*Posted July 19, 2015 by Gideon Greenspan (https://www.multichain.com/blog/author/gdg/) in Private blockchains (https://www.multichain.com/blog/category/private-blockchains/).*

# Ending the bitcoin vs blockchain debate

Is there any value in a blockchain without a cryptocurrency?

The debate has been running for a while but the past month has seen a serious uptick. The question being asked is:

**Is there any value in a blockchain without a cryptocurrency? And can these "tokenless shared ledgers" be called blockchains at all?**

So I've read Bailey's article (http://www.paymentssource.com/news/technology/a-very-public-confluct-over-private-blockchains-3021831-1.html), watched Tim's video (https://www.youtube.com/watch?v=k3pM8vB2QYc), read this Nasdaq post (http://www.nasdaq.com/article/is-a-blockchain-without-bitcoin-possible-or-practical-cm482964), followed Richard's every (http://gendal.me/2015/04/27/how-to-explain-the-value-of-replicated-shared-ledgers-from-first-principles/) word (http://gendal.me/2015/06/08/towards-a-unified-model-for-replicated-shared-ledgers/), and even had my own good-spirited debate (https://www.youtube.com/watch?v=ZIUI8YfMqgE) (see comments) with the Counterparty foundation's Chris DeRose. So much hot air.

One thing Chris does well is boil it down to the question: is the blockchain an economic or a computer science innovation? The implication is that if blockchains are a purely economic innovation, there is no point to blockchains without cryptocurrencies. So let me state my position at the start:

**The bitcoin blockchain was both an economic *and* a computer science innovation**.

I'm allowing "innovation" here to include *a new combination of existing techniques*, rather than something which has no precedent whatsoever. This definition allows the world wide web to be considered as an innovation, even though it did little more than combine hypertext with a twist on some existing Internet protocols. If you want to adopt a stricter definition of innovation, be my guest, but you'll be surprised at how few true "innovations" remain. To paraphrase The Teacher (https://en.wikipedia.org/wiki/Ecclesiastes), there is little new under the sun.

To be precise, I am making the claim that **blockchains without a token do serve a purpose**, but it's a **different purpose** compared to the original bitcoin blockchain. Crypto-heads laugh at token-free blockchains because they can't provide censorship resistance and decentralized security through proof-of-work. Fintech-heads laugh at public blockchains because they are slow, expensive and unsuitable for traditional finance. Well, keep laughing everybody, because I believe you are both right.

I'm going to argue that token-free blockchains are useful for keeping decentralized databases in sync, *even in a single organization in which there is perfect trust*. And then we'll see what other features blockchains offer, which make them suitable for creating consensus for *specific types of transactions* between organizations, where there is only limited and imperfect trust.

Unfortunately, to follow the argument, you're going to have to geek out with me on the bitcoin transactional model, database multiversion concurrency control (MVCC), and the problem of conflict resolution in multi-master database replication. I'll do my best to stick to English but still, this is technical stuff, and there's no avoiding it.

# Bitcoin's transactional model

The bitcoin transactional model is simple but powerful. Every bitcoin transaction has a set of inputs and a set of outputs. Each input "spends" one output of a previous transaction. All of the bitcoin in a transaction's inputs flow into that transaction and are distributed across its outputs according to the quantities written within. In this way, transactions form a multi-way connected chain which terminates at the "coinbase" transactions in which new bitcoins are created.

Bitcoin has a bunch of additional rules which are enforced by every node in the network:

- Every input in a transaction must prove that it has the right to spend the prior output to which it is connected. That right is restricted by conditions encoded within the prior output.
- A transaction must have sufficient total bitcoin in its inputs to cover the total written in its outputs. The only exceptions are coinbase transactions which create new units of the currency.
- Each output can only be spent once, in other words, it can only be connected to one input in one subsequent transaction.

Because of this last rule, the network requires a mechanism for reaching consensus about which transactions are valid, and this is what the blockchain does. Specifically:

**If two transactions attempt to spend the same output, then only one of those transactions will ultimately be accepted. A blockchain acts as a unified mechanism to detect and prevent these conflicts across the network.**

The blockchain is structured as a series of linked blocks, in which each block contains a set of transactions that don't conflict with each other or with previous blocks, starting from the first block created in 2009. In theory, the chain could contain a series of individual transactions, but by grouping transactions into blocks, we gain a number of efficiencies that make the scheme more practical.

So what is the purpose of a cryptocurrency in all this? It comes down to the question of who decides on the blocks which form the chain. Bitcoin is decentralized and has no authority that can make this decision, so it needs to find some other way of reaching consensus.

We might like to use a democratic approach, in which nodes in the network vote on blocks, and the majority wins. Unfortunately, as any Internet poll can demonstrate, representative democracy is not possible online, because of the problem of impersonation (also known as a Sybil attack (https://en.wikipedia.org/wiki/Sybil_attack)). One person can take over a million computers and decide how they vote, thus seizing control of the network consensus. Nobody else will even know this has happened.

To solve this, bitcoin makes it deliberately difficult to add a block to the chain, via a process called "mining". To create a block, you must solve a difficult but pointless mathematical problem that demands a lot of computation (and therefore electricity and money). You also need some luck, since you are in competition with many other block miners around the world. You can't get ahead for long by buying a more powerful mining computer, because the network regularly adjusts the problem's difficulty to keep a steady global rate of one block per 10 minutes.

If it's so difficult and costly to create a block, why would anyone bother? The answer is in the block reward. The successful miner of a block controls the coinbase transaction that awards them 25 bitcoins (this sum halves every four years). They can sell these bitcoins on the open market for $7,000 (at today's rate), pay off their electricity bill and hopefully pocket some profit. Miners also collect a little extra from fees that are attached to transactions, although for now these fees play a minor role.

So bitcoin generates consensus via proof-of-work and the crux of the bitcoin-heads' argument is this: **Without a cryptocurrency, there is no way to incentivize decentralized mining of blocks.** Therefore there is no way to secure an open blockchain against impersonation attacks. Therefore anybody can monopolize the network consensus and render the whole thing useless. I won't argue with any of that.

# Multiversion concurrency control

In the meantime I want to talk about something that may seem completely unrelated.

A database is a repository of structured information, grouped into spreadsheet-like entities called tables. A simple example of such a table is a list of bank accounts, in which each row contains an account number along with the balance of that account. Let's say your account starts the day with a balance of $900. Today an automatic mortgage payment of $750 is scheduled and you also need to withdraw $400 from an ATM. Unfortunately you do not have an overdraft facility so one of these operations is set up to fail.

The processes for mortgage payments and ATM withdrawals run on separate systems, both of which access this single account database. Let's say that each process works by reading your account's balance, checking it is sufficient for the operation, initiating that operation, verifying the operation completes, calculating the new balance and then finally writing it into the database.

So long as your mortgage payment and ATM withdrawal don't overlap, this logic will work fine. The first operation will execute successfully, and the second will abort because your account has insufficient funds. Depending on the order, you'll get an angry phone call from the bank or a rude message on the ATM screen.

But what happens if the two processes happen to start at the same time? In this case, each will read your account's balance and deem it sufficient to proceed. When the mortgage payment completes, your new balance will be calculated as $150 and written to the database. When the ATM withdrawal completes, your new balance of $500 will similarly be written. One of these write operations is going to override the other and, depending on your luck, you'll receive a $750 or $400 bonus from your bank. No doubt you'll soon learn to time your ATM visits for mortgage day.

Of course, this doesn't happen in reality, because of a database technology called concurrency control (https://en.wikipedia.org/wiki/Concurrency_control). Concurrency control keeps our data (especially financial) sane and secure, and it comes in many forms. But all share the principle that database operations are grouped into "transactions", which are treated atomically, meaning that they succeed or fail as a whole. Concurrency preserves consistency by locking or freezing parts of a database while they are in use by one transaction, to prevent other transactions from operating on the same information in a conflicting way.

If we didn't need to run transactions in parallel, we could lock the entire database for the entire duration of every single transaction. However this is not practical in most real-world applications. A good concurrency control scheme permits parallel operations by locking as little data as possible for as short a time as possible. In the example above, only the database row corresponding to your account would be locked, and only for the split second in which a final check and deduction took place. A conflicting transaction operating in parallel would simply have to wait until this lock is released.

One popular concurrency control technique is called multiversion concurrency control (https://en.wikipedia.org/wiki/Multiversion_concurrency_control), or MVCC for short. In MVCC, each transaction sees a consistent snapshot of the data at a certain point in time, even if part of that data is in the process of being updated by a second simultaneous transaction. This snapshot isolation (https://en.wikipedia.org/wiki/Snapshot_isolation) property ensures, for example, that a statement showing our total balance across several accounts will always be correct, even if some funds are in the process of moving from one account to another. One transaction will only affect the data seen by a second transaction if the second begins after all of the first's changes have been successfully applied.

Behind the scenes, MVCC works by allowing multiple versions of a row to be maintained simultaneously, alongside a timestamp that represents each version's date of last modification. Modifying a database row in MVCC marks the current version of that row for deletion, while applying the modification to a *copy of that row* with an updated timestamp. From the perspective of the database's storage layer, there is no such thing as modifying a row in place. Each transaction knows exactly when it started, and only sees versions of rows whose timestamp predates that time. Old versions of rows can be removed from storage once there are no ongoing transactions which might need to access them.

Crucially for our purposes here, MVCC prevents conflicts between write operations. Specifically:

**If two transactions attempt to delete the same row version, then only one of these transactions will ultimately be accepted. Multiversion concurrency control acts as a unified mechanism to detect and prevent these conflicts within a database.**

Ring any bells? There's one more piece of background we need to discuss.

# Multi-master database replication

Now let's talk about database replication, in which a database exists in multiple copies. There are a number of good reasons to replicate a database, such as:

- To increase reliability, so that if one copy of the database is lost (e.g. due to a disk failure), we can instantly switch over to a second copy.
- To increase throughput, if the volume of operations goes beyond the capacity of a single database server.
- To reduce latency, so that processes running in the Singapore office need not wait for responses from a database sitting in Toronto.

When it comes to *reading* data from databases, replication is an ideal technique, because all of the replicas contain the same information. However things get stickier when it comes to write operations, because we need to decide where those write operations are performed, and how they get transferred to other copies of the database.

The most common answer is to use master-slave replication, in which a single database (the "master") is considered authoritative. Any changes to the data are performed exclusively on the master and then trickle down to all of the other "slave" databases via a transaction log. This keeps all the database copies (more or less) instantly in sync.

Unfortunately, if write operations are frequent, master-slave replication brings us right back to the problem that replication was designed to solve. The master database becomes a bottleneck in terms of reliability, throughput and latency, since every write operation is performed on it alone.

A more complex strategy is called multi-master replication, in which writes can be performed on any of the database copies, rather than on a single master. In this case, the copies share updates with each other in a peer-to-peer fashion in order to remain in sync.

This sounds simple in theory, but multi-master replication introduces a new problem because conflicts can arise. What if two copies of a database update the same row at the same time, then attempt to exchange these updates with each other? Both databases will notice that a conflicting update has taken place, and have to apply some agreed strategy for resolving these conflicts. And here things get pretty complex (http://datacharmer.blogspot.com/2013/03/multi-master-data-conflicts-part-1.html) – see the docs for MySQL (https://dev.mysql.com/doc/refman/5.1/en/mysql-cluster-replication-conflict-resolution.html), SQL Server (https://technet.microsoft.com/en-us/library/bb934199.aspx) or Oracle (https://docs.oracle.com/cd/E11882_01/server.112/e10706/repconflicts.htm#REPLN005) for some examples of conflict resolution strategies. (I'm ignoring synchronous or so-called "eager" multi-master replication, in which all replicas must commit to a write operation before it can take place, because that turns *every* copy of the database into a bottleneck.)

So here's where all this background is leading:

**Wouldn't it be nice if we could have distributed multiversion concurrency control, to prevent conflicts occurring in multi-master replication?**

Well, yes, I imagine that would be very nice indeed. And I believe that this is precisely what blockchains do.

# Blockchains as distributed MVCC

Let's copy down a couple of sentences that I wrote in bold above:

If two transactions attempt to **spend** the same **output**, then only one of those transactions will ultimately be accepted. **A blockchain** acts as a unified mechanism to detect and prevent these conflicts **across the network**.

If two transactions attempt to **delete** the same **row version**, then only one of these transactions will ultimately be accepted. **Multiversion concurrency control** acts as a unified mechanism to detect and prevent these conflicts **within a database**.

These sentences are identical except for the bold terms. So here's what I'm going to claim:

**A blockchain provides distributed MVCC (with a few extra bells and whistles).**

Let's flesh out the comparison a little further. From the perspective of a blockchain node, the current set of unspent bitcoin transaction outputs forms a database, in which each row is a single unspent output. This is similar to the database of bank accounts we described earlier, with the minor difference that the balance of each account can be split across multiple rows, each of which is marked with the same account number.

A bitcoin transaction spends one or more of these outputs and creates one or more new outputs as a result. This is exactly like a database transaction which deletes one or more row versions, and creates one or more new rows as a result (recall that in MVCC that there is no such thing as modifying a row in place). The bitcoin blockchain ensures that a single output cannot be spent by more than one transaction. This is equivalent to ensuring that a single row version cannot be deleted by more than one database transaction.

Now before we get carried away, I'm not claiming that blockchains are a great general purpose technology for distributed database synchronization in a fully trusted environment. There are plenty of other technologies such as Paxos (https://en.wikipedia.org/wiki/Paxos_(computer_science)), Raft

(https://en.wikipedia.org/wiki/Raft_(computer_science)) and Two-phase commit (https://en.wikipedia.org/wiki/Two-phase_commit_protocol) which perform the job very nicely. But I do believe that blockchains have a sweet spot, which can be characterized as applications where:

- We can accept a short delay between when a transaction is probably accepted and when it is definitely accepted. (This delay can be a matter of seconds rather than 10 minutes as in bitcoin.)
- Conflicting transactions should never happen if everyone is being honest and their systems are functioning properly.
- Each transaction modifies just a few rows simultaneously (otherwise our blockchain transactions will have an unwieldy number of inputs).
- The size of each database row is fairly small (again, to prevent our blockchain transactions ballooning in size).

All of these criteria are fulfilled by financial applications. The financial world is already used to delays (of up to 3 days!) between conducting a transaction and its final settlement. In terms of preventing conflicts, it has contracts and regulations in place to detect fraud, and the consequences can be severe. And the amount of data involved in each transaction is pretty small – think of the bank account example above.

So far, all I've demonstrated is that blockchains are yet another synchronization mechanism for distributed databases. Big wow. Things only get truly interesting when we consider the additional features that blockchains provide.

# Blockchains beyond MVCC

A bitcoin transaction does much more than just point to some previous transaction outputs and create some new ones in their place. Even the simplest bitcoin transaction serves two additional purposes.

First, the rules regarding valid transactions contain some of the application logic for our account database. Recall that the total quantity of bitcoin in a transaction's inputs must cover the total quantity in the outputs. Translated into database application logic, this is a rule which states that database transactions (with the exception of coinbases) are not permitted to increase the total quantity of bitcoin in the database. This kind of constraint goes beyond regular database stored procedures (https://en.wikipedia.org/wiki/Stored_procedure) because it cannot be circumvented under any circumstances.

Second, recall that each bitcoin transaction output encodes the conditions under which it can be spent. For regular bitcoin outputs, this condition is based on public key cryptography. A public address is embedded inside the output "script" so that it can only be spent using the private key corresponding to that public address. If we consider this output to be a database row, what we have is a database with per-row permissions which are based on public key cryptography. Furthermore, every transaction presents a publicly auditable proof that its creator(s) had the right to delete/modify its prior rows. This (I believe) is a genuine novelty in database technology.

And again, it just so happens that both of these features are incredibly useful for financial applications. We like the fact that our database ensures, at the lowest possible level, that money cannot be created out of thin air. And we like having an incontrovertible audit trail demonstrating that every transaction was authorized by the holder of the funds which it moved. As discussed in detail here (/blog/2015/09/delivery-versus-payment-blockchain/), we may also like performing safe atomic peer-to-peer exchange transactions (delivery-versus-payment in finance-talk), without even knowing the identity of our counterparty.

# So where's the token?

Of course, none of this is a coincidence, because bitcoin itself is a beautiful peer-to-peer financial application. Still, none of the above characteristics of a blockchain are dependent on the token at all. If we modify our "database" schema so that each row can represent multiple assets, rather than the blockchain's native currency, then we can rid ourselves of that currency entirely. This leaves us with a blockchain as a way to achieve consensus and security in a peer-to-peer financial application for *any class of asset*.

Only one little question though: **Who does the mining to generate this consensus?** In bitcoin anonymous miners must perform expensive useless computations, and are incentivized to do so by the block rewards (and transaction fees) denominated in the blockchain's native currency or token. Do we have any other options?

It turns out that we do. We can have a closed list of permitted miners, who identify themselves by signing the blocks that they create. Rules about distributed consensus (or "mining diversity" as we call it in MultiChain (http://www.multichain.com/)) provide a different way of preventing minority control of the blockchain, **so long as you can accept that miners are pre-approved**. Of course for bitcoin this is not acceptable, because part of the point is to permit anonymous mining, so there is no way to censor transactions centrally. But if, say, we had a highly regulated financial system, in which bitcoin's model was inapplicable, perhaps we could accept a pre-approved list of miners after all? If we had enough of them, and spread them well enough between institutions, and had legal contracts with all of them, are they really likely to gang up and undermine the network they depend on, when doing so will land them in jail?

# Epilogue

I hope I have demonstrated that blockchains without tokens do have some useful applications, even if these are very different from the bitcoin blockchain. Nonetheless one question remains:

**Are these permissioned, token-free shared ledger systems really worthy of the name "blockchain"?**

The short answer is: who cares? It's rarely worth arguing about the meaning of words, because there is no right answer (https://en.wikipedia.org/wiki/Philosophical_Investigations).

But to go a little deeper, let's say I accept the premise that the bitcoin blockchain is the archetypal blockchain. In that case, what we should really be asking is:

**Are these shared ledgers similar enough to bitcoin to merit the name "blockchain"?**

My own personal view here is *yes*. Because they share a huge number of technical similarities, even while they differ in the permissions model and economic incentives. And most importantly, because they both generate consensus in a distributed database via a **chain of blocks**.

Thank you for reading.

You can follow me on Twitter here (http://twitter.com/CoinSciences). See also: Delivery versus payment on a blockchain (/blog/2015/09/delivery-versus-payment-blockchain/).

*Here are a couple of other pieces worth reading on this subject by Piotr Piasecki (http://tpbit.blogspot.com/2015/06/bitcoin-vs-blockchain.html) and Dug Campbell (http://www.dugcampbell.com/bitcoin-v-the-blockchain/).*

# 7 Responses to "Ending the bitcoin vs blockchain debate"

### Simon Taylor

Bravo

July 20th, 2015 (https://www.multichain.com/blog/2015/07/bitcoin-vs-blockchain-debate/#comment-3)

### Dug Campbell (http://www.dugcampbell.com)

Great post Gideon. For me, the key point here is that both sides in this debate tend to be correct – it's just that they're usually talking about different things 🙂

July 20th, 2015 (https://www.multichain.com/blog/2015/07/bitcoin-vs-blockchain-debate/#comment-4)

### Bailey Reutzel

Great stuff, Gideon. You explained the technical side of the use case better than I could… But thanks for the shout out!

July 21st, 2015 (https://www.multichain.com/blog/2015/07/bitcoin-vs-blockchain-debate/#comment-9)

### Nathan Wosnack (http://www.blockchainfactory.com)

Well said, Gideon. A very balanced, well cited, well thought out article. Completely agree with your points. I've been arguing this for months that "The bitcoin blockchain was both an economic and a computer science innovation".

July 22nd, 2015 (https://www.multichain.com/blog/2015/07/bitcoin-vs-blockchain-debate/#comment-10)

### Joshua Unseth, White Knight of Bitcoin (https://junseth.com)

Any time your views line up with Nathan Wosnack's, you probably need to have a hard sit down and do some reconsidering.

July 23rd, 2015 (https://www.multichain.com/blog/2015/07/bitcoin-vs-blockchain-debate/#comment-13)

### Rich Collina

Would these systems only allowed trusted parties to transact (vs mine)

July 24th, 2015 (https://www.multichain.com/blog/2015/07/bitcoin-vs-blockchain-debate/#comment-16)

### Gideon Greenspan

It depends. At least in a MultiChain blockchain you can choose whether to allow anyone to send/receive, or whether to restrict it to a list of allowed addresses.

July 24th, 2015 (https://www.multichain.com/blog/2015/07/bitcoin-vs-blockchain-debate/#comment-17)

## Recent Posts


- Scaling blockchains with off-chain data (https://www.multichain.com/blog/2018/06/scaling-blockchains-off-chain-data/)
- R3 Corda: Deep dive and technical review (https://www.multichain.com/blog/2018/05/r3-corda-deep-dive-and-technical-review/)
- Second MultiChain 2.0 preview release (https://www.multichain.com/blog/2018/01/second-multichain-2-0-preview-release/)
- Three (non-pointless) permissioned blockchains in production (https://www.multichain.com/blog/2017/11/three-non-pointless-blockchains-production/)
- First MultiChain 2.0 preview release (https://www.multichain.com/blog/2017/11/first-multichain-2-preview-release/)
- MultiChain 1.0 released with 14 new partners (https://www.multichain.com/blog/2017/08/multichain-1-released-new-partners/)
- A rational take on cryptocurrencies (https://www.multichain.com/blog/2017/07/rational-take-cryptocurrencies/)
- MultiChain 1.0 beta 2 and 2.0 roadmap (https://www.multichain.com/blog/2017/06/multichain-1-beta-2-roadmap/)
- The Blockchain Immutability Myth (https://www.multichain.com/blog/2017/05/blockchain-immutability-myth/)
- Video talk: Blockchains vs databases (https://www.multichain.com/blog/2017/04/video-difference-blockchain-database/)

- MultiChain enters beta with 15 new partners (https://www.multichain.com/blog/2017/03/multichain-enters-beta-new-partners/)
- Wolfram Mathematica to add MultiChain integration (https://www.multichain.com/blog/2017/03/wolfram-mathematica-multichain-integration/)
- MultiChain source code release (https://www.multichain.com/blog/2017/01/multichain-source-code-release/)
- How to spot a half-baked blockchain (https://www.multichain.com/blog/2016/12/spot-half-baked-blockchain/)
- Understanding zero knowledge blockchains (https://www.multichain.com/blog/2016/11/understanding-zero-knowledge-blockchains/)
- First MultiChain Partners Announced (https://www.multichain.com/blog/2016/10/first-multichain-partners-announced/)
- Introducing MultiChain Streams (https://www.multichain.com/blog/2016/09/introducing-multichain-streams/)
- Announcing the new MultiChain wallet (https://www.multichain.com/blog/2016/07/announcing-the-new-multichain-wallet/)
- Smart contracts and the DAO implosion (https://www.multichain.com/blog/2016/06/smart-contracts-the-dao-implosion/)
- Four genuine blockchain use cases (https://www.multichain.com/blog/2016/05/four-genuine-blockchain-use-cases/)
- Beware the impossible smart contract (https://www.multichain.com/blog/2016/04/beware-impossible-smart-contract/)
- Blockchains vs centralized databases (https://www.multichain.com/blog/2016/03/blockchains-vs-centralized-databases/)
- Recent Features and 2016 Roadmap (https://www.multichain.com/blog/2016/03/recent-features-2016-roadmap/)
- Moving on from big blockchains (https://www.multichain.com/blog/2016/01/moving-on-from-big-blockchains/)
- Avoiding the pointless blockchain project (https://www.multichain.com/blog/2015/11/avoiding-pointless-blockchain-project/)
- Smart contracts make slow blockchains (https://www.multichain.com/blog/2015/11/smart-contracts-slow-blockchains/)
- Smart contracts: The good, the bad and the lazy (https://www.multichain.com/blog/2015/11/smart-contracts-good-bad-lazy/)
- Private blockchains are more than “just” shared databases (https://www.multichain.com/blog/2015/10/private-blockchains-shared-databases/)
- Delivery versus payment on a blockchain (https://www.multichain.com/blog/2015/09/delivery-versus-payment-blockchain/)
- Ending the bitcoin vs blockchain debate (https://www.multichain.com/blog/2015/07/bitcoin-vs-blockchain-debate/)

## Categories

- Cryptocurrencies (https://www.multichain.com/blog/category/cryptocurrencies/)
- MultiChain (https://www.multichain.com/blog/category/multichain/)
- Private blockchains (https://www.multichain.com/blog/category/private-blockchains/)
- Smart contracts (https://www.multichain.com/blog/category/smart-contracts/)

## Archives

- June 2018 (https://www.multichain.com/blog/2018/06/)
- May 2018 (https://www.multichain.com/blog/2018/05/)
- January 2018 (https://www.multichain.com/blog/2018/01/)
- November 2017 (https://www.multichain.com/blog/2017/11/)
- August 2017 (https://www.multichain.com/blog/2017/08/)
- July 2017 (https://www.multichain.com/blog/2017/07/)
- June 2017 (https://www.multichain.com/blog/2017/06/)
- May 2017 (https://www.multichain.com/blog/2017/05/)
- April 2017 (https://www.multichain.com/blog/2017/04/)
- March 2017 (https://www.multichain.com/blog/2017/03/)
- January 2017 (https://www.multichain.com/blog/2017/01/)
- December 2016 (https://www.multichain.com/blog/2016/12/)
- November 2016 (https://www.multichain.com/blog/2016/11/)
- October 2016 (https://www.multichain.com/blog/2016/10/)
- September 2016 (https://www.multichain.com/blog/2016/09/)
- July 2016 (https://www.multichain.com/blog/2016/07/)
- June 2016 (https://www.multichain.com/blog/2016/06/)
- May 2016 (https://www.multichain.com/blog/2016/05/)
- April 2016 (https://www.multichain.com/blog/2016/04/)
- March 2016 (https://www.multichain.com/blog/2016/03/)
- January 2016 (https://www.multichain.com/blog/2016/01/)
- November 2015 (https://www.multichain.com/blog/2015/11/)
- October 2015 (https://www.multichain.com/blog/2015/10/)
- September 2015 (https://www.multichain.com/blog/2015/09/)
- July 2015 (https://www.multichain.com/blog/2015/07/)

## RSS feed

- Recent blog posts (https://www.multichain.com/feed/)