

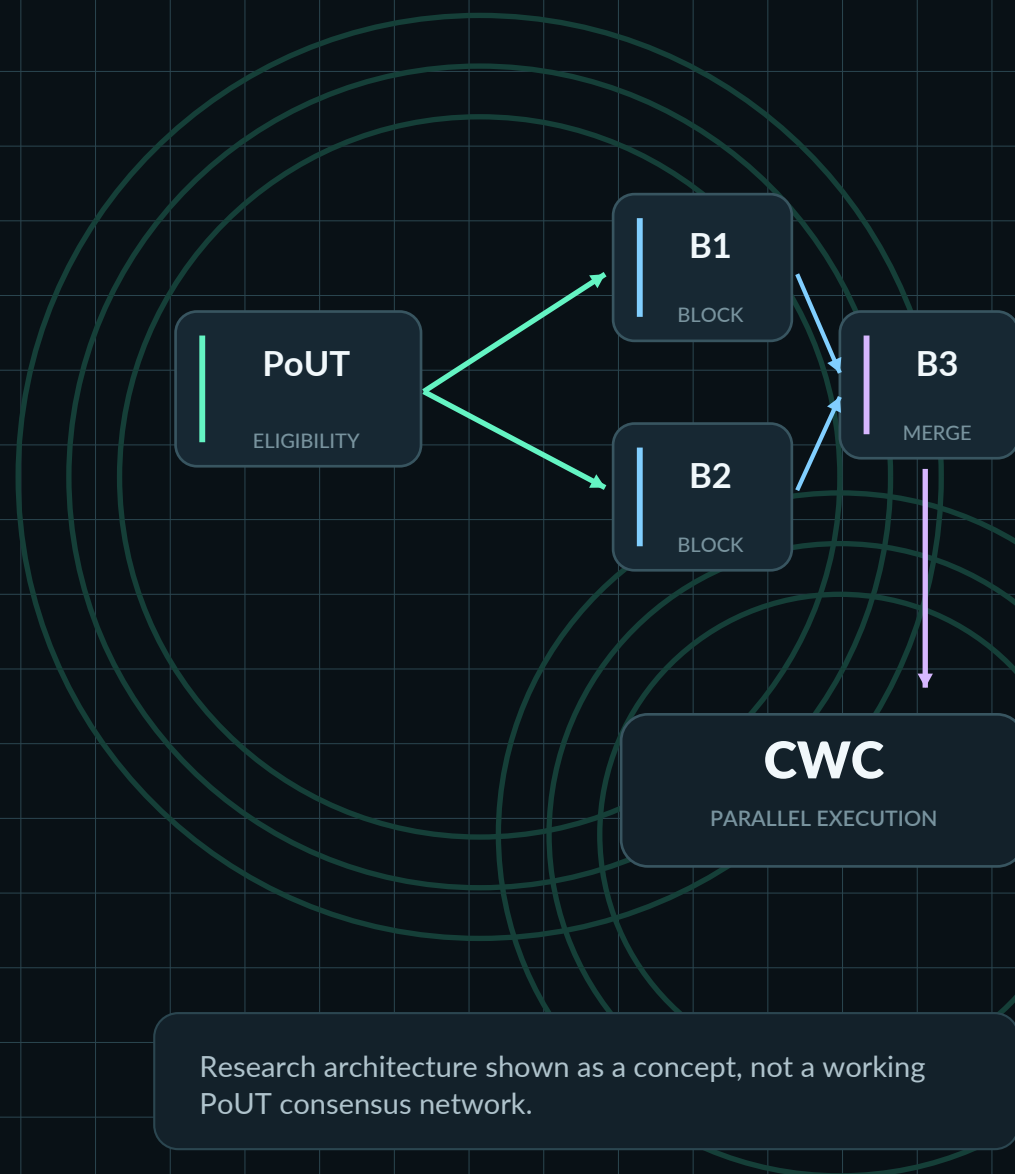
INSIDE THE ENGINEERING.

How the Rust network works today — and how PoUT, GHOSTDAG and CWC could fit together in a future native chain.

REAL RUST EXCERPTS

PLANNED CONSENSUS

A compact, non-reproducible look inside a private codebase. No source repository, credentials or identifying metadata disclosed.



01 / THE FOUNDATION

A service network first.

The current repository is a Rust prototype for encrypted file delivery with bounded simulated accounting. The native blockchain is a separate, unfinished research goal.

CURRENTLY IMPLEMENTED / CONTROLLED TESTS



Files are encrypted locally, split into verifiable ciphertext chunks and placed with storage providers. Recipients check integrity and reconstruct/decrypt on their own device.

WHAT THIS DOES NOT CLAIM

No production decentralized consensus, no native token payments, no fully decentralized coordinator, and no proof that signed traffic is scarce useful work.

Chunk metadata pout-core

GENUINE RUST EXCERPT

```
01 pub struct ChunkInfo {
02     pub index: u32,
03     pub id: String,
04     pub plaintext_bytes: u64,
05     pub ciphertext_bytes: u64,
06     pub nonce: String,
07 }
```

● Crypto & integrity

Signed manifests, authenticated encryption, chunk and final-file verification.

● Recovery & storage

Replica fallback, retention, quotas and interruption-aware state.

● Mock economics

Funded authorizations, receipts, refunds and atomic settlement.

02 / FROM FILE TO CIPHERTEXT

Privacy and integrity by design.

These are selected lines from the actual Rust workspace. They demonstrate verification boundaries without publishing complete algorithms, key handling or internal architecture.

Manifest authenticity

pout-crypto

REAL SOURCE / MANIFEST CHECK

```
01 pub fn verify_manifest(manifest: &SignedManifest) -> Result<()> {
02     manifest.manifest.validate()?;
03     verify(
04         MANIFEST_DOMAIN,
05         &manifest.manifest,
06         &manifest.manifest.publisher_key,
07         &manifest.signature,
08     )
09 }
```

Chunk integrity before commit

pout-storage

REAL SOURCE / STORAGE VALIDATION

```
01 let expected = self.expected_chunk_bytes(id, now)?;
02 ensure!(
03     expected == upload.expected_bytes && actual_bytes == expected,
04     "invalid chunk length"
05 );
06 ensure!(hash == id, "corrupt chunk hash");
```

HOW A VERIFIED TRANSFER FLOWS



Encrypted chunks can be copied between providers; plaintext and decryption keys are not intended for provider storage or on-chain publication.

03 / RECEIPTS ARE EVIDENCE, NOT MINING

A signature is not proof of useful work.

A provider and customer can cryptographically acknowledge a transfer. That does not independently prove the transfer happened, was useful, or consumed a scarce resource.

Two-party receipt shape

pout-protocol

REAL SOURCE / SIGNED RECEIPT

```
01 pub struct SignedReceipt {  
02     pub claims: ReceiptClaims,  
03     pub provider_signature: String,  
04     pub customer_signature: String,  
05 }
```

THE UNSOLVED ATTACK



Both parties may sign convincing receipts even if no independent valuable service was delivered.

WHAT THE FUTURE POUT CONSENSUS NEEDS

Independent checks

A validator must verify bounded evidence without calling outside services.

Scarcity & abuse costs

Fake traffic, self-payments and unlimited new identities cannot farm weight.

Fork-safe consumption

One service event cannot earn weight repeatedly on competing histories.

Engineering boundary: Verified delivery and funded mock payouts work in a controlled prototype. Consensus-grade useful-service verification does not yet exist.

04 / THREE DIFFERENT JOBS

PoUT -> GHOSTDAG -> CWC

Think of the future chain as three separate questions: who may produce blocks, how concurrent blocks are ordered, and how state changes can be executed efficiently.

01

PoUT

ELIGIBILITY / WEIGHT

Who can take part? Could independently verified useful service replace conventional mining work?

RESEARCH

02

GHOSTDAG

BLOCK ORDERING

If eligible producers create concurrent blocks, how does every node agree on their order?

RESEARCH

03

CWC

IN-BLOCK EXECUTION

Once blocks are ordered, which state operations are independent and safe to process concurrently?

SPECIFICATION

THE IMPORTANT SEPARATION

GHOSTDAG-style ordering does not establish scarce work. CWC-style parallel execution does not establish consensus security. Both remain downstream of a PoUT proof model that can resist collusion and fabricated traffic.

05 / GHOSTDAG-INSPIRED ORDERING

Parallel blocks. One agreed history.

A BlockDAG can represent blocks discovered at similar times. An ordering rule is needed so every honest node interprets those blocks consistently. The exact PoUT adaptation is not decided.

ILLUSTRATIVE BLOCKDAG / NOT LIVE POUT CODE

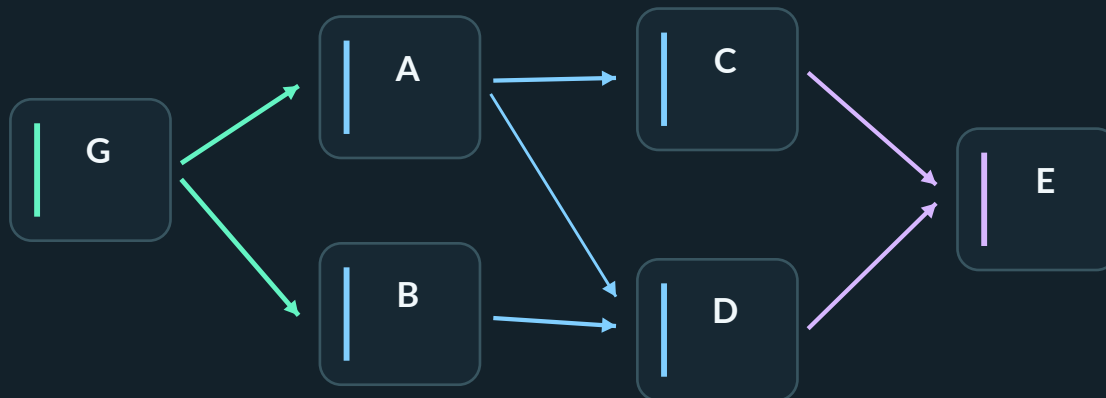


Illustration only. This shows concurrent ancestry and later merging, not a specific GHOSTDAG parameterization, blue/red decision or PoUT block-validity rule.

WHAT MUST BE DECIDED

- **Block eligibility**
Valid, scarce useful-service evidence before an accepted block.
- **Ordering + finality**
Unambiguous history, conflicts, safety, liveness and confirmation rules.
- **Security re-analysis**
PoW assumptions in established GHOSTDAG cannot simply be reused for PoUT.

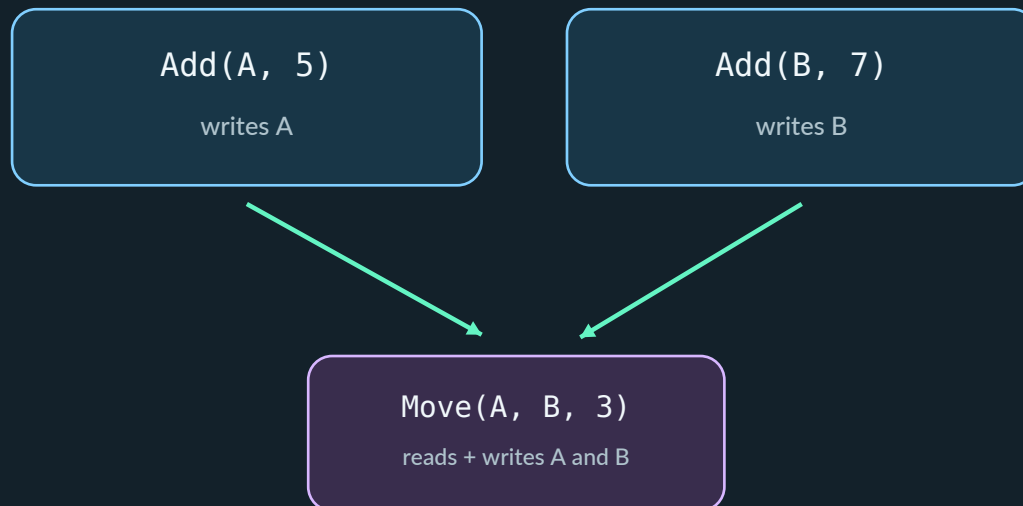
Background concept: Sompolinsky, Wyborski & Zohar, PHANTOM GHOSTDAG (2021). PoUT is investigating an inspired architecture, not asserting it has adopted or secured that algorithm.

06 / WHAT CWC ACTUALLY MEANS

Parallel work, identical state.

CWC is the internal project term in the technical specification (not a separate documented “CVC” module). It concerns independent state operations inside one already ordered block.

ILLUSTRATIVE OPERATIONS / DESIGN MODEL



The first two operations do not conflict. The move depends on both and must wait. Worker completion order must not change the final result.

REQUIRED SAFETY RULE

**SERIAL RESULT
= PARALLEL RESULT**

Both paths must agree on validity, operation outputs, final state and state root. A rejected block must never commit partial state.

This is a specification. There is no implemented native CWC executor or verified speedup yet.

Two meanings of “parallel”: GHOSTDAG-inspired research concerns concurrent blocks. CWC concerns independent operations inside one block. They solve different problems.

07 / INSIDE THE WORKING RUST

Small excerpts. Real invariants.

These compact, verbatim snippets come from separate private-source modules. They are intentionally too limited to reproduce the implementation.

 Exactly-once mock settlement

pout-ledger

```
01 if let Some((json, charged, provider_credit, network_fee)) = cached {
02     let stored: SignedReceipt = serde_json::from_str(&json)?;
03     ensure!(
04         *receipt == stored,
05         "authorization settled by a different receipt"
06     );
07     transaction.commit()?;
08     return Ok(Settlement {
09         charged, provider_credit, network_fee,
10     });
11 }
```

 Provider I/O trace boundaries

pout-node (test only)

```
01 pub(super) enum Boundary {
02     Submit,
03     WorkerEntry,
04     WorkerReturning,
05     AsyncResumed,
06 }
```

WHY THESE EXCERPTS MATTER

 Ledger consistency

A repeated receipt returns its prior settlement instead of paying twice.

 Timing transparency

A write can be traced through submission, worker execution and async continuation.

 Honest evidence

388 passed in final Rust validation, but historical network delays still need proof.

Source labels identify modules only; no private repository URL, user identity, credentials or complete sensitive implementation is included.

08 / HOW MIGRATION COULD FIT

A reserve is not a mining reward.

The Pump.fun community token already exists according to the project owner. A future native PoUT chain could separate an initial holder reserve from later consensus-issued coins, but no terms are approved.

CONCEPTUAL FUTURE SUPPLY PATHS

EXISTING TOKEN

Pump.fun / Solana



MIGRATION RESERVE

genesis escrow - hypothetical



NATIVE HOLDER

future claim only

Separately: future PoUT mining emissions

Checked ratio arithmetic (simulation)

pout-migration-sim

```
01 let product = u128::from(source_units) * u128::from(self.numerator);
02 if product % u128::from(self.denominator) != 0 {
03     return Err(Error::NonintegralEntitlement);
04 }
05 u64::try_from(product / u128::from(self.denominator))
06     .map_err(|_| Error::Overflow)
```

ACCOUNTING IDENTITY

Supply = Genesis + Emissions - Burns

Claims move already-reserved coins. Mining would issue new coins under separately approved rules. Burn, lock and snapshot models still have unresolved safety and legal questions.

09 / EVIDENCE, NOT HYPE

What has been verified?

Current source snapshot: Round 7F, 16 October 2026. Test claims below refer only to the named development sources and platforms.

388

RUST TESTS PASSED

0

FAILURES IN FINAL RUN

4 / 4

CI JOBS SUCCESSFUL

OPEN

LONG-RUN GATE

VERIFIED DEVELOPMENT CAPABILITIES

- Encrypted transfer and chunk verification
- Signed manifests and delivery receipts
- Replica fallback and controlled restarts
- Bounded mock settlement / replay guards
- Focused Linux reliability instrumentation

MAJOR OPEN RESEARCH / RELEASE GATES

- 30-minute sustained reliability acceptance
- Independent useful-service scarcity proof
- Secure PoUT eligibility / GHOSTDAG adaptation
- CWC executor, finality and native settlement
- Reviewed source-token migration mechanism

Updates and discussion: <https://t.me/PoUTnetwork>