



POSSIBILITY SPACE • FULL USE CASE

06

History nobody has to host

A blockchain's entire past, and public datasets like it, kept reachable by the people who use them instead of by whoever can afford the servers.

BUILD STATE

Yours to take

COMPLEXITY



Vision

A blockchain's whole proposition is that anyone can check it. In practice almost nobody does. Asking what happened three years ago means asking a provider, because keeping that history yourself runs to terabytes and a machine that does little else, so the network built so that no one has to be trusted gets read, in the end, through a handful of companies' endpoints.

Nothing about that looks like censorship until the day it does. An endpoint starts filtering, or rate-limits at the wrong moment, or a team discovers that everything their application believes about the chain arrives from one vendor who has changed the pricing. The data is public and out of reach at the same time, which is a strange way for a public record to end up.

The same shape shows up wherever a dataset outgrows the people who care about it. An offline copy of Wikipedia matters most where the connection is worst or the government is, and every copy of it still sits somewhere with a bill attached. Municipal records go dark when a domain lapses. A dataset a hundred papers were built on stops answering at the address all hundred of them cite. The file usually still exists on somebody's drive; what ends is the address everybody else had for it, which for most purposes is the same as the data ending.

Underneath both is a funding problem wearing a technical costume. Keeping something available indefinitely means somebody paying indefinitely, so preservation depends on an institution outliving the material it preserves, and the institutions with the longest horizons are rarely the ones holding the interesting data.

Swarm changes who is allowed to pay. Storage is prepaid, and anyone can add to any batch: paying to keep something alive is not reserved to whoever put it there. A chain's history could be kept by the people who read it, each contributing a little, none of them in a position to switch it off, and the address would not change when the payer did. Node operators would fetch the past on demand instead of every one of them carrying all of it.

For encyclopaedias and maps this already works, and there are copies on Swarm to read today. For a blockchain's history it does not work yet. That gap is the whole of this space: the mechanism is proven at one scale and unsolved at the scale that matters most.

Architecture

Split this space in two before anything else, because the two halves are at completely different stages. Public reference data is done: Wikipedia and OpenStreetMap are both on Swarm and readable now, and they fit well because each is made of a very large number of small files. Blockchain history is not done. It is the same primitives pointed at roughly a terabyte of data with query patterns those primitives do not natively answer, and every hard problem in this document lives on that side.

What is already solved, and proven by the live datasets. Every file's address derives from its own bytes, which gives deduplication nobody has to arrange: successive snapshots share every chunk that did not change, so a monthly republication costs roughly the size of the diff. A Mantaray manifest maps paths onto references so an uploaded directory tree resolves by URL path the way a static site does, and the manifest is stored as ordinary content itself; this is what makes large collections practical, since tar-based collection upload does not scale to millions of entries. Uploads can carry an erasure-coding redundancy level that adds parity chunks so content reconstructs when some chunks are unavailable, and an uploading node can pin its own chunks. Underneath it all a prepaid postage batch pays for every write, adding funds extends how long the content stays retrievable, expiry loses it outright, and raising depth buys capacity at the cost of lifetime. Topping up

is open to anyone rather than to the batch's creator alone, which is the specific property that lets the people who use a dataset fund it.

Problem one: content addressing does not answer the questions asked of chain data.

Content addressing answers “give me these exact bytes”. Blockchain data is queried by block number, block hash, transaction hash, address and log topic, and none of those are native. Feeds map a stable name onto a moving reference and manifests map paths onto content, which gets you part of the way: the research tooling puts three prefixes in a manifest so that a block number, a block hash and a transaction hash each resolve to a block bundle. Past that you are designing a data structure, and proximity-order tries are the candidate rather than the answer.

Problem two: the numbers do not work yet, and they are worth stating exactly. The full mainnet archive is roughly 25 million blocks and 3.5 billion transactions, which is about 3.55 billion manifest entries. An in-memory manifest tree over that many keys runs to hundreds of gigabytes of RAM, which no ordinary machine has, and the manifest's own chunk bytes come to something like 300 to 500 GB before any block data. At a realistic ingest rate of 100 to 500 chunks per second per node, emitting just those manifest chunks takes between 80 and 400 days on a single node. The research project's own conclusion about its own design is that it is a dead end past about ten million entries, and that the full archive needs a different architecture rather than an optimised version of the current one: an on-disk key-value store as the authoritative path-to-reference map with manifests demoted to a serialisation format, the three indexes split so the transaction index (which is about 99% of the entries) can be published on its own cadence, streaming bottom-up manifest emission that holds only one root-to-leaf path in memory, and several Bee nodes behind a dispatcher.

Problem three: the two systems do not address content the same way. Swarm chunks are addressed by a BMT hash and Ethereum's state trie is addressed by keccak256. Storing archive files as opaque blobs sidesteps this entirely, which is what the working tooling does and why bulk historical distribution is the tractable case. Putting individual trie nodes into chunks does not sidestep it, and needs either a mapping layer or a change to Bee. The proposal on the table wraps each trie node in a Single Owner Chunk whose identifier is the keccak hash, which would give constant-time lookup and inclusion proofs end to end. It is a proposal in a research repository, and should be read as a direction rather than a capability.

Problem four: latency separates the two use cases. Retrieving chunks over a network is slower than reading local disk. For bulk historical distribution that is acceptable, since nobody minds how a terabyte arrives. For block sync, which needs rapid sequential state access, it is not, and would need aggressive caching to become so. This is the line between “node operators fetch old state on demand” and “nodes run from Swarm”, and only the first is plausible today.

Problem five: nothing makes storing this data anybody's job. There is no natural incentive for Swarm nodes to hold blockchain history unless somebody funds it, which the research states plainly. This is exactly where open top-up stops being a nice property and becomes the mechanism the whole thing runs on, because the people who benefit from historical state being available are numerous, identifiable, and already paying for it in other ways. Turning that into a funded batch is a design problem nobody has finished.

Problem six: verification has to survive the trip. Era files are self-verifying through accumulator roots and individual chunks are BMT-verifiable, but proving that what came back is genuine Ethereum history needs the header chain for context and Merkle proofs against known headers. Anything less

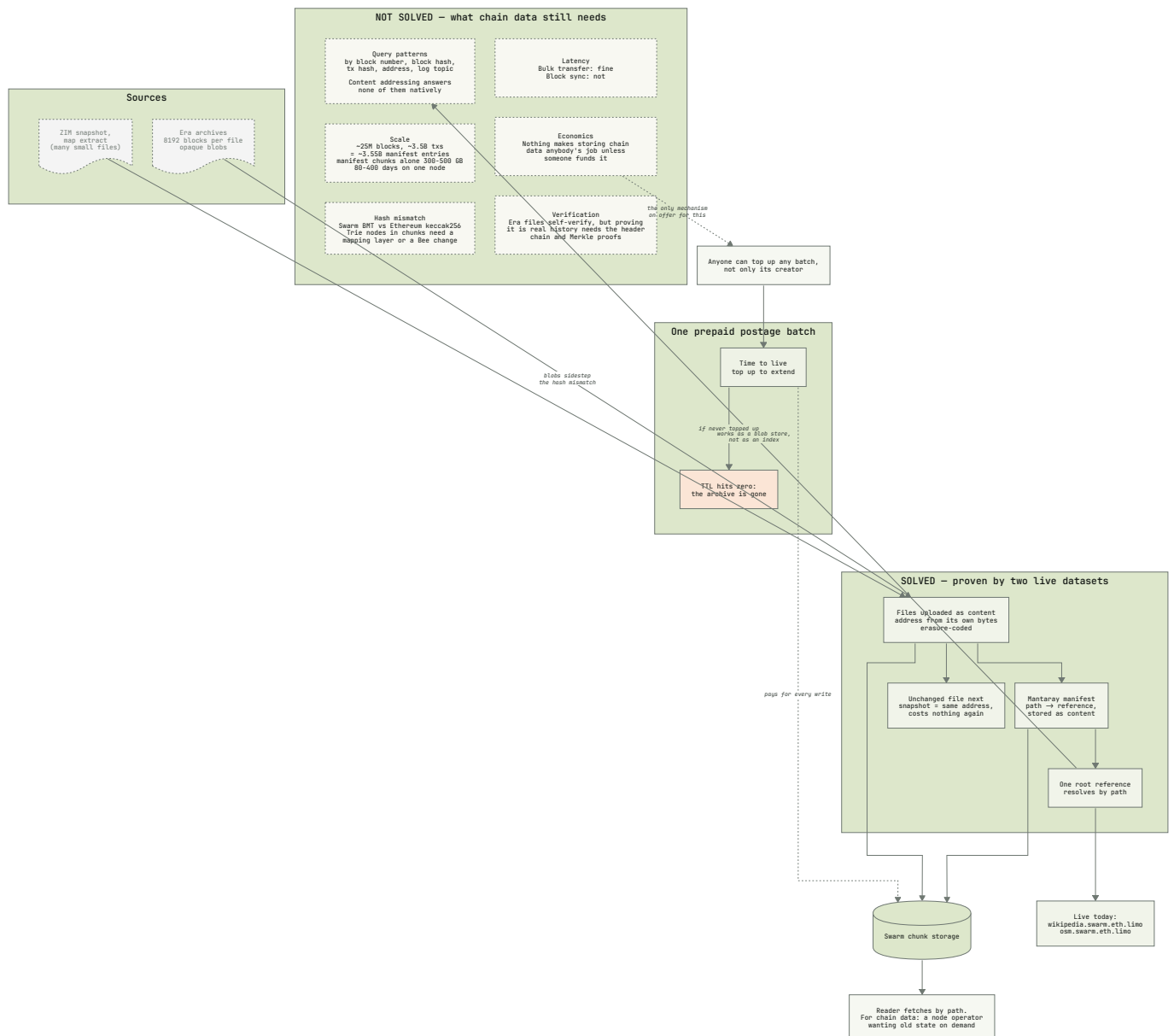
gives you data that is available without being trustworthy, which for this use case is most of the value gone.

PRIMITIVES

Content addressing Mantaray manifests Feeds Single Owner Chunks Erasure coding

Postage stamps, top-up and dilution

EXTERNAL DEPENDENCIES ethPanda0ps erae archives E2Store framing Kiwix / ZIM zimdump



Data flow, with the solved and unsolved halves separated. Files of any kind upload as erasure-coded content whose address comes from its own bytes, so an unchanged file in the next snapshot costs nothing again, and a manifest maps paths onto those references. That path is proven: two live datasets run on it. Chain data enters the same way as opaque era blobs, but the query patterns it needs – by block number, block hash, transaction hash – require index structures nobody has finished, at a scale where the manifest alone runs to hundreds of gigabytes. One prepaid batch underwrites everything, and anyone can top it up.

Development

The bulk-upload half is a known quantity and there is working code to read before writing any. Extract the archive to a directory tree, upload each file, build a mantaray-js manifest incrementally as you go, upload the finished manifest as ordinary content, and the reference it returns is the collection. Generating an index page at the root that redirects into the archive's real entry point is what makes the result browsable from one reference.

Traps on a first large upload, all found the hard way by the tool that put Wikipedia on Swarm. Run against a local Bee node, because gateways impose size restrictions a real dataset will exceed. Use manifests rather than tar-based collection upload. Keep the HTTP connection alive, since calling `uploadData` per file opens a new connection each time and leaves `TIME_WAIT` sockets accumulating until the machine runs out. Create a tag and watch it for upload and push progress. Note that `/bytes` defaults to deferred uploads, so a successful call does not yet mean the chunks are distributed and a fresh archive can be briefly unreachable elsewhere. The in-memory manifest has a ceiling; past it, build partial manifests and combine them. Parallel uploads are faster and raise the risk of exceeding time settlements and having to issue cheques for push data transfer.

Build the upload proxy once and reuse it on everything. Put it between the uploader and Bee. Cache successful responses from the content-addressed POST endpoints in a local SQLite database keyed by the body hash, the batch and the path, so re-uploading identical bytes short-circuits instead of making the round trip, which pays off on every resumed or retried run. Proxy the chunk-stream WebSocket with per-frame caching and drain acknowledgements in arrival order so cache hits cannot overtake still-unacknowledged misses. Retry connection failures with exponential backoff and, on the WebSocket path, reconnect and replay pending frames, which is safe precisely because content-addressed writes are idempotent. Two rules come with it: never cache the mutable endpoints, meaning feeds and stamps, and key the cache per upstream so pointing at a different node does not poison it.

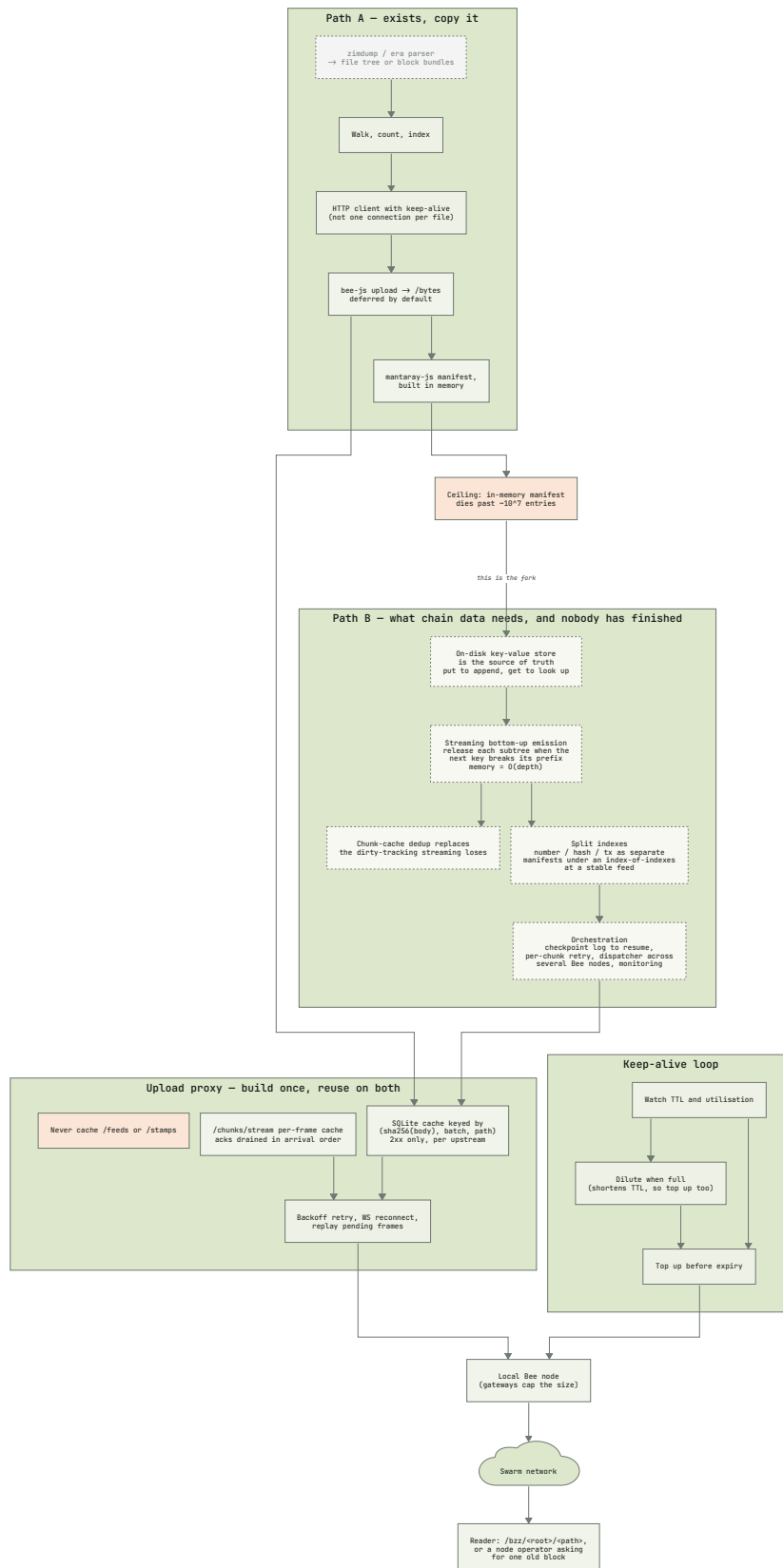
Then keep it alive, which is the ongoing job. Watch the batch's time to live and top it up before it runs out, since expiry loses content rather than degrading it. Watch utilisation and dilute when the batch fills, remembering that dilution shortens the lifetime and so usually needs a top-up alongside. If the archive is meant to be funded by the people who use it, the interesting work is making that legible: showing the current time to live, what a given contribution buys in days, and letting somebody contribute without first having to understand postage batches.

Everything above gets you a large static collection. Blockchain data is where the work actually is, and anyone taking this space should expect to build most of the following. Move the authoritative path-to-reference mapping into an on-disk key-value store so appends are a put and lookups are a get, with manifests emitted from it rather than being the database. Emit those manifests as a streaming, bottom-up pass over sorted keys, accumulating entries that share a prefix and releasing each subtree once the next key breaks it, so memory stays proportional to depth rather than to the dataset. Split the block-number, block-hash and transaction indexes into independent top-level manifests referenced from a small index-of-indexes at a stable feed, so the transaction index does not force a republication of everything else. Recover the deduplication that streaming emission loses by consulting the content-addressed chunk cache before every upload; unchanged subtrees hash identically and drop out for free. Then treat the upload itself as an operation rather than a script: a checkpoint log separate from the key-value store so a killed process resumes, per-chunk retry with backoff, several Bee nodes behind a dispatcher, and monitoring for chunk rate, queue depth, stamp

depletion and error taxonomy. The parsing side is comparatively easy and already exists: era archives carry 8,192 blocks per file, and a parser that keeps each block's raw RLP alongside the decoded number, hash and transaction hashes gives an append-friendly index where stitching on a later block is one file append rather than a rewrite.

TOOLS & REFERENCES

- bee-js library — <https://bee-js.ethswarm.org>
- mantaray-js, manifests for large collections — <https://github.com/ethersphere/mantaray-js>
- Postage batches, top-up and dilution — <https://docs.ethswarm.org/docs/develop/tools-and-features/buy-a-stamp-batch>
- Erasure coding — <https://docs.ethswarm.org/docs/develop/tools-and-features/erasure-coding>
- Pinning — <https://docs.ethswarm.org/docs/develop/tools-and-features/pinning>
- Feeds — <https://docs.ethswarm.org/docs/develop/tools-and-features/feeds/>
- FullCircle: Ethereum history on Swarm, research, era tooling and upload proxy — <https://github.com/snaha/fullcircle-research>
- Proximity order trie — <https://github.com/ethersphere/proximity-order-trie>
- Swarming-Wikipedia, a working ZIM uploader — <https://github.com/ldeffenb/Swarming-Wikipedia>
- Wikipedia on Swarm, live — <https://wikipedia.swarm.eth.limo/>
- OpenStreetMap on Swarm, live — <https://osm.swarm.eth.limo/>
- **External dependencies (not Swarm primitives):** ethPandaOps erae archives · E2Store framing · Kiwix and the ZIM format · zimdump



Component assembly, split the same way. The solved path walks a file tree, uploads through a keep-alive client, builds a manifest incrementally, and passes everything through a deduplicating proxy that caches content-addressed writes and replays frames after a reconnect. The unsolved path replaces the in-memory manifest with an on-disk key-value store, emits manifest chunks bottom-up as a streaming pass, splits the three indexes so the transaction index publishes on its own cadence, and fans uploads across several Bee nodes behind a dispatcher. A keep-alive loop watches the batch either way.

Reference implementations

Wikipedia and OpenStreetMap are the proof, not the territory. Both are on Swarm and readable now, at `wikipedia.swarm.eth.limo` and `osm.swarm.eth.limo`. They demonstrate that a very large collection of small files can live on Swarm and resolve by path, and they are why the solved half of this document can be stated as fact rather than as expectation. They are also finished work, which is precisely why they are not what this space is for.

Swarming-Wikipedia is the working uploader for that shape of data, BSD-licensed, written for a Fair Data Society hackathon bounty in partnership with Kiwix. It takes a Kiwix ZIM archive and a stamp batch and produces a browsable collection, and its stated limit is the size of the batch rather than the size of the snapshot. Its README is the most useful part, listing the traps in bulk uploading plainly. The Swarm Association also ran a 50,000 DAI prize for making Wikipedia unstoppable, with Kiwix.

FullCircle is the open frontier and the reason this space is unclaimed. It is research plus real tooling for putting Ethereum's execution history on Swarm so that node operators can reach historical state on demand instead of each carrying all of it. It parses ethPandaOps era archives into blocks and indexes, publishes a manifest layout where block number, block hash and transaction hash all resolve to a block bundle, runs a local Bee stack for testing, and ships the upload proxy this document recommends. Its research documents set out the six problems above and propose answers to several, including a staked bridge subnetwork of nodes running both an Ethereum client and a Swarm client, and single-owner-chunk wrapping of trie nodes. Those are proposals rather than implementations, and its own scale analysis says the full archive needs a fresh architecture. Somebody has done the hard thinking here and written down what remains; the remaining part is large and specific and nobody has finished it.