter-shard transfers must not allow double spending. That means when we use an output in one shard in an input in a transaction in another shard we have to remove that output from the UTXO set in the originating shard. The other restriction is that we can only use outputs that have been finalised. If we use an output in one shard 1 in an input in shard 9, and shard 1 gets reversed, then we're in deep trouble. As explained above in a block lattice we can reach finality very quickly. Even one round on a 2-D lattice gets 9 confirmations. (actually 10 if you include the long range connection).

5.1 One possible method

Taking the view that "*perfection is the enemy of done*", in the proof of concept I've adopted the following design.

- Each transaction contains the address of the shard it happens in.
- The shard address is included in the data forming the hash of the transaction.
- Each output also has a shard address, but it can be different from the transaction's shard address.
- Each output has it's own hash, including the shard address.
- The output hash must be unique within a shard.
- Each shard only lists the UTXO set for outputs having its shard address.
- Each transaction includes a merkle root for all its output hashes. That merkle root is used in the data forming the transaction hash. That way we can verify a output is present in a transaction, even if we don't have all the outputs. This is required when we copy the transaction to another shard. We only copy the outputs destined to the target shard and no inputs.

- All inputs to a transaction can only use outputs in the same shard as the transaction.
- We have a special 'transfer' transaction that is a copy of the originating transaction but only including the outputs valid in the destination shard. The validation conditions on that transaction have to be different, it's only necessary to validate via merkle proofs that the outputs were includes in the transaction. This can be done either by querying the originating shard, or copying the merkle proof over to the destination shard. We also validate that the transaction is in a block in the originating shard.

With that basic design along with the basic logic of UTXO we get this behaviour

- To send an output to another shard, we create a transaction that set the shard address of the output to the destination shard.
- Because it has a different shard address the output is not listed in the pool of UTx0s in that shard. So it can't be spent in that shard.
- Because all inputs must use only outputs destined for the same shard as the transaction the input is in, then an output destined for another shard can't be spent in the originating shard.
- Because output hashes must be unique, only one can exist at any one time in a shard's database. so you can't transfer it twice.
- When we transfer a reduced copy of the transaction from the originating shard to the desination shard, then the output will have the right shard address to be used by inputs in transaction in that shard.

These are the steps to transfer an output from shard 1 to shard 2

1. Create a transaction in shard 1 with an output having shard address 2
2. Create a special transaction in shard 2 that is a copy of the transaction in shard 1 which only contains outputs with shard address 2
3. Block producers for shard 2 query shard 1 to get the merkle proofs that verify the transaction exists in a block in shard 1 and the outputs exists in that transaction. Once verified they include it in a block.
4. Once the special transfer transaction is included in a block then the outputs transferred are available in the UTXO set for that shard.
5. Transactions in shard 2 can now create inputs using the transferred output.

Even though it's not perfect this seems a good solution. And it's working in the proof of concept.

Unavoidable delays All sharding systems require a number of blocks to transfer value from one shard to another. This reduces the actual through put in a sharded blockchain. In addition, and in common with all parallelised processes, there comes a point where the extra work done sending messages between components starts to degrade performance as you add more nodes. We need to run tests to find out how this design works in practice.

6 Block validation & finality

In a sharded system finality is critically important. If a transaction in a block in a shard transfers value to another shard, and then the block is replaced, then we've got a big problem, because it's hard to undo. But we can flip that around, if information in a block propagates out to other shards then it's very hard to replace that block since it is now tied to blocks in other shards. That's the whole point of having neighbouring shards exchanging block hashes. But we also want a level of finality at the block level to be sure that the leading block is valid before neighbouring shards pick up its block hash.

Typically in a POS or DPOS system finality is due to an informal social contract such that clients accept that a block is final if the validators say it's final. However it is physically possible for validators and block producers to collaborate and reverse blocks after they've been published. The EOS system estimates finality takes as long as 180 seconds. Which means a transaction can be reversed before that time. Because the energy costs are low in POS/DPOS systems it is physically possible for block producers and validators to reverse blocks after that time. However in the Indra's Pearls system it's nearly impossible to reverse a transaction once a block has been picked up by neighbouring shards and the block hash incorporated in their next block. Once a block is broadcast and picked up that's it, it's final unless the block producers and validators collaborate with the block producers and validators in neighbouring block. which in a simple 2-D square lattice means collaborating with all the BP and validators in 8 additional shards.

Typically non-anonymous POS and DPOS systems use Byzantine Fault Tolerance (Lamport et al., 1982) based schemes to find consensus amongst a predeterimed group of block producers. A review of the key differences between approaches can be found here (Larimer, 2018), which also includes an explanation of the pipelined BFT fault tolerance system used in EOS. The system described cannot be used as it is because it requires proposing

block producers in advance and if they don't produce a block then you lose time finding a replacement. This cannot be allowed to happen in the Indra's Pearls system since neighbouring shards will be waiting for the block so they can proceed. Therefore we have to have a system that post selects producers after a block has already been produced, so they all have to produce a block, and we won't have a problem with dropouts. In addition there are clearly defined rules for checking the validity of a block, what we really want to know is if anyone can produce a proof that the block is invalid according to the published rules. If 99 out of 100 producers agree that a block is valid and 1 can produce a proof that it's invalid, then the block is invalid.

We'll explain the mechanism in these steps

1. How we calculate 'stake' 6.1
2. How 'stake' is posted 6.2
3. How 'stake' is used to select who produces a block 6.3
4. How a block reward is calculated and how it's claimed 6.4
5. How bad actors are penalised 6.5

6.1 How we calculate 'stake'

We need to allocate rights to produce blocks according to 'stake' and 'stake' has more than one dimension. 'Stake' is a proxy for honesty, and different facets of stake are more or less useful as proxies for honesty. The simplest form of stake is just money in the native token of the system, EOS, ETH and so on. But what we're really interested in is honesty and there's an inverse correlation between wealth and honesty (Piff et al., 2012) so Proof of Stake based purely on staking money is a very bad proxy for honesty. In the DPOS method a block producer has to acquire votes, and votes are according to the number of tokens staked, so producers can vote for themselves, but more worryingly we're effectively crowd sourcing due diligence on block producers. So that's not a good guide to honesty either. Although in practice it has worked so far. In theory block producers should contribute to the community to show that they're all round good guys and get votes that way, anyone caught buying votes gets penalised. But that's rather vague. However if someone has been producing blocks without errors for thousands of blocks that's concrete evidence of honesty. Who would you trust more? (a) someone with a stake of $1,000 who's been producing blocks without an error for a year, or (b) someone who arrived yesterday and put up a stake of $1,000,000 or just 'acquired' a million votes. Trust has to be earned over time, it can't be bought.

Therefore we'll factor 'stake' into these components which are fairly independent of each other.

1. The trust I place in myself by putting up my own money **S**
2. The trust other people place in me by voting for me **V**
3. The trust I've earned through consistent honest behavior. Reliability **R**

'Stake' is composed of all these things, in other systems they're not made explicit, we're making them explicit here. Some types of stake you can lose, **S** and **R** you will lose if you produce a bad block. For **V** It's up to other people to decide if they're going to carry on voting for you if you lose your credibility by creating a bad block.

When viewing stake in this way we are in effect asking a question about how much information do each of these components convey about the thing we're really interested in;- honesty. What's the information content of each component of the stake? Since we're interested in the amount of information conveyed rather than the absolute value then that immediately suggests we need to use Shanon Information Theory (Shannon, 1948). In Information Theory the information content of a message is the log of the inverse probability of receiving that message.

$$S = -\ln(p) \tag{6}$$

Where p is the probability of getting that message. Infrequent messages convey more information. This is immediately applicable to **R** in the above list. If someone has been producing blocks without error for a year, then one more honest block doesn't convey much information, the evidence of honesty is the log of (R, not **R** itself. Since we will allowing the issue of non-transferable tokens, we issue one non transferable token for each completed block and then take the log of the total to be a measure of reliability. The same applies to **S** and **V**. The information content of someone increasing their pesonal stake **S** from of IPRL 1,000 to IPRL 2,000 is much greater than if someone increases it from IPRL 1,000,000 to IPRL 1,001,000. We'll call the total 'stake', **K** for 'Kudos'. **K** is then

$$K = a\ln(S) + b\ln(V) + c\ln(R) \tag{7}$$

Where a,b,c are contants to be decided. The criteria for defining the values of a,b,c are that a has to be greater than b so that it's better for a producer to stake their own money rather than use their tokens to vote for themselves, and c has to be really important.

The result is to give block producers with smaller stakes a greater probability of being selected to produce a block than if we took the raw values. E.g. a personal stake **S** of IPRL 1,000,000 makes it only twice as likely that you'll be selected than someone who puts up **S** of IPRL 1,000 , not 1,000 times more likely. However this creates a problem with the returns on **S**. Producers would be getting less than a fair return on their personal stake for larger amounts of stake, the result would be to disincentivise producers from staking and the best rate of return on stake would be the lowest possible amount, see 6.4 for a way to deal with this.

6.2 How stake is posted

To produce a block the producer has to be elected via voting, similar to the EOS DPOS system, then they have to post their stake composed of the three dimensions **S**, **V**, **R** as a transaction with those inputs. The outputs will have unlocking scripte that will allow them to reclaim their stake after a certain period of time, but in addition the script will allow anyone to claim the **S** portion of their stake and wipe out the **R** portion, if and only if they can produce a proof that the producer has produced a block containing an error according to the published rules, see 6.5 below.

6.3 How 'stake' is used to select who produces a block

The reason for post selecting block producers is that if a block producer is pre-selected to produce a block in a particular time slot, and then don't, then producers in other shards will be waiting for that block so they can incorporate the header hash into their own block. It takes time to switch to an active producer and that cannot be allowed to happen. So all block producers have to produce a candidate block which is then post selected according to their stake. (Another potential reason higlighted by Kadena in (Will Martino, Monica Quaintance, 2018) is that the US laws regarding money transmission agents (MTAs) do not classify cryptocurrency miners as MTAs due to the random nature of allocating a block producer. But that's their reading of the law).

A simple way to post select BPs biased according to their stake is to collect together the block hashes for the next block from all producers who generate a block before a cut off time. Put the hashes into a merkle tree and find the root. Since the hashes are numbers we automatically have a distance function, and so each block hash has a distance from the merkle root hash. Since the hashes are generatred by a cryptographic hash function, the distances will have a uniform distribution. We bias the distance

by dividing it by the staked 'Kudos', **K**. We then take the block with the shortest biased distance from the merkle root of all the blocks. That is both random but biased towards BPs that stake the largest **K**.

However such a method creates an incentive to BPs to be the last to announce their block. If they can see all the other block hashes, the last BP can attempt to adjust their block hash to be closest to the merkel root hash of all blocks. However that will take time, and they'll end up missing the cut off time. Another solution would be to add an extra source of randomness which is inaccesible to all BPs at the moment the block is being produced, but completely transparent after it's been produced. E.g. the merkle root of all neigbouring block hashes at the same block height. But that would mean a two step selection process, such that a block can only be published after a delay to get the neighbouring block hashes at the same height, selecting the BP, and then doing a BFT round to accept it. This is similar to the pipelined BFT processes described in (Larimer, 2018). It would mean that each block has a dependency on the neighbouring blocks two steps back instead of one.

6.4 Reward incentives

Creating the native token IPRL will follow the same rules for creation as all other tokens see 4.7. That means the token id and total supply is created only once. It then has to be gradually distributed in the form of block rewards to BPs over a period of decades. Similar in some ways to IOTA (Popov, 2017)). With a sufficiently expressive scripting language we can create unlocking scripts on outputs that can only be unlocked by a block producer. E.g. the signature on the transaction that claims the reward has to be from the same private key as the signnature on the whole block. That means we can divide up the total supply of IPRL into denominations that can be redemed by block producers. But we don't want to do that too far in advance because the UTXO set will be filled with UTXOs destined to be redemed by BPs in decades to come. The best way is to divide up the initial supply into a small number of large denominations, which are futher subdivided by agreement between all BPs into smaller denominations closer to the time when they're to be distributed. This also means we can divide up the supply into different denominations which can be claimed according to the size of staked **S**.

Since the probability of becoming a block producer depends on **K** which has a term dependent of the log of **S** there's a strong disincentive to stake money. The more you put up the less return you get per IPRL staked. Someone staking IPRL 1,000,000 would be only twice as likely to get selected as

a producer than someone staking IPRL 1,000, so their return from staking their own IPRL would be 1/500th per IPRL compared with putting up IPRL 1,000. Not a very good deal. To balance things out we have to allow BPs with a greater amount of **S** to claim larger denomination outputs of the initial distribution of IPRL. So the unlocking scripts for the outputs have to take into account the amount of **S** a BP has. A starting suggestion would be that a producer could claim reward outputs with a value calculated like this.

$$ClaimValue \propto \frac{S}{\ln(S)} \tag{8}$$

That means someone staking IPRL 1,000,000 would be only twice as likely to produce a block as someone staking IPRL 1,000, but be allowed to claim 500 times more IPRL. With **V** and **R** only contributing to **K**, not to the value a BP is allowed to claim. The result is that smaller producers don't get squeezed out of being able to produce blocks, and large producers will still get a fair return on their stake.

There is a fear that systems like POS and DPOS tend to a plutocracy where whales dominate block production. Though this is not born out by facts see 6.7. However this method of allocating block producers reduces the ability of whales to influence voting and block production but at the same time allows them to get a fair return on their investment.

6.5 Penalties

A block is either valid or not according to published rules which can be evaluated by a deterministic alogorithm. Because transaction outputs can be claimed by anyone who presents a valid unlocking script then we can create a locking script that locks up outputs posted for staking in such a way that they can be re-claimed either by the block producer at later point in time, or by anyone who can present a proof that the producer has created a bad block. The fact that anyone can claim someone's stake if they can produce a proof that they've created a bad block is strong incentive to be honest, and also a strong incentive to check for evidence of dishonesty in other people.

Someone presenting a proof of dishonesty would be allowed to claim all of the **S** staked, and be required to wipe out the **R** staked by including a negative **R** input (therefore negatives do come in useful). The **V** component would be left alone for other token holders to decide if they want to keep on voting for a BP that has produced a bad block

It would also be possible to penalize producers who use the hash of a bad block when they incorporate it into their own blocks. E.g. the next

block in the sequence on the same shard, and neigbouring shards. Since it only requires one person to produce a proof of dishonesty and get that proof into a block, it would be very hard for a group of producers to collaborate and produce dishonest blocks.

6.6 Reward stability

One of the problems faced by miners and block producers is the volatility of cryptocurrency prices. They have constant expenses and wildly varying income. It has been suggested that EOS block producers should get increased rewards when the price of EOS relative to fiat currencies is low, and reduced rewards when it is high. This obviously has a problem that it increases inflation when the EOS price is low, and decreases inflation when it's high, which might create a feedback loop. There's also the problem of deciding which oracles for prices to use and embedding them into the protocol. In Indra's Pearls it's easy to create options, and by a combinations of options other financial instruments can be created which can be used for hedging. Combine this with tokens that can be exchanged via atomic swaps with tethered cryptocurrencies and it is possible for block producers to shift the risk due to volatile prices to people who want that risk.

The combination of a fairly predictable return on investment in IPRL due to 6 and 7 together with the ability to use options and atomic swaps into tethered cryptocurrencies will ensure that being a block producer for Indra's Pearls is a stable and viable business

6.7 Centralised or Decentralised?

In theory BTC, ZEC, ETH and so on, are decentralised because anyone can join in mining, but in practice they're highly centralised because economies of scale and volatility of returns benefit large miners and pools. In theory EOS is centralised, but in practice more decentralised because the block producers share the chance of creating blocks more evenly. Up to date data on hashrate distribution is available at Staff Blockchain.com, 2019 and on EOS producers rank at (Eosauthority, 2019). It's easy to see that BTC has 8 producers with more than 2% of the hashrate, with just 4 entities controlling nearly 57% of the hashrate. But in EOS the largest block producer has only 1.99% of the votes, and 40 out of the 100 listed have between 1% and 2% of the votes. The data show that EOS is more decentralised than BTC.

We will adopt the EOS system which restricts the number of block producers for each shard, because this is essential for fast finality. There is an argument that POS/DPOS leads to plutocracy, but our use of the logarithm of stake rather than the raw value of stake benefits smaller producers.

Also in a sharded system, each shard needs a quorum of block producers. Assuming each shard needs 21 block producers as in EOS, then with only 15 shards there are seats at the table for 315 block producers. Initially it's likely that we won't have enough block producers. Correctly balancing the constants a, b, c in 6 could encourage block producers to produce on more than one shard. For example someone who's earned 1000 units of reliability **R** would be better off producing on 10 shards risking 100 units of **R** on each one because we take the log of **R**.

Even though each shard has a restricted number of block producers would appear to make Indra's Pearls very centralised, the end result should be the most highly decentralised cryptocurrency of all because there are lots of shards and our method of allocating block producers means smaller block producers are not squeezed out by the whales.

6.8 Fixing the problem of selfish collaboration

A deep issue with BFT consensus is possible selfish collaboration between BPs to undermine the whole system. This problem is described in an article on selfish collaboration in the Lisk system (Gunther, 2018). The technical approaches to minimize this are many and varied, this article lists 72 different approaches to blockchain consensus (Walter, 2019). While there are many ways to ameliorate the problem, and our approach of using logarithms of stake and votes certainly helps, it is ultimately a political problem. If block producers judge that their short term self interest can be advanced by collaborating in ways that damage the overall health of the community around a crypto project they'll find a way to do it. Therefore the best way to ensure that all block producers work towards maintaining the growth of the crypto project they're a part of is to have a clear direction, properly articulated, so that people can see they're better off collaborating towards a goal that benefits everyone.

We see Indra's Pearls as like an ASIC for the token economy. It's perfect for tokenising assets. But to do that in the real world requires legal structures to catch up, which will take decades. Blockchain token economies are perfect for transnational communities that want to track exchanges of value and there are already a set of such communities in the world of gaming. Using Indra's Pearls in the world of games and virtual reality will allow us to perfect token economics prior to national laws being passed that permit transnational token economics in the real world.

Therefore we will be actively promoting Indra's Pearls in the games com-

munity with the goal of becoming a significant player in that sector. Because most of the value in a blockchain is in the community around it not in the technology itself we will dedicate a large part of our effort in community building. Laying out a clear goal and viable steps towards that goal, and in particular community building, will prevent selfish collaborations developing.

7 Resarch and Development stages

Presently the Indra's Pearls proof of concept (POC) has been developed in Ruby with Postgresl backend for rapid prototyping and excellent testing facilities. It's been taken as far as running with 15 shards in a 3×5 toroidal topology, exchanging value between shards. The significant Items to be completed are coding the exact method of token creation, but that depends on BPs being able to redeem IPRL tokens, which in turn depends on developing a sufficiently expressive scripting language. A temporary stop gap to get things moving is simply to allow IPRL tokens to be created as they are in BTC, ZEC and so on. This will be removed in the next proof of concept. Since the consensus mechanism described above also requires the domain specific scripting languarge that also has a temporary stop gap solution which will be replaced with the full mechanism in the next proof of concept.

R&D in POC 2.0 will therefore include ;-

1. Backend running in Elixir/Erlang.
2. First iteration of expressive non-Turing complete scripting language.
3. Multi token creation as described above.
4. Research and testinging the consensus mechanism as described above
5. First public testnet
6. First public block producer service code published on Github
7. First iteration of a wallet
8. Published APIs and Swagger documentation

POC 3.0 will add

1. A better API to enable people to make use of Indra's Pearls as a distributed exchange

2. An easy way to interact with the system so that users, e.g. games developers, can create fungible and non-fungible tokens with minimal effort.

3. First iteration of non-scalar tokens, e.g. geographic areas.

4. A comprehensive reference test suite to facilitate anyone wanting to write their own implemtation in Golang, Rust, C, C++, Python, Scala, etc

8 Conclusion

Indra's Pearls combines these features into one package

1. Sharding for scalability

2. UTXO system with multIple tokens at the base protocol layer that do not require smart contracts. An ASIC for token economics

3. Rapid finality via hash propagation and a modified DPOS protocol.

4. It's a distributed exchange by default.

5. An expressive, non-Turing complete script which allows multi party fair exchange transactions to be created.

6. That also allows financial instruments to be created, e.g. Put and Call options, and all combinations from that.

7. An effective way to allow smaller block produces to participate, while not overly restraining the return on investment of larger block producers.

8. By using puts, calls and so on coupled with atomic swaps into tethered cryptocurrencies unwanted volatility for block producers is reduced.

References

Mustafa Al-Bassam, Alberto Sonnino, Shehar Bano, Dave Hrycyszyn, and George Danezis. Chainspace: A Sharded Smart Contracts Platform. arXiv, aug 2017. URL http://arxiv.org/abs/1708.03778.

Wikepedia article. Relativity of Simultaneity, 2019. URL https://en.wikipedia.org/wiki/Relativity{_}of{_}simultaneity.

A Begicheva and I Smagin. RIDE: a Smart Contract Language for Waves. Technical report, Waves, 2018. URL https://wavesplatform.com/files/docs/white{_}paper{_}waves{_}smart{_}contracts.pdf.

Cardano. Introduction - Cardano, 2019. URL https://cardanodocs.com/technical/plutus/introduction/.

Nicolas Charlon, Flavien; Andreev, Oleg; Gundry, Devon; azuchi, Shigeyuki;Dorier. Open Assets Protocol, 2014. URL https://github.com/OpenAssets/open-assets-protocol.

Many Contributors. Whitepaper:Nxt, 2016. URL https://nxtwiki.org/wiki/Whitepaper:Nxt.

Staff Eosauthority. Block producers ranking - real time statistics, 2019. URL https://eosauthority.com/producers{_}rank.

Gareth Johnson. Ethereum Classic 51% Attack — The Reality of Proof-of-Work, 2019. URL https://cointelegraph.com/news/ethereum-classic-51-attack-the-reality-of-proof-of-work.

Simon Gunther. Lisk — the mafia blockchain – Coinmonks – Medium, 2018. URL https://medium.com/coinmonks/lisk-the-mafia-blockchain-47248915ae2f.

Michael Peyton Jones. Multi-currency on the UTXO Ledger, 2019. URL https://github.com/input-output-hk/plutus/blob/master/docs/multi-currency/multi-currency.md.

Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine Generals Problem. ACM Trans. Program. Lang. Syst., 4(3):382–401, jul 1982. ISSN 0164-0925. doi: 10.1145/357172.357176. URL http://doi.acm.org/10.1145/357172.357176.

Daniel Larimer. DPOS BFT— Pipelined Byzantine Fault Tolerance – eosio – Medium, 2018. URL https://medium.com/eosio/dpos-bft-pipelined-byzantine-fault-tolerance-8a0634a270ba.

Marc-André Bélanger. Mt. Gox : Failure and opportunity, 2014. URL https://web.archive.org/web/20140610064048/http://www.belmarca.com/2014/02/26/mt-gox-failure-and-opportunity/.

David Mumford, Caroline Series, and David Wright. Indra's Pearls. Cambridge University Press, Cambridge, 2002. ISBN 9781107050051. doi: 10.1017/CBO9781107050051. URL http://ebooks.cambridge.org/ref/id/CBO9781107050051.

Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. Www.Bitcoin.Org, 2008. ISSN 09254560. doi: 10.1007/s10838-008-9062-0. URL https://bitcoin.org/bitcoin.pdf.

Paul K. Piff, Daniel M. Stancato, Stéphane Côté, Rodolfo Mendoza-Denton, and Dacher Keltner. Higher social class predicts increased unethical behavior. Proceedings of the National Academy of Sciences, 109(11):4086–4091, mar 2012. ISSN 0027-8424. doi: 10.1073/PNAS.1118373109. URL https://www.pnas.org/content/109/11/4086.

Waves Platform. WAVES whitepaper. – Waves Platform, 2016. URL https://blog.wavesplatform.com/waves-whitepaper-164dd6ca6a23.

Serguei Popov. The Tangle. 2017. URL https://iota.org/IOTA{_}Whitepaper.pdf.

Monica Quaintance, Monica@kadena Io, and Will Martino. Chainweb Protocol Security Calculations. Technical report, Kadena, 2018. URL http://kadena2.novadesign.io/wp-content/uploads/2018/08/chainweb-calculations.pdf.

Witek Radomski. ERC-1155: The Crypto Item Standard – Enjin, 2019. URL https://blog.enjincoin.io/erc-1155-the-crypto-item-standard-ac9cf1c5a226.

James Ray and Vitalik Buterin. On sharding blockchains, 2019. URL https://github.com/ethereum/wiki/wiki/Sharding-FAQs{#}what-are-some-trivial-but-flawed-ways-of-solving-the-problem.

Staff reporter. Verge Is Forced to Fork After Suffering a 51% Attack - Bitcoin News, 2018. URL https://news.bitcoin.com/verge-is-forced-to-fork-after-suffering-a-51-attack/.

C E Shannon. A Mathematical Theory of Communication. Technical report, Bell Labs, 1948. URL http://math.harvard.edu/{~}ctm/home/text/others/shannon/entropy/entropy.pdf.

staff Ardor. Ardor Whitepaper. Technical report, Jerulida Inc, 2017. URL https://www.jelurida.com/sites/default/files/JeluridaWhitepaper.pdf.

staff Blockchain.com. BTC hashrate distribution, 2019. URL https://www.blockchain.com/en/pools.

Cedric Walter. Blockchain Consensus Encyclopedia - More than 72 Blockchain Consensus described., 2019. URL https://tokens-economy.gitbook.io/consensus/{#}more-than-72-blockchain-consensus-described.

Wikipedia. Metric spaces, 2009. URL https://en.wikipedia.org/wiki/Metric{_}space.

Stuart Popejoy Will Martino, Monica Quaintance. Chainweb: A Proof-of-Work Parallel-Chain Architecture for Massive Throughput. Technical report, Kadena, 2018. URL http://kadena2.novadesign.io/wp-content/uploads/2018/08/chainweb-v15.pdf.