The Vertical Layer Problem

Josh · Secure Sovereign

Published August 27, 2026

Why Bitcoin cannot be governed like an open source project

Contents

Bitcoin cannot be governed like Linux. A dictator is a good way to ship a kernel. It is a bad way to write Bitcoin's node software, even when the dictator is competent and the code is good.

Bitcoin's consensus layer is horizontal. Every node checks blocks on its own, and no node can force another to take a bad one. The software that implements those rules is written by a hierarchy. Replacing the person at the top does not fix that, and making that person more accountable does not fix it either. You need four things: nobody shoves a change through because they hold a title, the work sits on a public record, leaving a development project is cheap, and other implementations can build against a written spec. Nodes stay equal. The people who write the code never will be. That is not a reason to give those people a throne.

The model works

Benevolent dictatorship produces good software. Linux runs most of the internet's infrastructure and was governed for decades by one person's judgment about what merged. Python, Django, and Rust all had stretches of single-authority governance, and the work held up.

Concentrated authority makes architectural decisions faster than committees do. It keeps one design across a large codebase, and it ends fights that would otherwise eat years. Linux is a product. A small group decides what goes in the kernel and everyone else downloads the result. Someone has to pick what ships, and a hierarchy is the right way to run that.

You can copy the code and start your own tree. Almost nobody does, because the copy stops getting the patches that keep landing in the original project, and those patches are what people actually want. Bitcoin's software layer already runs that way. Most operators download a default and wait for patches. Bitcoin's money rules and the extra services people want on top of them are not the same product. Treating them as one tree is how a wallet fight or a relay fight becomes a governance crisis.

That model works for ordinary software. It is the wrong model for money.

The structure arrives on its own

The hierarchy also shows up because paid engineers bring their workplace with them. Nobody has to conspire for that.

Most of the work is done by people employed to do it. The majority of Linux kernel contribution comes from paid engineers at Intel, Red Hat, Google, IBM, and a handful of others, and that has been true for well over a decade. The same pattern shows up across most infrastructure projects of any size. Volunteer contribution is real. It is not what carries the load.

People who spend forty hours a week inside a company's engineering process bring that process with them. Review turns into an approval chain. Roadmaps get set by a small group. Release management belongs to specific people on a schedule. Disputes go up to whoever is supposed to end them. Nobody votes any of that in. It arrives as professional habit, because it is the only model of software coordination most working engineers have ever practiced.

Core's contributors are largely employed or funded by a small number of organizations. The governance model that emerged at Core is what happens when people trained in corporate engineering coordinate a project and nobody is asked to design a governance model on purpose.

The usual worry about concentrated funding is that donors pressure specific outcomes. That takes intent, coordination, and a visible fight. The quieter path runs all the time. People paid by companies bring their company's way of working with them, with no one telling them to.

Replacing the current contributors accomplishes nothing on its own. A different set of people, from the same professional background and funded the same way, would rebuild the same structure within a few years. The structure comes from where those people were trained, not from who they are. The social-layer version of that argument is *The Social Layer Is the Attack Surface*.

If paid engineers really import the hierarchy, a project staffed from different backgrounds and funded differently should not reconcentrate merge authority the same way. If it still does, the workplace-habit story is wrong, and hierarchy is coming from something more basic about coordinating software.

Not all of Core's structure is imported habit. Some of it was chosen on purpose. Merge access tightened after the inflation bug for stated reasons. Review culture hardened after the blocksize war as a considered response to what that conflict cost. Reluctance to touch consensus lightly is a position people hold out loud. They also decide which changes count as consensus changes, which is how a proposal gets labeled too dangerous to merge. If that label is doing real work, it should show up in what shipped and what sat. That record is *What Bitcoin's Stalled Proposals Tell You*. Attributing all of it to unexamined corporate reflex would be too clean. Habit and deliberate choice both contributed. Habit set the default. Deliberate choice then reinforced it.

Bitcoin's coordination problem is a different problem

Bitcoin's coordination problem is agreement among peers who do not trust each other, and Satoshi solved that part in 2008. Linux produces a kernel people download. Bitcoin produces agreement about which chain is valid, and each node makes that agreement on the same terms.

Writing the code that does that check is not equally shared, and never will be. The Commons white paper puts a number on that: the people who can do consensus-critical Bitcoin work across cryptography, distributed systems, and adversarial review number in the dozens to a few hundred worldwide. Most participants will never write consensus code. Expertise is uneven, and no governance model changes that. Uneven expertise is not a reason to put merge authority in one seat. It is a reason to record work in public and let that record decide who gets a vote.

The contradiction

On paper a node can refuse a release. In practice there is nowhere to go. There is no independently written client an operator can switch to. Knots is a Core fork, not a second implementation, and it is not a destination. You can pin an old Core version for a while. You cannot pin it forever. Unmaintained node software rots. Bugs pile up, the network moves, and eventually you take the next download or you fall behind. Shipping code and changing the money are still not the same act. The people who write the default still govern, because refusal is a delay, not an exit.

After the OP_RETURN relay change merged, Bitcoin Knots went from about 4.7% of reachable nodes at the end of April 2025 to a peak of 25.45% on September 14, 2025, and stayed above 19% through that stretch, according to Coin Dance. That was a policy protest on the same codebase,

not a second client. It did not give anyone a place to stay. *Making Core Irrelevant* treats the reachable-node split as demand for a policy alternative, not as a break in the implementation monoculture.

Running different defaults is cheap compared to risking a chain split. Miners can refuse an upgrade, so a consensus change can be blocked. That veto is rarely used. The cost of refusing a default falls on the operator. The benefit of setting it accrues to whoever sets it. Nobody has to capture consensus enforcement to govern this way.

What Core actually looks like

On paper the model is rough consensus among a broad contributor base. In the merge log it is a handful of accounts doing most of the merges. Bitcoin Core is not formally a benevolent dictatorship and has never claimed to be. How it actually merges looks like a dictatorship with the name taken off, which is worse, because undeclared concentration is harder to see and harder to contest.

Concentration of merge rights on its own still proves nothing. Any mature project restricts commit access, and Linux would look just as lopsided on the same measure. A high concentration of who may merge could just mean they are limiting who can push to a codebase where a bad merge could print coins. That would be a defense if the scarce mergers were also doing the review. They are not.

Since 2017, the top three reviewers account for roughly nineteen percent of review activity while the top three mergers account for roughly ninety percent of merges, according to the Bitcoin Governance Research dataset drawn from sixteen years of Core commit history. The dataset counts comments, ACKs, and review submissions on GitHub. It does not count hours spent checking consensus-critical code or how deep each review went, so this is about who shows up in the record, not who understood the code best in any given week. Review is spread out. Merge authority is not. Merge Gini sits near 0.85. In one recent year, more than half of all merges flowed through a single individual funded by one grant organization, a figure Brink published in its 2025 Engineering Impact Report. Self-merge among top contributors ran above a quarter of their pull requests, meaning the same people proposing changes were approving them.

The bug usually cited with these numbers is CVE-2018-17144, the inflation bug that sat in Core for eighteen months. People use it to prove Core is incompetent, which it does not prove. It sat in production, was never exploited, and was patched quickly after disclosure. Patching it fast after disclosure is not the same as catching it. It propagated to Knots, ABC, and Unlimited, because those clients descend from Core's codebase. Listing those clients as independent checks is a mistake when they share a lineage. The case study is *Governance Paralysis Was The Victory*. What the CVE does show is that implementations copied from a common codebase inherit that

codebase's blind spots, which is the argument for building from a specification rather than from observed behavior. An implementation written against a stated standard does not inherit Core's bugs, because it was never built by copying Core.

Three things that are fine for ordinary software are a problem in the process that writes Bitcoin's defaults. Decisions about what ships cannot be reversed except by whoever made them. Authority lives in a person rather than in a record, so when that person leaves you do not have a procedure, you have a crisis. There is nothing to transfer except the habit of people listening to you, and that habit does not transfer. Independence lasts exactly as long as one person's capacity to resist pressure. Legal, financial, regulatory, or personal leverage only needs one target.

Succession

Every vertical project eventually loses its founder. Python is the usual counterexample. Guido stepped back, the community designed a steering council, and the language came through it intact, with no cryptographic governance, no formal specification, and no distributed signing authority. What carried it through was that people still wanted to work together, and that they had time. There was enough shared purpose to design a governance model after Guido left, and nothing catastrophic happened while they were designing it. Users just kept running the old interpreter and waited.

If Bitcoin's maintainers disappeared tomorrow, existing nodes would keep running. Python users can do the same thing. The difference is what cannot wait. CPython can sit unchanged while people design a council. Bitcoin still needs someone who can patch a bug that prints coins, and someone still picks what new operators download. You cannot put those jobs on hold until a new council exists.

This ecosystem also does not have the shared purpose Python had. The blocksize war was the test of whether it could resolve a fundamental disagreement through social process under pressure, and it could not do so without years of conflict and permanent factional damage. Taproot is the usual reply, because that change did ship and activate. It is the wrong comparison. Taproot was not the lead maintainer disappearing, and it was not a fight on the scale of the blocksize war. That it shipped says this ecosystem can activate a change when the disagreement is smaller. It does not say the community can design a governance model after the people in charge are gone. A succession plan that needs the community to agree under pressure has already been tested here. It failed, and that failure is why Core still cannot process contested work.

The war ended in a fork. Bitcoin Cash left, Bitcoin kept the ticker and the market. Some people say a fork is how this ecosystem keeps developers honest. The blocksize war was that fork. You cannot treat it as both the check working and proof that social process failed.

What matters is which thing gets forked. Bitcoin Cash forked the chain, which meant splitting the ledger, the hashrate, the economic network, and the community, permanently and by design. The cost of that exit was so high that it scares people off without keeping anyone honest. Nobody threatens it credibly, because the threat costs the threatener nearly everything they were trying to protect.

A governance fork is a different operation. Two implementations built against the same specification remain consensus-compatible, which means a disagreement about how a project is run can be resolved by one group building differently without anyone leaving the money. The chain, the ledger, and the economic network do not split. What splits is a development process, and a development process can afford to split. The chain-fork trap is *The Social Layer Is the Attack Surface, §VII*.

What a horizontal development layer looks like

Vote weight comes from work recorded in public, not from a title someone holds.

The Bitcoin Commons white paper does that with two questions at once. Where is the change: the consensus spec, the protocol, the node, or an optional module. What kind of change is it: a routine fix, a feature, something that touches consensus validation, an emergency patch, or a change to the governance rules themselves. When both apply, you take the higher signature count and the longer wait. A bug fix in the consensus layer still needs the consensus-layer bar. A feature in an optional module does not.

Routine maintenance clears on fewer signatures and a short wait. Consensus-adjacent changes need near-unanimity and a delay measured in months. Changes to the governance rules themselves need more than that. A contributor cannot approve their own merge. Even a repository administrator cannot bypass the signature check.

Optional features load as separate processes outside the money rules. Lightning, mining interfaces, and similar tools can compete without anyone treating them as a consensus change. The base node still validates blocks against the spec. That is how a feature fight stops being a throne fight.

A further ladder, vote weight on the crates you have actually shipped, is specified as a complement to that merge policy. It is not what is enforced today.

Every action is cryptographically signed and publicly auditable. Power is visible because anyone can check the record, not because people are supposed to behave. The software layer cannot make every contributor equal the way every node is equal. It can make the record checkable, which is what matters for who may change the code.

Where it fails

The crate ladder, if it ever ships, can let a shallow contributor outvote the person who wrote the subsystem. A single merged pull request can put someone on questions they cannot evaluate. Someone who has shown they understand the architecture still has to gate who votes. That limit is real. Below it, more votes make worse decisions.

The merge policy specified today has a different failure: the people who hold the keys can collude. That is why the keys are supposed to sit in different places, and why a governance ruleset has to be forkable without splitting the money.

Distribution is slow when you need a same-day security patch. Ordinary merge rules should stay slow. The specified answer is a declared emergency with a clock, not a person who can merge anything.

If one implementation still defines the rules everyone else must copy, distributed governance is theater. A spec plus full-chain tests is what turns "we match Core" from a claim into a pass/fail check, and that only works if the spec matches mainnet, including the ugly historical bugs.

This model has not been run at scale. The white paper says production governance is not activated yet. Gates and declared emergencies are rules on a public record, not a person constrained only by custom. That should break differently under pressure. That is a prediction until someone runs it. Those limits are why the spec, the emergency path, and cheap exit have to exist.

The governance model described is Bitcoin Commons, the specification is the Orange Paper, and the implementation is BLVM. Those are the things being argued for.

Without a spec, fights still go to whoever has been around longest. There is nothing independent to measure a judgment against, so authority reconcentrates no matter what a governance document says. That is why Core cannot be repaired by redistributing merge rights, and why the missing spec is what keeps the monopoly in place, not an oversight. The longer write-up is *Who Controls Bitcoin*.

A written specification gives contributors something to be right about that is not a person's opinion, and writing it is the hard part, because Bitcoin's live rules include serialization quirks, historical bugs the network accepted, and a policy and consensus boundary that is blurrier in practice than on paper. A spec that matches mainnet has to encode all of it, including the parts nobody would design on purpose, which makes it a large contested document rather than something someone sits down and writes in a week.

With a spec, fights become what the document says. Formal verification and machine-checkable proofs shrink the room to argue about what the spec means.

Exit has to be cheap enough to be credible. A formally specified implementation can be forked against the specification rather than reverse-engineered from Core. A governance ruleset can be exported as a signed package and replaced without splitting the ledger. That makes a fork a real option rather than a threat nobody exercises. Cheap exit keeps people honest in ways internal process cannot. The sequencing of that exit is *Making Core Irrelevant*.

The standard

Bitcoin does not get a pass because some of the people writing the code are skilled or well-intentioned. Failed monetary institutions had skilled, well-intentioned people too. What matters is whether someone with merge authority still decides what Bitcoin nodes run on ordinary work, and whether an inflation bug can be patched without giving that person the power to merge whatever they want.

Commons specifies the second as a three-class emergency system. Emergency keyholders declare a class. That declaration changes the merge rules until a clock expires. Network-threatening work gets the fastest path, including no review wait while the declaration is live. Less severe classes keep longer waits and higher bars. Declaring an emergency is a different act from tagging a pull request as emergency work. After the clock, a written post-mortem is required and ordinary rules return. Nobody keeps a low bar because they might need one later.

That is the replacement for one person merging an overnight patch because they can. A governance model that works only while the right person is in charge was built for Linux, and it does not belong on Bitcoin's node software.

Sources

- Bitcoin Governance Research. Merge/review concentration, Gini, self-merge
- Brink, Engineering Impact Report 2025. One-person merge share
- Coin Dance. Knots reachable-node share
- Bitcoin Commons white paper. Dual bars (where and what), optional modules, three-class emergency (declaration vs PR tag), governance-ruleset exit
- Orange Paper / Bitcoin Commons consensus spec
- Bitcoin Commons documentation. BLVM spec lock, governance enforcement

SECURESOVEREIGN

Markdown is the canonical source: https://secsov.com/articles/the-vertical-layer-problem/index.md

Formatted snapshot of https://secsov.com/articles/the-vertical-layer-problem

Josh · Secure Sovereign