ARCADE

A coin paid for keeping data

Whitepaper: the chain, its storage economy, blobs, the ledger, borrowing, code on chain, and the fog-of-war storage plan

Draft of 2026-11-18. Describes the ARCADE testnet (*arctest*) as it runs at block 2,091, the mainnet rules set for block 0 on 2027-01-01, and the plans approved since. Every number is taken from the code (`crates/arcade-arc`, `crates/arcaded`, `crates/arcade-ledger`) or the design documents; where they differ, the text says which is built.

Status marks used throughout: **[BUILT]** running in the released code **[PARTLY BUILT]** some of it runs **[IN PROGRESS]** being built now **[PLANNED]** decided, not built **[OPEN]** not decided

Contents

Abstract 5

1. Motivation 6

1.1 What is new . . . 6

2. Overview 7

2.1 The parts, and where they live . . . 7

3. The chain 9

3.1 Hashes, keys and numbers . . . 9

3.2 The block header . . . 9

3.3 Block 0 and the genesis list . . . 10

3.4 The writer draw . . . 10

3.5 The proof of time and time keepers . . . 11

3.6 Difficulty . . . 11

3.7 Fork choice, reorg limit and finality . . . 12

3.8 Rules versions . . . 12

4. Coins, issuance, fees and pay 13

4.1 Transactions and script . . . 13

4.2 Issuance . . . 13

4.3 Fees . . . 13

4.4 Weights . . . 14

4.5 The pot, epochs and payouts . . . 14

4.6 Live numbers (testnet, 2026-11-19 01:52 UTC) . . . 15

5. Storage: what earns and how it is checked 16

5.1 Listed chains . . . 16

5.2 Segments . . . 16

5.3 Families . . . 16

5.4 The catalog . . . 17

5.5 Keeping the catalog honest: reveals and fraud proofs . . . 17

5.6 Sealing . . . 18

5.7 Registration . . . 19

5.8 Storage audits . . . 19

5.9 Serving checks . . . 20

5.10 Section limits per block . . . 20

6. Blobs 21

6.1 The record . . . 21

6.2 Classes, as built today . . . 21
6.3 Files bigger than a block . . . 22
6.4 Serving and the viewer cache . . . 22
7. Fog-of-war storage 23
7.1 The rules . . . 23
7.2 How it works . . . 23
7.3 Pricing . . . 24
7.4 Single blobs up to 100 MB . . . 24
7.5 Open decisions . . . 24
8. The ledger on ARCADE 26
8.1 The record envelope . . . 26
8.2 @tags . . . 26
8.3 A:// ops . . . 27
8.4 Repositories, tokens and the market . . . 28
9. Borrowing and the mesh 29
9.1 ARCADE frames (kind "ac") . . . 29
9.2 Waiting transactions . . . 29
9.3 Borrowing ARCADE . . . 29
9.4 Borrowed = installed . . . 30
9.5 Services over the mesh (svc) . . . 31
9.6 Relays . . . 31
10. Code on chain and releases 32
10.1 The release record . . . 32
10.2 Keeping and fetching . . . 32
10.3 Publishing and updating . . . 32
11. Economics 33
11.1 Supply . . . 33
11.2 What hosting costs, and the penny target . . . 33
11.3 The tension: issuance against fees . . . 33
11.4 The hosting oracle . . . 33
11.5 Who earns what . . . 34
12. Security and risks 35
12.1 Stated plainly . . . 35
12.2 Restart, reorg and failure behavior, component by component . . . 36
13. Status and open questions 37
13.1 Where things stand (2026-11-18) . . . 37

13.2 Open questions . . . 37

Glossary **38**

Appendix: parameters **39**

Appendix B: worked examples **41**

B.1 What a cheater loses . . . 41

B.2 Splitting the pot . . . 41

B.3 A difficulty retarget . . . 41

B.4 The lifetime of a blob, today and under fog of war . . . 41

B.5 Paying a tag in another coin (planned) . . . 42

B.6 A release reaching an arcade . . . 42

Abstract

ARCADE is a blockchain built into Arcade V1.0, the program every arcade on the mesh runs. Its coins are not mined by hashing and not staked by holding. They are earned by **keeping data**: the blocks of the chains the mesh follows (Pepecoin, Dogecoin, DigiByte, Litecoin, Bitcoin and others), the index journals built from them (inscriptions, tokens, names, the social ledger), and ARCADE's own blobs (app pages, pictures, video, code). An arcade seals each piece of data it keeps under its own key, registers the sealed copy on the chain, and answers storage audits drawn from block randomness. Its chance of writing the next block, and its share of every block's reward, follow its share of the network's **verified** storage.

Blocks come every 150 seconds. Each mints 50 ARCADE: 1 to the block's writer, 49 into a pot that, with every fee, is shared every 100 blocks among the holders of verified storage, in proportion to bytes held and split among the copies of each piece. Who writes a block is decided by a verifiable random draw weighted by storage, and when by a verifiable delay function (a proof of time) of the same kind Chia uses, so history can't be rewritten quickly. Every arcade checks every rule.

On top of the coin, ARCADE carries a **ledger**: @tags, profiles, follows, posts, messages, wallet proofs, likes, replies and tips (the A:// records), Git repositories, and signed releases of Arcade V1 itself, which every arcade keeps and updates from. Arcades that don't keep the chain **borrow** it from those that do, checking every header themselves.

This paper describes all of it as built, to the byte where it matters, and the decided next step: **fog-of-war storage**, in which data spreads to the arcades that view it or choose to keep it, with audited copy floors per class of data, rather than living on every node.

1. Motivation

The arcade mesh began as a network of programs that each ran full nodes of several small coins and their indexes. That work is useful to everyone (it is what lets a phone or a fresh install read an inscription or a balance without running a node) but nothing paid for it. Running a node costs disk, bandwidth and power; the operator of a VPS that relays, lends chains and serves pages pays a monthly bill and gets nothing back.

ARCADE turns that work into the coin's issuance. Three design goals follow from it:

- **Pay for real storage, provably.** Only bytes an arcade can be audited on count, and cheating (claiming data one doesn't keep) must always cost more than keeping it honestly.
- **No premine, no committee.** Every coin is minted by a block, every block's pay is computed by every arcade from the chain itself and enforced by a coinbase rule. There is no founders' allocation and no set of signers.
- **Reuse what arcades already do.** Transactions are Dogecoin's legacy format, so the wallet, signing, explorer and coin index of V1.0 work on ARCADE as on any other chain. Blocks travel over the same mesh links arcades already keep.

A second motivation came later: the social and app data the arcades produce (handles, posts, apps, videos) needs a home that every arcade reads alike and that doesn't depend on another coin's fees and politics. ARCADE became the **home chain** for that ledger (decided 2026-11-14), with Pepecoin testnet as the failover while ARCADE is stalled.

1.1 What is new

Idea	Where it comes from	What ARCADE does differently
Proof of storage with sealing	Filecoin, Chia	Sloth sealing per 127-byte leaf: unsealing is about 320 times faster than sealing, and leaves are independent, so reads are fast and there are no checkpoints to cheat with. Cheating is made expensive, not impossible.
Proof of time	Chia	The same class-group VDF (Chia's library), with time keepers serving every candidate of a height from one run.
Paying for public chain data	new	Paid data is fixed by the source chain: one right root per range, so nobody can list data of their own making.
Per-owner audits	new	Audits are sampled per arcade (4 replicas a period), so the chain's work grows with the number of arcades, not with what they keep.
Ledger + code on chain	the arcade's own history	The arcade's source, dependencies and programs live on the chain; arcades update from it.

2. Overview

The diagram follows a piece of data from a source chain to an arcade's payout. Every box is a rule every arcade checks.

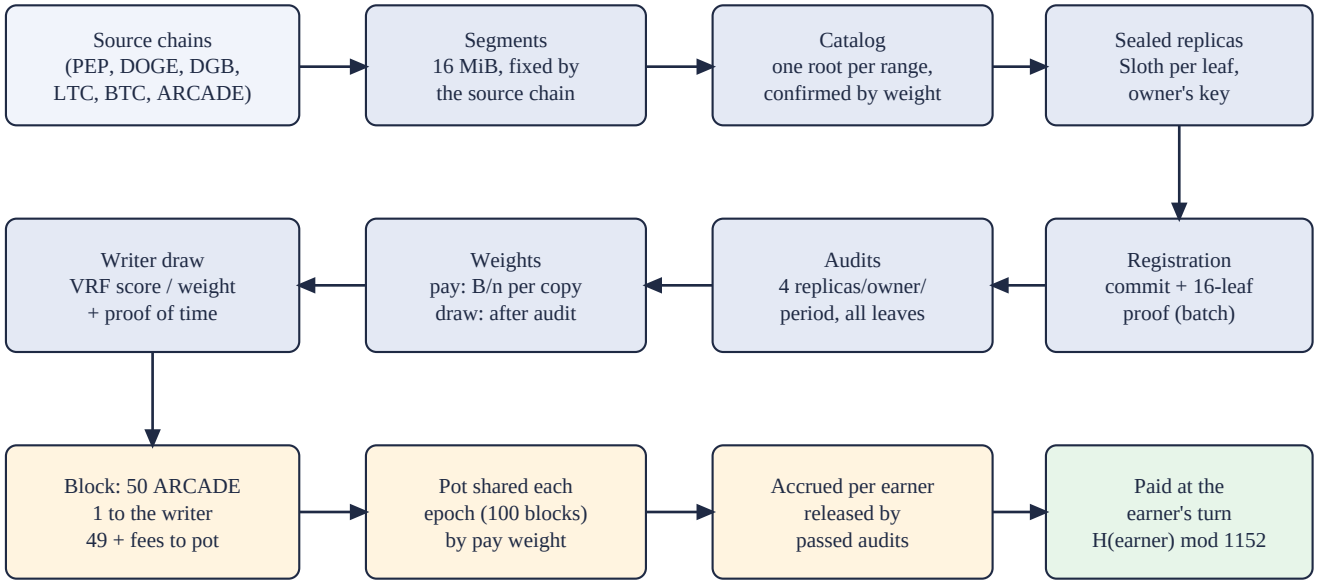


Figure 1. The storage economy. Sections 3 to 5 explain each box.

The second diagram is the fog-of-war plan for ARCADE's own blobs (section 7): data starts with its publisher and spreads to the arcades that view it or choose to keep it, never below an audited copy floor.

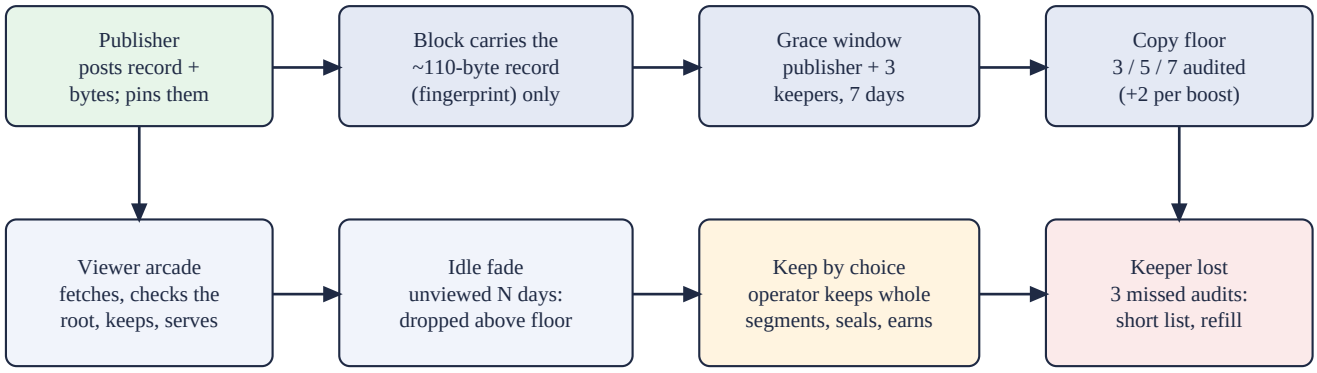


Figure 2. A blob's path under fog-of-war storage **[IN PROGRESS]**. Today every full arcade stores every blob (section 6).

2.1 The parts, and where they live

Part	What it does	Code	Status
Chain rules	blocks, draw, proof of time, fork choice, coins, script, pool	crates/arcade-arc/src (chain.rs, draw.rs, time.rs, coins.rs, script.rs, pool.rs)	**[BUILT]**

Part	What it does	Code	Status
Storage	segments, catalog, sealing, registry, audits, verdicts, fraud proofs	segment.rs, catalog.rs, sloth.rs, registry.rs; arcaded arcseg.rs, arcseal.rs, arcaudit.rs, arcserve.rs, arcfraud.rs	[BUILT]
Blobs	records, blob section, store, cache, pruning, files	arcade-arc blob.rs; arcaded arcblob.rs, arcsvc.rs	[BUILT]
Time keepers	one VDF run serving every candidate	arcaded arckeep.rs; arcade-arc time.rs	[BUILT]
Ledger on ARCADE	@tags, A:// ops 1 to 10, repos, tokens	crates/arcade-ledger (tags.rs, aproto.rs, repos.rs, engine.rs); arcaded arcledger.rs, asocial.rs	[BUILT]
Borrowing	light headers, lenders, borrowed = installed	arcaded arcborrow.rs, borrow.rs, lentrows.rs; arcade-arc light.rs	[PARTLY BUILT]
Code on chain	signed releases, updates from the network	arcaded arccode.rs, updater.rs; scripts/code-publish.py	[BUILT]
Services	the svc mesh channel (model marketplace)	arcaded svcmesh.rs	[BUILT]
Fog-of-war storage	demand spread, floors, keep by choice	branch in progress	[IN PROGRESS]
Hosting oracle	ARCADE per hosting unit	plan only	[PLANNED]

3. The chain

This section describes blocks, who writes them, how long they take, and which branch is the chain. Everything here is **[BUILT]** and runs on the public testnet with the real proof of time.

3.1 Hashes, keys and numbers

- **Tagged hash.** H(label; parts) = SHA-256(net tag || "/" || label || 0x00 || parts). Net tags: `ARCADE/arc` (mainnet), `ARCADE/arctest` (testnet). Each label's parts have fixed lengths, or one free-length part that comes last, so no two inputs collide.
- **Owner key.** An arcade's mesh identity, an ed25519 key. It signs blocks and storage items and makes the writer draw with ECVRF-EDWARDS25519-SHA512-TAI (RFC 9381); only the canonical proof encoding is accepted.
- **Coin keys.** secp256k1, as Dogecoin. Testnet addresses: P2PKH version 0x53 (start with *a*), P2SH 0x57 (*c*), WIF 0xd3. Mainnet: 0x17 (*A*), 0x1c (*C*), WIF 0x97.
- **Encoding.** Little-endian unless said otherwise. Merkle trees are RFC 6962 (leaf prefix 0x00, node prefix 0x01); an empty tree's root is 32 zero bytes.
- **Units.** 1 ARCADE = 100,000,000 units, 8 decimal places.

3.2 The block header

Fields in order (arcade-arc block.rs):

#	Field	Bytes	Meaning
1	version	u32	the rules version its height calls for (Net::rules_at); any other is refused
2	height	u64	one more than the block before
3	prev	32	id of the block before
4	time	u64	seconds since 1970; later than prev's, at most 120 s ahead of the checker's clock
5	tx root	32	Merkle root over the transaction ids, in the block's canonical order
6	storage root	32	Merkle root over the storage items' bytes, each starting with its tag
7	difficulty D	u128	set at each epoch's first block (section 3.4)
8	writer	32	the writer's owner key (ed25519 public key)
9	claimed weight W	u64	the writer's draw weight in bytes, checked when the block joins a branch
10	VRF proof	80	the writer's draw for this height
11	proof of time	len + output, len + proof	at most 512 bytes each (2,048 for up to 8 runs from rules version 2)
12	seed flag	1	0, or 1 followed by the draw seed's proof of time (first block of an epoch only)
13	blob ids root	32	from rules version 3: root over the blob ids the block brings (zeros when none)
14	signature	64	ed25519 by the writer over ("block"; id)

The **block id** is the double SHA-256 of everything but the signature. Because the id covers the body only through the roots and doesn't cover the signature, an arcade never remembers an id as bad for a wrong signature or a body that doesn't match: anyone can wrap a good header in junk. An id is remembered as bad only for what the id itself covers.

Storage item tags

Storage items in the storage section are ordered by group (tag >> 4), groups never going down. An unknown tag makes the block invalid.

Tag	Item	Tag	Item
0x10	audit opening	0x50	registration commit (batch)
0x11	transaction answer (to a challenge)	0x60	catalog entry
0x12	reveal answer	0x61	attestation
0x20	audit commit	0x62	fraud proof: bad header
0x21	re-audit request	0x63	transaction challenge
0x30	registration proof	0x64	equivocation proof
0x40	serving verdict	0x65	reveal demand

3.3 Block 0 and the genesis list

Block 0 is six numbers: time, hash of the closed list, hash of the final list, beacon 1, beacon 2, first difficulty. Its id is H("block 0"; those six). The first epoch's draw seed is H("draw seed 0"; closed list || beacon 2), so the final list can't steer it. Block 0 has no writer and no transactions; the first 50 ARCADE are minted by block 1. There is no premine.

The first-writer problem. On a chain where storage decides who writes, whoever registers first could shut everyone out. So arcades seal *before* genesis and a public list of replicas is closed before anyone can know what will be checked **[BUILT]** (genesis.rs, arcgenesis.rs; rehearsed as gennet2/gennet3):

When (UTC)	What
2026-12-21 to 12-27	Arcades send registration commits over the mesh, each with a work stamp; the list is public as it grows.
2026-12-28 00:00	The list closes; its hash is published.
2026-12-28 12:00	Beacon 1: the hash of the first Bitcoin block with header time at or after this moment, 6 deep. With the list hash it picks 16 leaves per replica and seeds a full audit.
by 12-29 12:00	Every listed replica posts its 16-leaf proof and its audit commit A.
2026-12-30 00:00	Beacon 2 picks the 4 leaves each replica opens.
by 12-30 24:00	Openings posted; replicas without a valid proof and passed audit are struck.
2026-12-31	Final list public for a day; the assembler's signed receipts let anyone prove a commit was left out, and it is put back.
2027-01-01 00:00	Block 0, carrying the final list's hash, both beacons, and the first code release's id.

The testnet (arctest) instead has two genesis writers of 1 GiB each (the project's first two arcades) and was restarted three times; its current block 0 dates from 2026-11-15 and carries the Pepecoin testnet ledger snapshot (322 tags, 121 tokens, 462 balances).

3.4 The writer draw

Storage decides **who** writes, the proof of time decides **when**. The draw is fixed for each 100-block **epoch**, so a writer's score for a height is the same on every branch and there is nothing to grind by withholding or re-trying blocks.

```
draw input   = "draw" message of (the epoch's draw seed || height), signed by the VRF
x            = the first 8 bytes of the VRF output, big-endian
L            = -log2((x + 1) / 2^64), rounded up to a multiple of 2^-32
score        = floor(L * 2^96 / W)              W = the writer's draw weight in bytes
steps        = floor(score * D / 2^96), at least 1
block seed   = H("seed"; VRF output || proof-of-time output)
```

The logarithm is base 2, not e: every score is scaled by the same constant, which D absorbs. No floating point is used anywhere. Lower scores are better; twice the weight gives twice the chance. **Example:** two writers with W = 1 GiB each draw u values whose L are 0.7 and 1.9; their scores are $0.7 \cdot 2^{96}/2^{30}$ and $1.9 \cdot 2^{96}/2^{30}$, so the first needs about 37% of the second's proof-of-time steps and usually writes the block.

The epoch's draw seed

The next epoch's seed is made from H("draw seed time"; this epoch's seed || the VRF outputs of the writers of blocks 1 to 80 of the epoch, in order), run through a proof of time as long as the epoch's blocks 66 to 80 took together (about 15 blocks, roughly 40 minutes on a fast core). The seed is H("draw seed"; output). The first block of the next epoch carries the proof; no other block may. From rules version 4 the proof may come in up to 8 consecutive runs, saved under arc/<net>/seed-pieces, so a restart carries on instead of starting over.

Why a proof of time on the seed: without it, someone with many keys on a secret branch could choose which key writes each block, compute the next seed for millions of choices in seconds, and keep the one that makes its next epoch fastest. That was the second external review's most serious finding; with the proof, each try costs about 40 minutes of a fast core.

3.5 The proof of time and time keepers

A block at height h is valid only with a proof of time of exactly *steps* steps, started from H("time"; id of block h–1). ARCADE uses Chia's class-group VDF with Wesolowski proofs (the chiavdf library, Apache-2.0): a 1,024-bit discriminant from the challenge, 100 bytes of output and 100 of proof. On the project's ARM test machine it runs about 270,000 steps a second on one core; a proof checks in about 2 ms (time.rs, ChiaVdf) **[BUILT]**.

Every candidate for a height starts from the same block hash, so **one calculation serves everyone**. A **time keeper** runs from the challenge to the lowest candidate's count, hands that candidate its proof, continues from that output to the next candidate's count, and so on. The k-th candidate's proof is the first k runs (rules version 2: up to MAX_POT_RUNS = 8 runs, MAX_TIME_BYTES 2,048).

Frame ("ac")	Fields	Meaning
cand	prev, w, writer, vrf	a writer's draw for the block after prev, taken only on the keeper's own tip, with a VRF that checks and a score that could win
tp	prev, steps, ol + output || proof	a proof for one candidate's count; checked in full before being passed on

Writers announce a draw that could be the height's best (score below 3× the best expected from the whole weight, about 19 heights in 20) and also run their own proof, writing with whichever is ready first. A draw that can't win is not announced and its own proof runs at the lowest CPU priority, so the chain moves even if every likely writer is away. Keepers are opt-in (arc.keeper) and unpaid **[OPEN]** (as in Chia).

3.6 Difficulty

D changes at the first block of each epoch: D' = D × target ÷ actual, where *actual* is the time from the last block of the epoch before the last one to the last block of the last one (from block 0 for the first epoch), *target* is 150 s per block in that stretch (15,000 s for a full epoch), and *actual* is clamped between ¼ and 4× the target. No block is left out between two measured stretches, which closes the Bitcoin-style time-warp trick (first review, finding 8).

Example. An epoch that took 18,750 s (187.5 s a block) gives D' = D × 15,000 / 18,750 = 0.8 D: the next epoch's proofs are 20% shorter.

3.7 Fork choice, reorg limit and finality

Rule	Value	Why
Best branch	most total difficulty (plain sum of D from block 1); then lower total score; then lower last id	every arcade reaches the same answer whatever it saw first
Reorg limit	REORG_LIMIT = 100 blocks (~4 h)	never leave the chain for one splitting off further back; with the 200-block draw wait, nothing registered on a secret branch can write there in time
Final for payments	FINAL_AFTER = 60 blocks (~2.5 h)	a 30% storage holder out-runs 60 blocks about 5 times in a million
Coinbase maturity	100 blocks	rewards can't be spent before a reorg could undo them
Checkpoints	each release carries a recent block id	a fresh or long-offline arcade can't tell an old secret branch from the chain by rules alone (cpnet passes)

Honest writers race on score, not speed (ARCADE issue #11): a writer whose block is still in its proof of time when a rival on the same parent arrives keeps going if its own score is lower, since its branch would win the tie-break; it gives up once the rival is built on. It never signs a second block on one parent, even after a restart (parents written on are kept in arc/<net>/wrote). Two signed blocks by one key on one parent are an **equivocation proof** (item 0x64): the writer's replicas are removed, its accrued pay goes to the pot, and its weight is gone for good.

Limits, stated plainly. The safe share is below half of the storage: about 40% by the second review's estimate. Against a secret branch, a faster time calculator multiplies its owner's storage (twice as fast counts as twice the storage), so the network needs at least one honest keeper near the fastest hardware. A small network is cheap to outweigh; the guards are many arcades sealing before genesis and release checkpoints.

3.8 Rules versions

The rules themselves are versioned, as Bitcoin and Dogecoin switch on upgrades, rather than checking app versions (which are easy to fake). Each network has a schedule of (version, start height); a header must carry the version its height calls for; a header of an unknown version is refused with "update this arcade". Every ARCADE frame carries PROTOCOL = 1 and the sender's schedule, so an arcade can warn "update before block N".

Version	What it adds	Testnet (arctest)	Mainnet (arc)
1	the base rules	before the 2026-11-15 restart	—
2	time keepers (up to 8 proof runs); the writer's 1 ARCADE	included	from block 0
3	blobs and the blob section; block/coins segments confirmed by a tenth with a 100-block challenge window	from block 0	not scheduled in params.rs (see note)
4	Shibacoin, Bellscoin, Luckycoin, Junkcoin listed; dogefeed journals (family 12) paid; repo pushes name their base; draw weight from the first passed audit (no 200-block wait); index entries confirmed by a tenth; the draw seed's proof in up to 8 saved runs	from block 1,000	never (testnet only)

Doc vs. code. The design documents say blobs, the ledger and code are on mainnet from genesis, but `params.rs` schedules mainnet as `rules: &[(2, 0)]`. Version 3 (blobs) still has to be added to mainnet's schedule before block 0.

4. Coins, issuance, fees and pay

4.1 Transactions and script

[BUILT] A transaction is Dogecoin's legacy format, version 1, at most 100,000 bytes, written the one canonical way. Every spendable output holds at least 0.001 ARCADE (DUST, a chain rule, not a relay habit); an OP_RETURN output burns what it holds. Lock times are block heights. At most 8,000 signature checks per block, and 1 MB of coin transactions per block (MAX_TX_BYTES).

Script is a strict subset of Bitcoin's, enough for what V1.0 signs: pay to key hash, pay to script hash, the exchange's pre-signed trades, the prize pool, bare multisig, a hash lock and a height lock. An opcode outside the list makes a script invalid even in an unexecuted branch; pushes are minimal; signatures are strict DER with low S; hash types are ALL, ALL|ANYONECANPAY and SINGLE|ANYONECANPAY. New opcodes come by rules version.

Transaction order is not the writer's choice (taken from an external blueprint): a block's transactions are ordered by H("txorder"; block seed || txid), each after the ones it spends from the same block (coins::canonical_order). A block in any other order is refused, so writers can't front-run or reorder for advantage.

4.2 Issuance

Quantity	Value	Source
Block reward	50 ARCADE, forever (no halving, no cap)	params.rs BLOCK_REWARD
Writer's share	1 ARCADE of the 50 (rules version 2+)	WRITER_REWARD
To the pot	49 ARCADE + every fee + anything forfeited	registry.rs
Block time	150 s: 24 an hour, 576 a day	BLOCK_SECONDS
Per day	28,800 ARCADE	576 × 50
Per 30-day month	864,000 ARCADE	17,280 blocks
Per year	about 10.5 million ARCADE	210,240 blocks
Premine	none	design decision 2026-11-10

Because issuance is constant in coins, it falls as a share of supply every year: 100% of the first year's supply, 50% of the second's, 33% of the third's, and so on.

4.3 Fees

Fee	Rule	Where
Transaction floor	0.001 ARCADE per 1,000 bytes (relay rule); wallets pay twice that when they choose	MIN_FEE_PER_KB; pool.rs
Blob bytes	0.01 ARCADE per started kB of *new* blob bytes (10× the floor); a boost pays it even without new bytes	blob.rs fee_for; chain.rs
Tag check-in	at least 0.1 ARCADE into the pot (keeps a tag alive)	ledger tags, kind 20
Where fees go	all of them into the pot, shared by weight; none to the writer (no fee sniping)	§14.18

Worked example. A 2 MB picture as a blob costs ceil(2,000,000 / 1,000) × 0.01 = 20 ARCADE plus the transaction's own fee (a ~300-byte transaction at 0.002/kB, about 0.0006 ARCADE). The project's game ASHVALE published 128 modules in one transaction for 39.2 ARCADE.

4.4 Weights

Two weights are kept apart on purpose **[BUILT]**:

- **Pay weight** (what the pot is shared by): for each registered replica that hasn't missed its last audit, from the block holding its registration proof, the segment's bytes split among all its registered replicas: w = B / n. Plus each genesis earner's fixed bytes.
- **Draw weight** W (what writes blocks, lists and attests catalog entries, and makes serving checks): the same replicas, but each only from DRAW_WEIGHT_WAIT = 200 blocks after its first passed audit (0 on the testnet from rules version 4), and only while its audits pass.

Example. A 16 MiB segment held by 4 arcades gives each 4 MiB of weight; one held by 1 arcade gives it all 16 MiB. Every held segment pays the same per byte however many hold it, so the pay pulls disks toward data the network lacks. Registering one segment under several keys requires a separately sealed copy for each, so it buys a larger share only with real disk.

4.5 The pot, epochs and payouts

At every block whose height is a multiple of 100 (EPOCH_BLOCKS) the pot is shared by pay weight among every earner: genesis earners by payout address, and every owner (an arcade's key, for all its replicas together). Shares are rounded down to whole units; the leftover units go one each to the largest remainders, ties broken by lower address. Each share is **accrued**, not paid. If nobody has weight, the pot carries over.

Each earner has a **turn**, turn = H(earner) mod 1,152 (registry::turn): one block in every audit period. The coinbase of that block pays the earner everything released, in one output. Turns spread payouts over the period, so each coinbase pays about 1/1,152 of the earners, which is what keeps payouts small at 100,000 arcades (about 87 a block).

At the owner's turn	Result
every audit of the period passed, serving not held	paid in one output to its payout address
audits still open	paid if they pass before then
no audit yet (registered after its last turn)	nothing: this turn draws the first audits; the next pays
an audit missed	the period's accrued pay goes back to the pot
5+ serving verdicts, most "not served"	held; paid at the next turn that isn't held; forfeited the third time in a row
under 0.01 ARCADE (MIN_PAYOUT)	waits for a later turn

A coinbase starts with exactly the payouts owed, in address order, then the writer's 1 ARCADE; a zero-value OP_RETURN may follow. Its outputs are spendable 100 blocks later. Genesis earners need no audit.

Worked example: switch-on to first coin

Block	What happens
~400	An arcade keeps Pepecoin testnet in full; Earn ARCADE is on; sealing runs block by block, 100 blocks below the coin's tip.
~520	The first 12 block segments are sealed; they were listed long ago by the genesis writers.
545 / 556	The batch commit goes in at 545; the seed of block 555 picks 16 leaves; the proof goes in at 556. Pay weight from here.
600, 700, ...	Each epoch's pot (~4,900 ARCADE plus fees) is shared by weight; this arcade's share accrues.
1,497	First turn (its owner key hashes to 345 mod 1,152): no audit covers the period, nothing is paid; 4 replicas are drawn for audit.
e.g. 1,600	A picked replica commits in 1,601–1,620 and opens in 1,612–1,631; passed.

Block	What happens
2,649	Second turn: all audits passed; the coinbase pays everything accrued from 600 to 2,600.
2,749	That output can be spent.

From registration proof to first coin: 1,152 to 2,303 blocks (2 to 4 days), depending on where the proof falls before the turn.

4.6 Live numbers (testnet, 2026-11-19 01:52 UTC)

Quantity	Value
Height	2,091
Difficulty D	18,136,255,301,591,064
Total difficulty	29,444,757,921,826,454,817
Unspent coins	39,374
Pot (undistributed)	4,809.549318 ARCADE
Total weight	2,112,661,858 bytes (about 1.97 GiB)
Owners with registered replicas	15 (plus 2 genesis earners)
Rules	version 4 known and in force; protocol 1; no outdated peers

5. Storage: what earns and how it is checked

Paid data is **public chain data that is fixed once written**, so a sealed copy never goes stale, and its right content follows from the source chain alone, so nobody can be paid for data of their own making. This section is **[BUILT]** and runs on the testnet.

5.1 Listed chains

Network	Listed chains (params.rs Net::listed / listed_v4)
Mainnet arc	arc, pep, peptest, doge, dgb, flop, wjk, kot, ltc, btc
Testnet arctest	arctest, pep, peptest, doge, dgb, dgbtest, flop, wjk, kot, ltc, ltctest, btc; from rules version 4 also shic, bel, lky, jkc

Listing is read at a height (Net::listed_at): a catalog entry is judged at its block's height. Adding a chain is a rules version. Every ARCADE node keeps the **headers** of every listed chain it follows (whole, or headers only in light mode), so fraud proofs about blocks can be judged.

5.2 Segments

- **Ranges.** A chain is cut into 10,000-height ranges (RANGE) from height 0.
- **Segments.** A segment starts where the previous one ended and takes whole ranges until its blocks hold at least 16 MiB (SEGMENT_BYTES; the table isn't counted). Ranges left at the end wait. A segment never changes once cut. A segment is listed only when its last block is at least 100 source blocks deep.
- **Bytes.** For heights first..=last: a table of one u64 per height (little-endian) giving where that block starts, counted from the segment's start; then each block as its chain sends it, in height order.
- **Root.** The Merkle root over those bytes in 127-byte leaves, the last zero-padded. A run of leaves with one path proves any stretch of a segment.
- **Too long is provable from one table row:** if the block bytes before the first block of the last range already reach the full size, the segment should have ended earlier.

5.3 Families

#	Family	Holds	Paid
0	blocks	every raw block in the range	always
1	ord	per-block journal of inscription rows	when journals run from the index's start
2–8	ledger, profiles, messages, names, tokens, maps, tags	per-block journals of each index	same
9	coins	the coins each block spends	always, made from blocks
10	blobs	the ARCADE chain's Permanent and Release blobs	ARCADE only
11	collections	(kept, unlisted)	never: not built the same on every arcade
12	dogefeed (D://)	per-block journal	testnet from rules version 4

An index journal record for a height is: count, then for each row the block touched, ordered by (table, key): table name, key, and "gone" or the value after the block. Writes inside a block collapse to the value after it, so live and backfilled indexes make the same bytes. Indexes without a journal family (addresses, collections, runes, dunes)

are never paid.

5.4 The catalog

Which segments exist, as every arcade agrees. Catalog items are in a block's storage section (at most 64 per block), so a branch change undoes them block by block.

Catalog entry (tag 0x60)

Field	Meaning
chain	V1.0 key of a listed chain
family	0–12 as above
first, last	heights; whole ranges
bytes	at least the table plus a full segment
manifest	a fetching hint (compressed pieces can differ between library versions); not in the id
root	Merkle root over the raw bytes, 127-byte leaves
poster	owner key with draw weight
signature	the poster's

Segment id = H(chain, family, first, last, bytes, root). An entry is taken only if, in order: its chain is listed; its family exists; first..=last are whole ranges; bytes is at least table plus a full segment; the signature is the poster's; the poster has draw weight; the id was never removed; it isn't waiting already; its span is open; and the poster doesn't stand behind another entry for the same first height.

Attestation (tag 0x61) and confirmation

Offering is attesting. An attestation {segment id, attester, signature} is taken if the entry waits, the attester has draw weight, and it stands behind no other entry for that first height. An entry is **confirmed** in the block that brings the weight behind it over the threshold:

Family	Threshold	Challenge window
blocks, coins (rules v3)	more than a tenth of draw weight	100 blocks (CHALLENGE_WINDOW) before replicas may register
index families (mainnet)	more than half	none needed
index families (testnet, rules v4)	more than a tenth	100 blocks

5.5 Keeping the catalog honest: reveals and fraud proofs

Item	Tag	Who	What	Outcome
Reveal demand	0x65	a key with draw weight, one open at a time	up to 64 kB of a listed segment's leaves	unanswered in 100 blocks: segment and its replicas removed; answered: the demander loses draw weight for one audit period
Reveal answer	0x12	any holder	those leaves with their path to the root	refutes the demand
Bad header	0x62	anyone keeping the source chain	a table row and a header of one height, as runs of leaves with paths	if the header isn't the source chain's (100 deep): segment removed
Transaction challenge	0x63	a key with draw weight	table rows, the header, one transaction's bytes (≤64 kB) said not to be in the block	unanswered in 100 blocks: removed
Transaction answer	0x11	any holder	the transaction covering the challenged byte, whole, with its Merkle path to the header root	refutes; challenger silenced for a period

Item	Tag	Who	What	Outcome
BadCoins, BadOrd	—	watchers	a wrong coins record; a wrong one-piece inscription row	entry removed

Index journals other than ord's one-piece rows have no fraud proof of their own **[OPEN]**: they rest on more than half of the draw weight confirming each root.

5.6 Sealing

Chain data is public, so three cheats are open: many names on one disk, keeping nothing and fetching on demand, and recomputing on demand. A sealed copy is encoded with the owner's key: slow to make, fast to read, useless under another key.

```
leaves    d_i = 127-byte pieces of the segment's raw bytes
mask      m_i = H(owner key || segment id || i)
seal      s_i = sloth(d_i + m_i)       one modular square root mod p = 2^1024 - 105 (p = 3 mod 4),
                                        with Sloth's rule for non-residues so it is a permutation
unseal    d_i = (+/- s_i^2) - m_i      one squaring (minus when s_i is odd)
sealed leaf = 128 bytes (0.8% larger); replica root = Merkle root over the s_i
```

Measured on the project's ARM machine, first code	Value
Sealing	1.03 core-hours per GiB, once (about 1,023 modular squarings a leaf)
Unsealing	11.6 s per GiB on one core (about 320× faster); a 1 MiB read about 11 ms; about 1.4 µs per leaf
Audits	each sealed GiB is read through once every 2 days on average: 10–15 s of disk and CPU per GiB
All of Bitcoin's blocks (~650 GB)	about 3.5 days on 8 cores, in the background

Each leaf stands alone: sealing runs on all cores, any leaf reads back alone, and there is no chain of leaves to shortcut with checkpoints. The sealed copy is the only copy on disk; the arcade reads and serves from it (one copy, by design).

Why cheating doesn't pay

No fast-unsealing scheme is cheat-proof mathematically: whoever has the plain data can always recompute a sealed piece. So the design rests on one measurable claim: **every sealed piece an arcade doesn't store costs it a fixed amount of CPU work each time it is audited, far more than the disk it saves, and nothing is paid until an audit passes.**

Per GiB per day	Cost (estimates)
Storing it honestly	about $0.00001
Recomputing for each audit on a PC (1 core-hour every 2 days)	about $0.001 (about 100×)
Recomputing on graphics cards (reviewer's estimate)	about $0.0003 (about 30×)

The weak point is special hardware: a custom chip could narrow the gap. The audit period and prime size are rule parameters, so an upgrade can widen it again; and even then the cheater pays energy on every audit while every claimed copy lowers pay per copy.

5.7 Registration

A registration takes two steps, so the registrant can't choose what gets checked **[BUILT]** (registry.rs):

Commit (tag 0x50), per owner, in batches

Field	Bytes	Meaning
owner	32	mesh key
payout	var	ARCADE address; where all the owner's pay goes from then on
replicas	64 each, 1 to 4,096	(segment id, replica root) pairs
anchor	32	a block at most 100 back on the branch (ANCHOR_AGE)
nonce	8	work stamp: the digest must have at least 20 leading zero bits (STAMP_BITS), about a core-minute
signature	64	owner's, over one digest of it all (so stamp work is the same for any batch size)

134 bytes plus 64 per replica. One batch per audit period: a commit is refused until 1,152 blocks after the owner's last proved batch. When more than 200 commits wait, the ones with the most stamp work go first, so filling the places with junk gets dearer the longer it lasts.

Proof (tag 0x30)

The seed of the block 10 after the commit's (CHALLENGE_DELAY) picks 16 leaves (REGISTRATION_LEAVES): draw k = H(seed || owner || k) mod (the batch's leaves), the batch's replicas laid end to end in order. The proof is due within the 20 blocks after (ANSWER_WINDOW). For each leaf it shows s_i with its path to the replica root, d_i with its path to the catalog root, and that unsealing s_i with the owner's key gives d_i. The whole batch registers, or none. At most 8 proofs per block (MAX_REGISTRATION_PROOFS), about 4,600 segments (4.5 TiB) a day.

5.8 Storage audits

Checking a few leaves isn't enough: a cheater would recompute just those. An audit makes the owner answer for **every leaf at once**. Audits are sampled per owner, so the chain's work grows with the number of arcades:

Step	When	What (registry.rs, arcaudit.rs)
Pick	at the owner's turn	the turn block's seed samples AUDIT_SAMPLE = 4 replicas by bytes (one drawn twice is audited once), each with a picking block 1 to 1,092 blocks later (PICK_SPAN)
Commit (0x20)	in the 20 blocks after the picking block	A = Merkle root over audit_leaf(seed, i, s_i) for every sealed leaf, signed over the picking block's id: one read through the sealed copy; the fresh seed means A can't be precomputed
Open (0x10)	the seed of the block 10 after the commit's picks 4 leaves (AUDIT_OPENINGS); due in the 20 blocks after	each leaf: s_i with its path to the replica root, its entry's path to A, d_i with its path to the catalog root, the unsealing (about 10 kB)
Decide	at the next turn	all passed: the period's pay is released; any not passed: forfeited to the pot, and the missed replica's weight is 0 until it passes again
Re-audit (0x21)	any time after a miss	a signed request counts as a pick; passing counts the replica again at once (the period's pay stays forfeited)
Removal	3 misses in a row	MAX_MISSED_AUDITS: the replica is removed; it can register again

Why a sample is enough. An owner missing a share f of its sealed bytes passes a period only with chance $(1–f)^4$, and each failure costs the slice of everything it holds. Keeping 90% and claiming 100% loses about a third of all pay; keeping half loses about 94%. Answers go first in the storage section and windows are 20 blocks, so a full block or one unfriendly writer can't make an honest arcade miss. The audit period stretches by rule (not by a

writer's choice) when a block's scheduled openings would fill more than three quarters of the 4 MB section, at about 100 TiB of sealed storage **[OPEN]** (combined proofs are needed before then).

5.9 Serving checks

Audits prove data is on disk; serving checks show the arcade hands it out **[BUILT]** (arcserve.rs):

- A key with draw weight is a secret checker of an owner at a block when VRF(its key, (block seed, owner)) falls under a threshold giving each owner about SERVE_CHECKS = 6 checks per audit period. Nobody can tell until it reports.
- The replica checked is H(seed || owner) mod (the owner's bytes) over its replicas in id order. The checker asks, through the ordinary block fetch (over the owner's mesh link when there is one, else its offer), for about 8 MiB of plain blocks (SERVE_PIECE_BYTES) and compares them with its own copy.
- It posts a signed verdict (tag 0x40) within 20 blocks: served within 60 s (SERVE_SECONDS) or not. No way to reach the owner from this checker means no verdict.
- At the owner's turn, with 5 or more verdicts (HOLD_VERDICTS) since the last turn and most "not served", its pay is held; held 3 turns in a row (HOLD_FORFEIT), forfeited.
- Limits: a verdict is a report, not a proof; checkers are unpaid; an owner that keeps links only to friends is checked only by them. This is the weakest guard; the audits don't depend on it.

5.10 Section limits per block

Order	Item	Size each	Per block
1	audit openings; answers to reveals and challenges	~10 kB; ≤64 kB	what fits
2	audit commits, re-audits	~120 bytes	no separate limit
3	registration proofs	~30 kB	8
4	serving verdicts	~220 bytes	1,000 (MAX_VERDICTS)
5	registration commits	134 + 64/replica	200 (most work first)
6	catalog entries, attestations, demands, fraud proofs	100 B to a few kB	64
—	whole storage section		4 MB (MAX_STORAGE_BYTES)
—	blob section (rules v3)		4 MB (MAX_BLOB_BYTES)

6. Blobs

Blobs are ARCADE's own content: app pages, pictures, video, code packs, long texts. They are named by a small record in a transaction while their bytes ride in the block's own blob section **[BUILT]** (rules version 3; arcade-arc blob.rs, arcaded arcblob.rs).

6.1 The record

A transaction output OP_RETURN with one minimal push:

Offset	Bytes	Field	Meaning
0	4	marker	"ablb"
4	32	root	RFC 6962 Merkle root over the bytes in 127-byte leaves (the sealing leaf), the last zero-padded
36	8	length	u64 little-endian; 1 to 4,000,000
44	1	class	0 Permanent, 1 Release, 2 Temporary, 3 Boost
45	1	encoding	0 plain, 1 gzip (readers undo it; the root is over the stored bytes)
46	≤64	type	UTF-8 content type, for example image/png

About 110 bytes whatever the blob's size. The **id** is H("blob"; root || length), so contents that differ only in trailing zeros have different ids, and the same bytes always have the same id on a network.

- **Stored once:** a block's blob section holds exactly the blobs its transactions name for the first time on the chain, in record order. A record of a blob the chain has is just a name for it.
- **Committed:** the header's blob root is the Merkle root over the ids the block brings.
- **Paid:** a transaction bringing new blob bytes pays at least ceil(bytes/1,000) × 0.01 ARCADE (fee_for), into the pot.
- **Carried:** a block younger than blob_window (30 days) without all its blob bytes is refused. Older blocks may travel without bytes.
- **Limits:** 4 MB per blob and per block.

6.2 Classes, as built today

Class	Meant for	What every full arcade does today (arcsvc.rs prune_blobs)
0 Permanent	collectibles, documents, owned things	kept for good: the plain file is dropped after 30 days once a sealed copy exists here or 5 replicas are registered; the sealed copy stays
1 Release	app releases; a line is sha256(type || 0 || publisher script)[..16]	the newest 5 of each line stay plain; older ones become sealed-only
2 Temporary	posts, clips	deleted after 90 days (blob_temporary); not in the sealed blob family, not paid
3 Boost	paying again to keep a blob longer	every full arcade keeps the plain copy 90 days past the boost's block; a blob a boost brings first is Temporary

Pruning runs every clamp(blob_window/4, 5, 300) seconds (300 s on the real networks). An **archive** switch (arc.archive, API only, off by default) keeps everything plain. Permanent and Release blobs go into blob-family segments (family 10) and are sealed, audited and paid like blocks.

6.3 Files bigger than a block

A **file manifest** is a blob of type application/vnd.arcade.file+json: {"arcade": "file", "type", "size", "sha256", "parts": [[blob id, length], ...]}, up to 1 GiB (MAX_FILE_BYTES). It is valid only if its parts add up to its size and each is a blob the chain holds. Readers serve byte ranges part by part, fetching missing parts from linked arcades and keeping them in the viewer cache.

6.4 Serving and the viewer cache

- **/content/<blob id>** answers with the same policy as ord inscriptions (inline scripts, loads from this arcade); a gzip blob goes out with Content-Encoding: gzip. Manifests are served with byte ranges.
- **Mesh frames:** "bl" {id, seq, end} + bytes sends a waiting blob ahead of its transactions; "bq" asks for 48 kB pieces (window 16, 20 s timeout), answered from the plain store, the sealed copy, or the cache.
- **The viewer cache** (decided 2026-11-14): a blob an arcade lacks is fetched from linked arcades when viewed, checked against its id, kept in blobs/cache/ and served. Default 2 GB (Options → Blob cache; 0 turns it off), least recently viewed dropped first. Since v0.1.130 an arcade that only borrows ARCADE has a cache too, serves from it, and asks up to 6 other linked arcades when no lender has a blob.
- **Waiting blobs** (of pool transactions) are held in memory, at most 64 MiB (MAX_WAITING).
- **Git** is stored as packs (git pack-objects under 3 MB, one block's blobs each) served over git's dumb HTTP protocol at /git/@owner/name.

Doc vs. code. §15.1 of the design says a blob may be dropped once "5 audited replicas" hold it; the code counts 5 *registered* replicas. It plans Boost as a copy count; the code gives every node a 90-day timer. Release's "reference-counted chunks" weren't built. Step 4 (the cache) lists no passing network test.

7. Fog-of-war storage

[IN PROGRESS] Decided 2026-11-18 and being built on a branch with a closed-network test; switched on later by a rules version at an ARCADE height. In the project lead's words: data on ARCADE should be "somewhat redundant, but not on every node... like a fog of war in RPG games, where the map reveals itself the deeper you go into the world". Data is saved on an arcade only once it is viewed through it or chosen to be kept, and fades when nobody looks.

7.1 The rules

Rule	Status
No arcade stores all of the chain's data; it stays redundant across the network	**[PLANNED]**
Fog of war: an arcade keeps a blob's bytes once it is viewed through it, and serves it from then on	**[PARTLY BUILT]** (viewer cache)
Any arcade lacking a blob fetches it over the mesh from one that has it; both then serve it	**[BUILT]** for linked arcades (v0.1.130)
Data not viewed or served through an arcade for N days is dropped there (above the floor)	**[IN PROGRESS]**
The original publisher keeps its own blobs indefinitely	**[IN PROGRESS]**
Permanent data is kept only by arcades that choose to; they seal it and are paid	**[IN PROGRESS]**
Operators keep a portion, by demand or by choice; portions are split among arcades	**[IN PROGRESS]**
Copy floors: at least 3 copies, more for higher-cost classes	**[IN PROGRESS]**
Copy floors count only registered, audited replicas (the existing audits)	**[IN PROGRESS]**
A lost keeper is replaced automatically	**[IN PROGRESS]**
Single blobs up to 100 MB once blocks carry fingerprints only	**[PLANNED]**

7.2 How it works

Blocks carry fingerprints, not bytes

Every arcade keeps every block and every ~110-byte blob record, so the chain stays complete and anyone can check data when it arrives. From the switch-on height, a block is valid with blob records only: the "carry your bytes within 30 days" rule is relaxed. This is the one consensus change; before the height, old behavior holds (every full arcade takes the bytes).

Who keeps the bytes

Keeper	Keeps	For how long	Counts for the floor	Paid
Publisher	what it published (a "mine" mark)	indefinitely	once sealed and registered	yes, if sealed
Chosen keeper	whole segments the operator picked (Options "Keep a share of the chain", or arc.keep {mb | segments | off})	as long as it chooses	yes (audited replicas)	yes
Grace keepers	new blobs: the publisher plus 3 more arcades	7 days (default)	yes	yes
Viewer	what was viewed through it (the viewer cache)	until unviewed N days (default 30) or the cache is full	no: a bonus for fetching	no

Copy floors

An arcade never drops data whose known copies are at or below its class floor. Copies are counted from the registry (registered replicas whose audits pass), never from self-reported announcements, which are used only to find holders to fetch from.

Class	Floor (default)	Price
Temporary	3 copies	base rate × 3, for 90 days
Release	5 for the newest 5 of a line	base × 5
Permanent	7 copies	base × 7, prepaid 12 months to start
Boost	+2 copies while it pays	base × 2 per 90 days

Because the price scales with the floor, each keeper earns the same per GB-month whatever the class. Temporary data gets its own sealed, audited blob family that expires, so its floor of 3 is audited too. Keep-by-choice works by whole segment, matching how audits work.

Keeper replacement

A keeper that goes down misses its audits; after 3 misses (MAX_MISSED_AUDITS) its replica stops counting. The segment drops below its floor and goes on a **short-of-copies** list. Arcades with free room in their chosen share take short items first (ranked by H(node id || blob id) so they don't all rush), fetch from a remaining holder, check, seal, register, and start earning. When a returning keeper pushes a segment above the floor, the newest extra copy may drop. A short restart causes no re-copy. If no arcade has room, the item stays short and shows a warning; nothing can force an arcade to keep data. Scarce data pays its keepers more (weight B/n), which is the pull.

7.3 Pricing

The project's target: at 1 ARCADE = $0.01, every VPS that keeps data is paid twice its cost each month. First figures (to confirm with the real bills): the East Coast relay VPS about $7 a month (2 vCPU, 2 GB RAM, 30 GB disk) and a 6 vCPU / 11 GB / 193 GB VPS about $8.

```
paid twice over:      2 x ($7 + $8)       = $30/month = 3,000 ARCADE/month
storage, per copy:    2 x $7 / 30 GB      = $0.47 per GB-month
                                          = about 47 ARCADE per GB per copy-month
(193 GB VPS alone:    2 x $8 / 193 GB     = about 8 ARCADE)
Temporary, 3 copies, 3 months:   47 x 3 x 3  = ~420 ARCADE/GB (0.42 per MB)
Permanent, 7 copies, 12 months:  47 x 7 x 12 = ~3,950 ARCADE/GB (~4 per MB)
today's flat blob fee:           0.01 per kB = 10 ARCADE per MB
```

7.4 Single blobs up to 100 MB

[PLANNED] Once blocks carry fingerprints only, the 4 MB per-block cap stops limiting data, so MAX_BLOB_BYTES can rise to 100 MB with a per-block limit on records and fees instead. Needed: streaming to and from disk (today view() returns the whole blob in memory and waiting blobs live in 64 MiB of RAM), range fetches proven against the root (127-byte leaves make any range provable), sealing and audits across several 16 MiB segments, and pricing by the copy floors. Until then, files up to 1 GB use manifests.

7.5 Open decisions

- Exact floors per class (3 / 5 / 7, +2 per boost, proposed).
- N, the days before unviewed data fades (30 proposed).
- The grace window for new data (publisher + 3 for 7 days, proposed).

- How long Permanent prepays (12 months renewable, or long enough to count as forever).
- The real VPS costs, and the issuance question in section 11.

8. The ledger on ARCADE

ARCADE carries the arcade network's shared records: @tags, profiles, follows, posts, messages, wallet proofs, reactions, Git repositories, tokens and the market. It became the **home chain** for that ledger on the testnet in phases from 2026-11-15; at mainnet launch every arcade moves to ARCADE main, whose block 0 carries the Pepecoin mainnet ledger by snapshot. While ARCADE is stalled, writes fail over to Pepecoin testnet (test chains only; mainnet coins only by explicit pick).

8.1 The record envelope

Every ledger record is an **arcd** record: an OP_RETURN output carrying "arcd" followed by an AnyData payload. On ARCADE a record of up to 1 kB is one OP_RETURN (the older multi-output Class B form isn't used). Its **author** is the transaction's sender: the key that signs its first input. There is no registrar.

Offset	Bytes	Field	Value
0	4	marker	"arcd"
4	2	version	u16 big-endian
6	2	type	u16 big-endian: 200 (AnyData)
8	…	data	"INSC" ledger ops (tags, tokens, ...), or "A://" ops

8.2 @tags

[BUILT] A tag is 2 to 24 characters of a–z, 0–9 and _; a reserved list (admin, arcade, support, root, null and 15 others) can't be claimed.

Data bytes	Field	Value
0–3	marker	"INSC"
4	version	1
5	kind	3 claim, 4 transfer, 20 check-in, 21 commitment
6	length	tag length in bytes
7…	tag	UTF-8

- **Claim checks** ask arcades that keep the chain (claims.rs) when this one is still catching up, so a taken tag isn't claimed by mistake.
- **Expiry** (ARCADE only): activity is any record from the owner's address; a check-in pays at least 0.1 ARCADE; a tag is released idle + grace after its last activity (12 months then 30 days, defaults). For the next day (reveal_blocks) a claim needs a commitment sha256("arcade tag commit" || tag || address || salt) from an earlier block, and the earliest commitment wins. Sends warn for 90 days after a released tag changes hands.
- **The snapshot** carried 322 Pepecoin testnet tags into ARCADE testnet's block 0 under the same keys.

Tags per chain [PLANNED]

Approved 2026-11-18, to be built after the borrowed-tags work:

- A tag's full name is its name plus the chain it was claimed on, written with the chain's own key: **@alice.arctest, @alice.peptest, @alice.doge, @alice.dgb**. Tags only use a–z, 0–9 and _, so the dot can't be part of a name. The app shows the ticker: @alice · ARCADEtest.
- Any tag free on a chain can be claimed there, whoever holds the same name elsewhere. This replaces today's single namespace (earliest claim wins).

- Existing tags all stay valid on the chain they were claimed on; where two people held @bob on two chains, each now owns it on their own chain.
- Claimable on ARCADE and every chain arcades index for posts and messages (pep, doge, dgb today).
- **Paying a tag in any coin:** @alice.doge can be paid in Litecoin; the coins go to the Litecoin address @alice.doge has *proven* (wallet proofs, op 7). No proven address for that coin: the send is refused, never guessed.
- A plain @alice works only when exactly one chain has it; otherwise a picker shows each with its chain and profile. A post shows its author's tag on the chain it was written on.

8.3 A:// ops

A:// is the social layer: small records every arcade's ledger reads alike (arcade-ledger aproto.rs). The data is:

```
"A://" || (1 << 4 | op) || body          (the second byte is 0x10 + op: version 1 in the high nibble)
```

A target (ops 8–10) is **chain** (u8 length || ASCII chain key, 1 to 12 lowercase letters and digits) || **txid** (32 bytes in the order the hex txid is written). It can be any transaction on any chain; the ledger doesn't look it up. A record that breaks a rule stays on chain but counts for nothing on every arcade.

Op	Name	Body layout	Limits and counting	Status
1	profile	JSON object: n name, b about, u link, p picture (blob id, 64 hex), a addresses (chain → address)	≤1,000 bytes; n≤64, b≤400, u≤200, a≤12; newest wins	**[BUILT]**
2	follow	an @tag claimed on this chain, or a 40-hex key hash (kept as #hex)	counted on chain, so every arcade has the same followers	**[BUILT]**
3	unfollow	same as follow		**[BUILT]**
4	contact	id (8 bytes) || sealed box only the author can open	≤1,000 bytes	**[BUILT]**
5	contact delete	id (8)		**[BUILT]**
6	post	UTF-8 text	≤1,000 bytes; for followers' feeds	**[BUILT]**
7	wallets	set (u64 BE) || entries of coin (u8 len || key) || address (u8 len || text) || 65-byte recoverable signature	≤7 entries; every entry must check; a higher set replaces, same set adds, lower changes nothing; from arctest block 2,000	**[BUILT]**
8	react	target || kind (u8): 1 like, 2 unlike, 3 dislike, 4 undislike	nothing after the kind; author's newest wins	**[BUILT]** from 2,120
9	reply	target || UTF-8 text	1 to 500 bytes; kept oldest first	**[BUILT]** from 2,120
10	tip	target || memo (UTF-8, may be empty)	memo ≤100; every output of the same tx to another address counts as tipped to it; paying no one: refused	**[BUILT]** from 2,120

Wallet proofs (op 7)

Each entry's signature is Core's signmessage, made with that address's key, with the magic "ARCADE Signed Message:\n" (one for every coin, so a proof can't be replayed as a coin's own message) over: `ARCADE wallet proof: <owner> holds <coin> <address>`. A profile card shows each coin's address as verifiably the owner's. This is what makes paying a tag in another coin safe.

Reactions, replies and tips (ops 8–10)

- **React:** each author has one state per target (none, liked, disliked). Like or dislike replaces what was there; unlike clears a like only; undislike clears a dislike only.
- **Tip:** counted per target: total, count, and per recipient. The target's **author** is the key that signed its first input, as an ARCADE address (one key is one person on every chain). When it can't be found, author is null and tips still count per recipient.
- **Reads:** a.reactions {targets (≤100), me?} → per target {likes, dislikes, replies, tips {total, count, to_author, to[]}, author, mine}; a.replies {chain, txid, offset, limit ≤200}. Public underneath as /r/explorer/a_reactions and /r/explorer/a_replies, forwarded to arcades that keep ARCADE.
- **Writes:** a.react, a.reply, a.tip {amount, to?, memo?}; to defaults to the author and is required when there is none.
- **For apps:** window.arcade.social (/r/social.js): reads open to any app; writes need the app's Feed switch, at most one every 0.4 s; every tip is confirmed on a card the app can't see or press.

8.4 Repositories, tokens and the market

- **Repositories [BUILT]**: REPO records (push, maintainer, fork, open, comment/review, state) under the owner's @tag (@tag/name). Texts are blobs. From rules version 4 a push (op 7: name || manifest || base || flags, bit 0 force) names the manifest it was built on and is refused unless that base is current or the push is forced. An importer brings a GitHub repository's issues, pull requests and reviews.
- **Tokens and the market [PLANNED]** move from Pepecoin testnet to ARCADE in phase 3, priced in ARCADE; the snapshot already carried 121 tokens and 462 balances.

9. Borrowing and the mesh

ARCADE has no separate peer-to-peer network: blocks, transactions and storage items travel over the arcade mesh, the encrypted links arcades already keep.

9.1 ARCADE frames (kind "ac")

Every frame names block 0's id; an arcade answers only for the chain it runs.

Op	Fields	Meaning
tip	id, h, w	my best block, height, total difficulty; sent when it changes
get	id: n, loc	send me your branch after the newest of these block ids that is on it (a thinning locator, at most 32)
a	id: n, seq, end, more + bytes	up to 50 blocks or 4 MB, oldest first, each length-prefixed, in frames of ≤60,000 bytes; the next ask continues from the last block an answer brought
inv	+ bytes	up to 1,000 ids of transactions that came to wait here
gtx	id: n + bytes	send me these (≤100 ids)
txs	id: n, seq, end, left + bytes	the transactions, up to 4 MB; left counts ids that didn't fit
it	item	a waiting storage item (catalog, attestation, registration, audit...)
cand, tp	see 3.5	time keeper candidates and proofs
bl, bq	id, seq, end + bytes	blob pieces (waiting blob; asked pieces)

Asks from one arcade are answered at most twice a second. A peer that sends 5 rule-breaking blocks in an hour has its ARCADE frames ignored for an hour.

9.2 Waiting transactions

The pool's rules are each arcade's choice, not chain rules: a transaction waits if it would be valid in the next block, counting waiting transactions' coins, and pays at least 0.001 ARCADE/kB. The first spend stays (no replace-by-fee). A waiting transaction may spend another's coins up to 25 in a row (MAX_ROW). Room: 5 MB; a full pool takes a newcomer only if it pays more per byte than the cheapest, which leave with their descendants. Scripts are checked once on arrival.

> Chains of unconfirmed spends on the *coin* chains (Pepecoin, Dogecoin...), as used by inscription commit/reveal runs, are refused past the first child today; a fix allowing Core's 25 ancestors / 101 kB is **[IN PROGRESS]**. ARCADE's own pool already allows 25.

9.3 Borrowing ARCADE

[BUILT] An arcade that hasn't installed ARCADE borrows it from arcades that have it, exactly as it borrows a coin's node. Since v0.1.125 new arcades borrow ARCADE until the operator installs it, and an installed copy that is 30+ blocks behind borrows until it catches up (within 3 blocks).

What a borrower checks itself (arcade-arc light.rs, arcaded arcborrow.rs)

- Every header from block 0, by the chain's rules: the writer's signature, the VRF draw, the proof of time with exactly the steps its claimed weight gives, the epoch's difficulty, the draw seed's proof, the time limits. It keeps the id of every height and what the rules need of the newest 400 blocks (KEEP), so a restart carries on.

- **Weights:** a borrower doesn't keep the registry, so it takes each header's claimed weight on the lender's word (UNKNOWN_WEIGHT) while still checking the draw, the proof of time and the difficulty. (Until v0.1.127 genesis writers were checked against their settings weight, which broke at block 1,001 once rules v4 changed their real weight.) A committed per-epoch weight table in the header is proposed **[OPEN]**.
- **Branch:** decided as an installed copy decides it; a lender's other branch is read from up to 100 blocks back and taken only if better and every header passes.
- **Blocks:** taken only if the id is the one the headers give for its height and the body matches the signed roots. Older coins come from a lender's coin scan (arcade_coinscan, ≤100 addresses), each checked against its block.

What it takes on a lender's word

- That a block's transactions spend coins that exist and its reward is right (a bad block must still be written by a weighted writer, and every full copy refuses it).
- That a coin scan left nothing out (a lender can hide a coin, not invent one).
- That there's no better branch than its lenders show.

Lender calls

Call	Answer
getblockcount, getblockhash, getblock, getrawmempool, sendrawtransaction	as a borrowed coin's node
arcade_headers [from, count]	up to 500 headers of its branch, in the chain's own form
arcade_coinscan	unspent coins of up to 100 addresses, from its address index

9.4 Borrowed = installed

[PARTLY BUILT] Decided 2026-11-18: every command should work the same whether a chain is installed or only borrowed; what the local copy can't answer goes to an arcade that keeps it. A full audit of every command found seven batches of work:

Batch	What	Status
1	own sends on borrowed chains reach the network through lenders; tx.coins/fee/activity, mempool, profile	**[BUILT]** v0.1.130
A://	profiles, follows, contacts, posts, likes, replies, tips, wallet proofs from a borrowing arcade; blob cache and serving	**[BUILT]** v0.1.130
2	other people's @tags when tag tables are borrowed	**[IN PROGRESS]**
3	feed and messages polish	**[PLANNED]**
4	ARCADE ledger reads forwarded to keepers	**[PLANNED]**
5	token, ord and explorer reads	**[PLANNED]**
6	inscriptions, DRC-20 and D:// sends built locally, broadcast through lenders	**[PLANNED]**
7	swaps (lock depth from blocks two hosts agree on)	**[PLANNED]**

Not possible borrowed, by nature: mining, staking, sealing and earning, lending and serving copies, the chain's own indexes.

9.5 Services over the mesh (svc)

[BUILT] A general channel for programs beside arcaded to offer a service to other arcades (arcaded svcmesh.rs). Its first user is the AI model marketplace (arcade-models), which sells local-model inference for ARCADE.

Wire ("sv" frames)	Fields	Meaning
offer	s, o, at, tag, name, pay_to, body, sig	a service's offer (≤16 KiB), signed by the arcade's node key, flooded, kept 30 minutes, re-sent every 10
q	id, seq, end + piece	a request: zlib JSON {service, body, grant} in 48 KiB pieces, over the link
a	id, seq, end, ev?, err? + piece	an answer piece; the last carries end or err

A **grant** is a Core-style signmessage by the asker's ARCADE address of `arcade-svc grant <service> <arcade> <address> <expires>`, valid up to a year: it proves who pays. Payment is svc.pay (ARCADE with a memo of up to 80 bytes), checked with svc.paid. An ask reaches an arcade by public name, a relay or a UDP introduction within 25 s, and may stream for up to 20 minutes. Websites may ask in their own name only for asks a service lists as free reads ("reads": [...] in its offer), with size and rate caps.

9.6 Relays

[BUILT] Arcades behind NAT reach each other through relays: a public arcade that carries links and media (TURN) for a price per GB set by its operator, with a free allowance; payment is in ARCADE with a memo, credited once in a block. The project's East Coast VPS relays at 0.01 ARCADE/GB.

10. Code on chain and releases

[BUILT] The arcade's own source, with its whole git history, its vendored dependencies (so it rebuilds with no internet) and its prebuilt programs, are kept by every arcade and distributed over the mesh. Updates come from the network, not from a code host (arcaded arccode.rs, updater.rs; scripts/code-publish.py).

10.1 The release record

Field	Meaning
name	@arcade/arcade-v1
version	for example v0.1.130
commit	git commit id the release is built from
time	seconds since 1970
files	map of name → {sha256, size}: arcade-v1.bundle (git bundle of the history), vendor.tar.zst (cargo vendor --versioned-dirs into cargo-deps/, fixed order, times and owners), program-linux-x86_64, program-linux-aarch64 (release archives), BUILD.txt (how to rebuild offline)

The record is canonical JSON; its **id** is the SHA-256 of the record. The **signature** is ed25519 over "ARCADE code release\0" || id by a trusted release key (RELEASE_KEYS). Vendored crates check against Cargo.lock's checksums (in the commit); programs rest on the signature (builds aren't reproducible yet **[OPEN]**).

10.2 Keeping and fetching

- Every arcade keeps, under code/files/<sha256>, every release's bundle and dependencies and the programs of the newest 5 releases of each name (KEEP_PROGRAMS). Each file is hashed before it is kept.
- Mesh frames ("co"): r a release record (once per link), h the files held, g {id, f, at} a 48 kB piece of a file asked over an existing link, answered a; 16 pieces in flight (58.6 MB in about 2 s between two arcades on one machine). Fetches resume where they stopped.
- Sizes at v0.1.66: bundle 8.8 MB (577 commits), dependencies 58.6 MB (537 crates, 763 MB unpacked), ARM Linux programs 13.8 MB.

10.3 Publishing and updating

Step	What
1	Tag the release; CI builds the Linux programs and checksums.
2	code-publish.py builds the bundle and vendored dependencies, **scans the whole history** for anything that must never be published (private keys, WIF-shaped strings, extended private keys, home paths, private network addresses) and stops on any new hit, then signs with the release key.
3	code.add {record, sig, files} on one arcade; the release spreads over the mesh.
4	Each arcade's updater checks every 10 minutes plus a random 0–30 minutes, takes the newest held @arcade/arcade-v1 release for its platform, checks it, installs it (keeping .prev), and restarts cleanly, waiting for its running proof of time or the draw seed's proof first.

Release discipline: no network publish in the last 20 blocks of an epoch (heights x80–x99), when every writer is running the draw seed's long proof of time; two early releases that restarted arcades there threw that proof away twice and stalled the chain. Mainnet's block 0 will name the first release's id.

11. Economics

11.1 Supply

Issuance is 50 ARCADE every 150 s forever: 864,000 a 30-day month, about 10.5 million a year. There is no premine and no cap. Holding ARCADE earns nothing; keeping data does. The writer's fixed 1 ARCADE makes withheld or orphaned blocks cost their writers something, without creating a fee prize to fight over.

11.2 What hosting costs, and the penny target

The project's target is that, at 1 ARCADE = $0.01, every VPS keeping data is paid twice its cost each month. With the first figures (two VPSs, about $7 and $8 a month), that is about 3,000 ARCADE a month to the two, or about 47 ARCADE per GB per copy per month for storage (section 7.3).

11.3 The tension: issuance against fees

At a penny, one month's issuance is worth $8,640, shared among the storing arcades. With the 15 or so owners on the testnet today, each would get far more than twice any VPS from new coins alone, so storage fees hardly matter. For a penny to be a sustainable price, either:

- **issuance falls**, for example a reward set by need instead of a fixed 50 (the oracle's phase 5), within a hard supply schedule; or
- **the price follows demand**: ARCADE trades at whatever its use supports, and the penny is a reference for setting fees, not a peg; or
- **most of storers' pay comes from users**, through copy-floor pricing (section 7.3), with issuance a shrinking share as the network grows.

This is **[OPEN]**.

11.4 The hosting oracle

[PLANNED] A plan to value ARCADE by what it costs to run an arcade. A **hosting unit** (HU) is one month of a reference machine in three tiers:

Tier	Reference machine	Role
HU-S	2 vCPU, 4 GB RAM, 80 GB disk, 2 TB traffic	relay, light chains, the ARCADE chain
HU-L	8 vCPU, 32 GB RAM, 2 TB NVMe, 10 TB traffic	every chain in full, indexes, storage weight
HU-G	HU-L plus one 24 GB GPU	compute jobs

Two indexes feed it: **ARCADE in coins**, the volume-weighted median of settled ARCADE↔coin atomic swaps reported on chain with Merkle paths (median, distinct keys, a cap per key, ×1.25 clamp per epoch); and **hosting in coins**, quotes and on-chain payments to hosts by arcades with draw weight (median of 90 days, outliers dropped, payments counted double). ARCADE per HU = hosting index ÷ swap index. Phases: 1 dashboard; 2 operators price in HU and may pay invoices from earnings; 3 indexes on chain, nothing paid by them; 4 fees set in HU; 5 rewards set by need (±10% per epoch, hard supply schedule); 6 GPU compute priced in HU-G with compute checks.

11.5 Who earns what

Role	Earns	From
Storage owner	its share of 49 ARCADE a block + fees, by pay weight	the pot, at its turn, after passed audits
Block writer	1 ARCADE per block written	the coinbase
Genesis earner	fixed bytes' share, no audit	the pot
Relay operator	its price per GB	users, by memo payment
Model seller	its price per million tokens	buyers' prepaid credit (svc)
Time keeper, serving checker	nothing	unpaid **[OPEN]**

12. Security and risks

Two independent reviews attacked the design (eight groups of flaws in the first, nine points in the second); every fix is in the built rules. The table lists what remains.

Risk	Guard	Still open
Recomputing sealed pieces instead of storing them	whole-copy audits every 2 days; each missing piece costs a full square root per audit; each audit releases only its own slice	the gap is about 30× on GPUs and less on special hardware
Many names on one disk	each key's copy sealed and audited separately	none known
Seal, register, delete	pay only after a later audit; draw weight only after a passed audit	none known
Changing a commit after seeing its challenge	proof of time from the previous block's hash; signed blocks; challenges wait 10 blocks	a writer of two consecutive blocks can try a few challenges, one proof of time each
Rewriting history	proof of time; most difficulty; lower score on ties; 100-block reorg limit; payments final after 60	safe limit about 40% of storage; a faster time calculator multiplies its owner's storage
Grinding draws	draw fixed per epoch; draw seed behind a proof of time	each try costs a fast core about 40 minutes
Small network outweighed	sealing before genesis; release checkpoints	true of every new chain
First arcade captures the chain	genesis list closed before the beacon; every genesis replica audited; signed receipts	the list is assembled by one arcade; receipts and public checking are the guard
Keeping data without serving it	secret serving checks	verdicts are reports; checkers unpaid; the weakest guard
Made-up catalog entries	one root per range fixed by the source; confirmed by weight; reveal demands; fraud proofs	index journals need fraud proofs of their own; witness, merged-mining and MWEB data
Writers leaving out rivals' answers	20-block windows; answers first; re-audits	needs about 80% of storage to matter
Flooding the storage section	every item tied to a pick, a commit, a VRF draw, draw weight or a work stamp; commits ranked by work; audit period stretches by rule	none known
Testnet without the 200-block draw wait	mainnet keeps it	on the testnet a replica registered on a secret branch counts there at once, on purpose
Borrowers trusting lenders' weights	headers, draws and proofs of time still checked	a committed weight table per epoch **[OPEN]**
Fog of war: publisher vanishes before anyone views	grace keepers; audited copy floors	the floor and window numbers **[OPEN]**

12.1 Stated plainly

- No fast-unsealing scheme is cheat-proof in the mathematical sense; ARCADE makes cheating cost more than honesty and pays nothing until audits pass.
- A network split of more than 4 hours doesn't heal by itself; the smaller half's arcades must be told which branch to follow.
- Neither review looked at attacks on the mesh itself (cutting an arcade off from honest neighbors) or at groups of serving checkers working together.
- Byte-level rules still to be finalized before mainnet: the exact segment bytes for witness, merged-mining and MWEB data per chain.

12.2 Restart, reorg and failure behavior, component by component

What each part does when an arcade restarts, when the chain changes branch, and when something fails. This is what the code does today.

Component	On restart	On a reorg (branch change)	When it fails
Chain store	the best branch is read back from the chain store and **every check is run again**, so a damaged store can leave blocks out, never add one the rules refuse	blocks taken back are undone in order; side blocks are kept in memory only and come again from peers	a block failing a body check is remembered as bad with everything built on it; the arcade returns exactly to where it was
Writer	parents already written on are read from arc/<net>/wrote, so it never signs a second block on one parent	keeps its proof running only if its score still beats the rival's	a block ahead of the clock is offered again as it is; never rewritten
Proof of time	a block's own proof starts over; the draw seed's proof resumes from saved pieces (rules v4)	proofs on the old tip are dropped	updater waits for the running proof before restarting; no release is published at heights x80–x99
Pool	not saved: wallets offer their transactions again	transactions of blocks taken back are checked again in full and wait again	a transaction refused is answered "refused a moment ago" until the next block
Catalog	rebuilt from the chain's blocks	items are undone block by block	a wrong root is removed by a fraud proof or an unanswered reveal; its span reopens
Sealing	partial seal directories are deleted and restarted; finished replicas stay	segments are 100 source blocks deep before listing, so source reorgs don't reach them	sealing waits when free disk is under 2 GB
Registration	a batch waits until replicas stop arriving for a minute (or 30 minutes)	a commit or proof taken back can be posted again	a commit whose window passed can be made again
Audits	answered on a 5-second loop while the sealed copy is on disk	a commit or opening taken back is posted again inside its window	a miss forfeits the period's pay; a re-audit is requested at once; 3 misses remove the replica
Pay	accrued pay lives in the chain's state, not on the arcade	settlements and credits are undone with their blocks	under 0.01 ARCADE waits; held pay goes back to the pot after 3 holds
Borrowed copy	goes on from the newest 400 blocks' rule state; doesn't start at block 0 again	a lender's other branch is taken only if better and every header passes	a lender whose branch breaks a rule is set aside for a while; another lender is asked
Blob store	the plain store and cache stay on disk; waiting blobs in memory are lost and come again with their transactions	blobs are written once on any branch and kept	a fetched blob that doesn't match its id is dropped
Code store	files stay; downloads resume where they stopped	releases aren't on the block chain's branches	a file whose hash doesn't match isn't kept

13. Status and open questions

13.1 Where things stand (2026-11-18)

Area	Status
Public testnet arctest	**[BUILT]** running since 2026-11-15 (third start), height 2,091, rules version 4 from block 1,000, real proof of time, registration, audits, serving checks
Blobs, pruning, cache, files, /content/, Git on chain	**[BUILT]**
Ledger on ARCADE: tags, profiles, follows, posts, wallet proofs (from 2,000), reacts/replies/tips (from 2,120), repos	**[BUILT]**
Tag expiry	**[PARTLY BUILT]** (built; the commit-rule check is being rerun)
Code on chain, updates from the network	**[BUILT]** (latest release v0.1.130)
Borrowing; borrowed = installed	**[PARTLY BUILT]** (batches 1 and A:// done; 2 in progress; 3–7 planned)
Fog-of-war storage	**[IN PROGRESS]**
Tags per chain	**[PLANNED]**
Storage analytics app (segments, holders, weights, payments)	**[IN PROGRESS]**
Hosting oracle	**[PLANNED]**
Mainnet genesis list, settings, checkpoints, ledger migration	**[BUILT]** rehearsed on closed networks; dates fixed
Testnet rehearsal (mid-December), genesis window (Dec 21–31), mainnet launch (2027-01-01)	**[PLANNED]**

13.2 Open questions

- Copy floors per class, the idle-fade days, the grace window, and how long Permanent prepays (section 7).
- Issuance against the penny target (section 11.3).
- Pay for time keepers and serving checkers.
- A committed per-epoch weight table so borrowers check writers' weights.
- Fraud proofs for every index journal; segment bytes for witness, merged-mining and MWEB data.
- Reproducible builds, so programs rest on more than the release signature.
- Adding rules version 3 (blobs) to mainnet's schedule before block 0.

Glossary

Term	Meaning
Arcade	a running copy of Arcade V1.0 (arcaded), with its wallet, mesh links and the chains it keeps or borrows
Attestation	a weighted arcade's vote that a catalog entry's root is right
Audit period	1,152 blocks (2 days); each owner is audited once a period
Blob	content named by a ~110-byte record in a transaction, bytes in the block's blob section
Borrow	use a chain through arcades that keep it, checking what one can
Catalog	the on-chain list of segments every arcade agrees exist
Copy floor	the fewest audited copies a class of blob may have (fog of war)
Draw weight	the bytes that count for writing blocks
Epoch	100 blocks; the pot is shared and the draw seed changes each epoch
Family	a kind of segment data: blocks, an index journal, coins, blobs
Genesis earner	a holder of fixed bytes in block 0
Lender	an arcade that serves a chain to borrowers
Owner	an arcade's mesh key, holding all its replicas
Pay weight	the bytes the pot is shared by: B/n per registered replica
Pot	49 ARCADE a block plus fees and forfeits, shared every epoch
Proof of time (VDF)	a computation that takes a set number of sequential steps and leaves a short, quickly checked proof
Replica	an owner's sealed copy of one segment
Rules version	a numbered set of chain rules, switched on at a height
Sealing	encoding data with the owner's key so a copy is slow to make and useless to others
Segment	16 MiB or more of one family of one chain over whole 10,000-height ranges
Serving check	a secret checker fetching data from an owner and reporting whether it was served
Time keeper	an arcade running one proof of time per height for every candidate
Turn	H(earner) mod 1,152: the block in each period that pays an earner

Appendix: parameters

From `crates/arcade-arc/src/params.rs` unless another file is named. Values are mainnet and testnet alike unless noted.

Constant	Value	Meaning	File
COIN	100,000,000	units per ARCADE	params.rs
BLOCK_REWARD	50 ARCADE	minted each block	params.rs
WRITER_REWARD	1 ARCADE	writer's share (rules v2+)	params.rs
BLOCK_SECONDS	150	target block time	params.rs
EPOCH_BLOCKS	100	pot sharing, draw seed, difficulty	params.rs
DRAW_SEED_BLOCKS	80	VRF outputs mixed into the next seed	params.rs
DRAW_SEED_TIME_BLOCKS	15	length of the seed's proof of time (blocks 66–80)	params.rs
RETARGET_CLAMP	4	difficulty moves at most 4×	params.rs
MAX_FUTURE_SECONDS	120	how far ahead a block's time may be	params.rs
AUDIT_PERIOD_BLOCKS	1,152	audit period, turns	params.rs
CHALLENGE_DELAY	10	blocks before a challenge's seed	params.rs
ANSWER_WINDOW	20	blocks to answer a challenge	params.rs
AUDIT_SAMPLE	4	replicas audited per owner per period	params.rs
AUDIT_OPENINGS	4	leaves opened per audit	params.rs
REGISTRATION_LEAVES	16	leaves proved at registration	params.rs
MAX_MISSED_AUDITS	3	misses in a row that remove a replica	params.rs
DRAW_WEIGHT_WAIT	200 (0 on testnet from v4)	blocks after first passed audit	params.rs
REORG_LIMIT	100	deepest branch change	params.rs
FINAL_AFTER	60	blocks for payment finality	params.rs
COINBASE_MATURITY	100	blocks before a reward can be spent	params.rs
MAX_TX_BYTES	1,000,000	coin transactions per block	params.rs
MAX_STORAGE_BYTES	4,000,000	storage section per block	params.rs
MAX_REGISTRATION_PROOFS	8	per block	params.rs
MAX_VERDICTS	1,000	per block	params.rs
MAX_REGISTRATION_COMMITS	200	per block	params.rs
MAX_CATALOG_ITEMS	64	per block	params.rs
MAX_REVEAL_BYTES	65,536	per reveal demand	params.rs
REVEAL_WINDOW	100	blocks to answer a reveal or challenge	params.rs
SEGMENT_BYTES	16 MiB	minimum segment	params.rs
RANGE	10,000	heights per range	params.rs
MIN_FEE_PER_KB	0.001 ARCADE	transaction fee floor	params.rs
DUST	0.001 ARCADE	smallest output	params.rs
MIN_PAYOUT	0.01 ARCADE	smallest payout	params.rs
SERVE_CHECKS	6	serving checks per owner per period	params.rs
VERDICT_WINDOW	20	blocks to post a verdict	params.rs
HOLD_VERDICTS	5	verdicts needed to hold pay	params.rs
HOLD_FORFEIT	3	holds in a row that forfeit	params.rs
SERVE_PIECE_BYTES	8 MiB	fetched per serving check	params.rs

Constant	Value	Meaning	File
SERVE_SECONDS	60	time to serve	params.rs
BLOB_WINDOW	30 days	blocks carry blob bytes this long	params.rs
RULES_KNOWN	4	newest rules version	params.rs
MAX_POT_RUNS	8	proof-of-time runs per block	params.rs
PROTOCOL / MIN_PROTOCOL	1 / 1	ARCADE mesh protocol	params.rs
MAX_BLOB_BYTES	4,000,000	per blob and per block	blob.rs
MAX_TYPE	64	blob type length	blob.rs
BLOB_FEE_PER_KB	0.01 ARCADE	blob byte fee	blob.rs
LEAF / DATA_LEAF	127	data leaf bytes	blob.rs, sloth.rs
SEALED_LEAF	128	sealed leaf bytes	sloth.rs
P_SUB	105	Sloth prime p = 2^1024 − 105	sloth.rs
CHALLENGE_WINDOW	100	blocks before replicas of a lightly confirmed entry register	catalog.rs
FRAUD_DEPTH	100	source blocks deep before fraud proofs judge them	registry.rs
STAMP_BITS	20	work stamp on a commit	registry.rs
ANCHOR_AGE	100	how old a commit's anchor may be	registry.rs
MAX_POOL_BYTES	5,000,000	waiting transactions	pool.rs
MAX_ROW	25	unconfirmed ancestors in ARCADE's pool	pool.rs
KEEP	400	recent blocks a borrower keeps rule state for	light.rs
MAX_FILE_BYTES	1 GiB	file manifest limit	arcaded arcblob.rs
MAX_WAITING	64 MiB	waiting blobs in memory	arcaded arcblob.rs
viewer cache default	2 GB	Options → Blob cache	arcaded arcblob.rs
blob_temporary	90 days	Temporary lifetime, boost length	arcaded arcsvc.rs
MAX_REPLY / MAX_TIP_MEMO	500 / 100	A:// reply text, tip memo	arcade-ledger aproto.rs
MAX_PROFILE / MAX_POST	1,000 / 1,000	A:// profile, post	aproto.rs
MAX_WALLETS	7	proofs per wallets record	aproto.rs
tag length	2 to 24	a–z, 0–9, _	arcade-ledger tags.rs
KEEP_PROGRAMS	5	newest releases' programs kept	arcaded arccode.rs
Testnet rules schedule	(3, 0), (4, 1,000)	versions and heights	params.rs TESTNET
Mainnet rules schedule	(2, 0)	see the note in 3.8	params.rs MAINNET
Mainnet genesis time	1,798,761,600	2027-01-01 00:00:00 UTC	params.rs MAINNET
Address versions (main / test)	P2PKH 0x17 / 0x53; P2SH 0x1c / 0x57; WIF 0x97 / 0xd3		params.rs
Store chain numbers	1000 arc / 1001 arctest		params.rs

Appendix B: worked examples

B.1 What a cheater loses

An owner registers replicas it doesn't fully keep. With 4 replicas audited per period, a missing share f passes a period with chance $(1–f)^4$, and a failed period forfeits all of that period's pay:

Share missing f	Pass chance (1–f)^4	Expected pay kept	Extra disk saved
0%	100%	100%	0%
5%	81.5%	81.5%	5%
10%	65.6%	65.6%	10%
25%	31.6%	31.6%	25%
50%	6.25%	6.25%	50%

Saving 10% of disk costs about a third of all pay; disk is cheap enough that this never pays. Recomputing missing leaves instead costs CPU at every audit (section 5.6).

B.2 Splitting the pot

An epoch's pot is 4,900 ARCADE (49 × 100) plus fees, say 4,910. Three owners:

Owner	Holds	Pay weight (B/n)	Share of 4,910
A	S1 (16 MiB, alone) and S2 (32 MiB, shared with B)	16 + 16 = 32 MiB	32/80 = 1,964 ARCADE
B	S2 (shared with A) and S3 (32 MiB, shared with C)	16 + 16 = 32 MiB	32/80 = 1,964 ARCADE
C	S3 (shared with B)	16 MiB	16/80 = 982 ARCADE
Total		80 MiB	4,910 ARCADE

Shares are rounded down to units; leftover units go one each to the largest remainders. Each share accrues and is paid at the owner's turn if its audits passed. If a fourth owner D registered S1 too, A's weight from S1 would halve to 8 MiB: holding data few others hold pays more.

B.3 A difficulty retarget

Epoch e ran its 100 blocks in 21,000 s (210 s a block) because two arcades were offline. D' = D × 15,000 / 21,000 = 0.714 D. If the next epoch ran in 3,000 s (30 s a block), actual would be clamped at a quarter of the target (3,750 s), so D' = 4 D, not 5 D.

B.4 The lifetime of a blob, today and under fog of war

Day	Today (every full arcade)	Fog of war (planned)
0	publisher posts a 2 MB Temporary clip, pays 20 ARCADE; every full arcade stores it	publisher posts it, pays by the floor (3 copies × 90 days); only the publisher and 3 grace keepers store it
1	a viewer on a borrowing arcade fetches it (cache)	viewers fetch it, keep it, serve it; it spreads by demand
7	—	grace window ends; the floor of 3 audited copies holds
30	—	viewers who haven't watched it again for 30 days drop it (above the floor)
90	deleted on every arcade	Temporary period ends; keepers may drop it (unless boosted)

B.5 Paying a tag in another coin (planned)

Bob types "@alice.doge" and picks Litecoin, 1 LTC. His arcade resolves @alice.doge on Dogecoin's tag index (or asks an arcade that keeps it), reads @alice.doge's A:// wallets record (op 7) on ARCADE, finds an entry for ltc whose signature checks for that tag's key, and pays that Litecoin address. If no ltc entry exists, the send is refused with "@alice.doge hasn't proven a Litecoin address". If Bob types only "@alice" and both @alice.doge and @alice.ltc exist, a picker shows both, each with its chain and profile.

B.6 A release reaching an arcade

Time	What
T	the release's code.add on one arcade at a block whose height mod 100 is under 80
T + seconds	its record ("co" r) floods the mesh; arcades fetch the files in 48 kB pieces, 16 at a time
T + 10 to 40 min	each updater's next check finds it, checks hashes and signature, installs it
at the restart	waits for the arcade's running proof of time or draw seed proof; then a clean restart into the new version