WHY MANY SMART CONTRACT USE CASES ARE SIMPLY IMPOSSIBLE.

Gideon Greenspan

As the developer of a popular blockchain platform, I sometimes gets asked whether Ethereum-like smart contracts are on the MultiChain roadmap. The answer I always give is always: 'No, or at least not yet'.

But in the hype-filled world of blockchains, smart contracts are all the rage, so why ever not? Well, the problem is, while we now know of three strong use cases for permissioned bitcoin-style blockchains (provenance, company recordkeeping and lightweight finance), we've yet to find the equivalent for Ethereum smart contracts.

It's not that people don't understand what they want smart contracts to do. Rather, it's that so many of these ideas are simply impossible. When smart people hear the term "smart contracts", their imaginations tend to run wild. They conjure up dreams of autonomous intelligent software, going off into the world, taking data along for the ride. Unfortunately, the reality of smart contracts is more mundane.

A smart contract is a piece of code that is stored on an blockchain, triggered by blockchain transactions and which reads and writes data in that blockchain's database. That's it. Really.

A smart contract is just a fancy name for code that runs on a blockchain, and interacts with that blockchain's state. And what is the code? It's Pascal, it's Python, it's PHP. It's Java, it's Fortran, it's C++. If we're talking databases, it's stored procedures written in an extension of SQL.

All of these languages are fundamentally equivalent, solving the same sorts of problems in the same sorts of ways. Of course, each has its strengths and weaknesses – you'd be crazy to build a website in C or compress HD video in Ruby. But in principle at least, you could if you wanted to. You'd just pay a heavy price in terms of convenience, performance, and quite probably, your hair.

The problem with smart contracts isn't just that people's expectations are overblown, it's that these expectations are leading many to spend time and money on ideas that cannot possibly be implemented.

It seems large companies have sufficient resources to travel a lengthy path – from the moment when senior management encounters a new technology, to when that technology's advantages and limitations are truly understood. Perhaps our own experience can help shorten this time.

Over the past nine months, we've been pitched many smart contract use cases, and have found ourselves responding, time and again, that they simply cannot be done.

As a result, we've identified the three smart contract misconceptions that are most commonly held. These ideas aren't wrong because the technology is immature, or the tools are not yet available.

Rather, they misunderstand the fundamental properties of code which lives in a database and runs in a decentralized way.

1. CONTACTING EXTERNAL SERVICES.

Often, the first use case proposed is a smart contract that changes its behavior in response to some external event. For example, an agricultural insurance policy which pays out conditionally based on the quantity of rainfall in a given month.

The imagined process goes something like this: The smart contract waits until the predetermined time, retrieves the weather report from an external service and behaves appropriately based on the data received.

This all sounds simple enough, but it's also impossible. Why? Because a blockchain is a consensus-based system, meaning that it only works if every node reaches an identical state after processing every transaction and block.

Everything that takes place on a blockchain must be completely deterministic, with no possible way for differences to creep in. The moment that two honest nodes disagree about the chain's state, the entire system becomes worthless.

Now, recall that smart contracts are executed independently by every node on a chain. Therefore, if a smart contract retrieves some information from an external source, this retrieval is performed repeatedly and separately by each node. But because this source is outside of the blockchain, there is no guarantee that every node will receive the same answer.

Perhaps the source will change its response in the time between requests from different nodes, or perhaps it will become temporarily unavailable. Either way, consensus is broken and the entire blockchain dies.

So, what's the workaround? Actually, it's rather simple. Instead of a smart contract initiating the retrieval of external data, one or more trusted parties ("oracles") creates a transaction which embeds that data in the chain. Every node will have an identical copy of this data, so it can be safely used in a smart contract computation.

In other words, an oracle pushes the data onto the blockchain rather than a smart contract pulling it in.

When it comes to smart contracts causing events in the outside world, a similar problem appears. For example, many like the idea of a smart contract which calls a bank's API in order to transfer money. But if every node is independently executing the code in the chain, who is responsible for calling this API?

If the answer is just one node, what happens if that particular node malfunctions, deliberately or not? And if the answer is every node, can we trust every node with that API's password? And do we really want the API called hundreds of times? Even worse, if the smart contract needs to know whether the API call was successful, we're right back to the problem of depending on external data.

As before, a simple workaround is available. Instead of the smart contract calling an external API, we use a trusted service which monitors the blockchain's state and performs certain actions in response. For example, a bank could proactively watch a blockchain and perform money transfers which mirror the on-chain transactions. This presents no risk to the blockchain's consensus because the chain plays an entirely passive role.

Looking at these two workarounds, we can make some observations.

First, they both require a trusted entity to manage the interactions between the blockchain and the outside world. While this is technically possible, it undermines the goal of a decentralized system.

Second, the mechanisms used in these workarounds are straightforward examples of reading and writing a database. An oracle which provides external information is simply writing that information into the chain. And a

service which mirrors the blockchain's state in the real world is doing nothing more than reading from that chain. In other words, any interaction between a blockchain and the outside world is restricted to regular database operations.

We'll talk more about this fact later on.

2. ENFORCING ON-CHAIN PAYMENTS.

Here's another proposal that we tend to hear a lot: using a smart contract to automate the payment of coupons for a so-called "smart bond". The idea is for the smart contract code to automatically initiate the payments at the appropriate times, avoiding manual processes and guaranteeing that the issuer cannot default.

Of course, in order for this to work, the funds used to make the payments must live inside the blockchain as well, otherwise a smart contract could not possibly guarantee their payment.

Recall that a blockchain is just a database, in this case a financial ledger containing the issued bond and some cash. So, when we talk about coupon payments, what we're actually talking about are database operations which take place automatically at an agreed time.

While this automation is technically feasible, it suffers from a financial difficulty. If the funds used for coupon payments are controlled by the bond's smart contract, then those payments can indeed be guaranteed. But this also means those funds cannot be used by the bond issuer for anything else. And if those funds aren't under the control of the smart contract, then there is no way in which payment can be guaranteed.

In other words, a smart bond is either pointless for the issuer, or pointless for the investor. And if you think about it, this is a completely obvious outcome.

From an investor's perspective, the whole point of a bond is its attractive rate of return, at the cost of some risk of default. And for the issuer, a bond's purpose is to raise funds for a productive but somewhat risky activity, such as building a new factory.

There is no way for the bond issuer to make use of the funds raised, while simultaneously guaranteeing that the investor will be repaid. It should not come as a surprise that the connection between risk and return is not a problem that blockchains can solve.

3. HIDING CONFIDENTIAL DATA.

As I've written about previously, the biggest challenge in deploying blockchains is the radical transparency which they provide.

For example, if 10 banks set up a blockchain together, and two conduct a bilateral transaction, this will be immediately visible to the other eight. While there are various strategies for mitigating this problem, none beat the simplicity and efficiency of a centralized database in which a trusted administrator has full control over who can see what.

Some people think that smart contracts can solve this problem. They start with the fact that each smart contract contains its own miniature database, over which it has full control. All read and write operations on this database are mediated by the contract's code, making it impossible for one contract to read another's data directly. (This tight coupling between data and code is called encapsulation, and is the foundation of the popular object-oriented programming paradigm).

So, if one smart contract can't access another's data, have we solved the problem of blockchain confidentiality? Does it make sense to talk of hiding information in a smart contract? Unfortunately, the answer is no.

Because even if one smart contract can't read another's data, that data is still stored on every single node in the chain. For each blockchain participant, it's in the memory or disk of a system which that participant completely controls. And there's nothing to stop them reading the information from their own system, if and when they choose to do so.

Hiding data in a smart contract is about as secure as hiding it in the HTML code of a web page. Sure, regular web users won't see it, because it's not displayed in their browser window. But all it takes is for a web browser to add a 'View Source' function (as they all have), and the information becomes universally visible.

Similarly, for data hidden in smart contracts, all it takes is for someone to modify their blockchain software to display the contract's full state, and all semblance of secrecy is lost.

A half-decent programmer could do that in an hour or so.

WHAT SMART CONTRACTS ARE FOR?

With so many things that smart contracts cannot do, one might ask what they're actually for. But in order to answer this question, we need to go back to the fundamentals of blockchains themselves. To recap, a blockchain enables a database to be directly and safely shared by entities who do not trust each other, without requiring a central administrator.

Blockchains enable data disintermediation, and this can lead to significant savings in complexity and cost.

Any database is modified via "transactions", which contain a set of changes to that database which must succeed or fail as a whole. For example, in a financial ledger, a payment from Alice to Bob is represented by a transaction that (a) checks if Alice has sufficient funds, (b) deducts a quantity from Alice's account and (c) adds the same quantity to Bob's.

In a regular centralized database, these transactions are created by a single trusted authority. By contrast, in a blockchain-driven shared database, transactions can be created by any of that blockchain's users. And since these users do not fully trust each other, the database has to contain rules which restrict the transactions performed.

For example, in a peer-to-peer financial ledger, each transaction must preserve the total quantity of funds, otherwise participants could freely give themselves as much money as they liked.

One can imagine various ways of expressing these rules, but for now there are two dominant paradigms, inspired by bitcoin and Ethereum, respectively. The bitcoin method, which we might call "transaction constraints", evaluates each transaction in terms of: (a) the database entries deleted by that transaction and (b) the entries created.

In a financial ledger, the rule states that the total quantity of funds in the deleted entries has to match the total in those created. (We consider the modification of an existing entry to be equivalent to deleting that entry and creating a new one in its place).

The second paradigm, which comes from Ethereum, is smart contracts. This states that all modifications to a contract's data must be performed by its code. (In the context of traditional databases, we can think of this as an enforced stored procedure.) To modify a contract's data, blockchain users send requests to its code, which determines whether and how to fulfill those requests.

As in this example, the smart contract for a financial ledger performs the same three tasks as the administrator of a centralized database: checking for sufficient funds, deducting from one account and adding to another.

Both of these paradigms are effective, and each has its advantages and disadvantages. To summarize, bitcoin-style transaction constraints provide superior concurrency and performance, while Ethereum-style smart contracts offer greater flexibility.

So to return to the question of what smart contracts are for: Smart contracts are for blockchain use cases which can't be implemented with transaction constraints.

Given this criterion for using smart contracts, I'm yet to see a strong use case for permissioned blockchains which qualifies.

All the compelling blockchain applications I know can be implemented with bitcoin-style transactions, which can handle permissioning and general data storage, as well as asset creation, transfer, escrow, exchange and destruction. Nonetheless, new use cases are still appearing, and I wouldn't be surprised if some do require the power of smart contracts. Or, at the very least, an extension of the bitcoin paradigm.

Whatever the answer turns out to be, the key to remember is that smart contracts are simply one method for restricting the transactions performed in a database.

This is undoubtedly a useful thing, and is essential to making that database safe for sharing. But smart contracts cannot do anything else, and they certainly cannot escape the boundaries of the database in which they reside.