HF-A Implementation Plan

Decentralized Fast Transfer Rail for Local Transfer Acceleration

1. Summary

HF-A introduces an optional, decentralized fast transfer rail as a node, RPC, and P2P feature without changing Basechain consensus, block production, or finality.

The Basechain remains the sole settlement and finality layer. HF-A provides faster **local** visibility and local fast confirmation of transfers under normal network conditions, while remaining strictly bounded in CPU, memory, queue depth, and time. Under overload, instability, or attack, the fast rail may degrade, pause, or be disabled without impairing normal Basechain operation.

HF-A is not a second consensus, not a second chain, and not a fast-finality mechanism.

---

2. Goals and Non-Goals

2.1 Goals

HF-A is intended to provide:

- very fast local transfer confirmation under normal operating conditions,
- strict prioritization of liveness and stability,
- deterministic and reproducible semantics for status, conflicts, retention, restarts, and wallet presentation,
- implementation on the existing node architecture without consensus migration.

2.2 Non-Goals

HF-A explicitly does **not** provide:

- a second consensus,
- independent finality,
- on-chain fast settlement,
- committee certificates or slashing,
- fast-on-fast dependency handling,
- support for unconfirmed parents in the fast path.

---

3. System Model

HF-A consists of the following logical components:

- **FastIngress** for RPC and P2P entry,
- **AdmissionPipeline** for prechecks, policy checks, and base transaction validation,
- **ConflictArbiter** as the single serialized conflict point for fast and normal paths,
- **FastStateStore** for pre-lock and active fast state,
- **FastMicroblockEngine** for local ordering containers,
- **PullOnMissEngine** for bounded retrieval of missing intents,
- **ModeController** for `normal`, `degraded`, and `paused`,
- **RetentionStore** for bounded terminal-state visibility.

The Basechain path always has higher priority than the fast path.

---

4. Normative Invariants

The following invariants are mandatory:

1. Basechain and normal transaction processing always have higher priority than the fast rail with respect to CPU, queues, locks, workers, and scheduling.
2. The fast rail is optional and disabled by default.
3. HF-A introduces no new consensus rules and no new global finality claims.
4. `fast_confirmed` is local only, never global and never final.
5. Fast-vs-normal conflict decisions are made exclusively at the shared serialized arbiter.
6. All fast-path data structures are bounded in count, memory, queue depth, and execution cost.
7. Under overload or attack, the fast rail may be paused or disabled without affecting Basechain progress.
8. Fast state is ephemeral and is not persisted across process restart.
9. Cancel is node-local and is never propagated through P2P.
10. Fast-on-fast dependencies are forbidden.
11. Inputs depending on unconfirmed parents are forbidden in the fast path.

---

5. Terms and Semantics

5.1 Key Terms

FastIntent
A signed transfer intent with fast-path admission semantics.

base_tx
The underlying signed base transaction.

base_tx_fingerprint
The canonical hash of the normalized base transaction.

intent_id
The deterministic identifier of a FastIntent.

node_epoch
A node-local context identifier generated once per process start.

fast_confirmed
Local inclusion in a locally accepted fast microblock.

unknown_intent
The intent is not currently known on this node.

5.2 Status Definitions

received
The object has been accepted and successfully parsed at syntax/structure level, but full base transaction validation has not yet completed.

validated
Full base transaction validation and fast-path policy checks have completed successfully, but no input lock has yet been acquired.

locked
Atomic input reservation at the arbiter succeeded.

fast_confirmed
The intent is included in a locally accepted fast microblock. This is a local signal only.

expired
The local TTL elapsed and the lock was released.

dropped
The intent left the active fast path for a non-TTL reason.

rejected
Admission was rejected before active lock acquisition.

cancelled
The node-local fast context was ended. This does not imply reversal of any Basechain transaction.

unknown_intent
An RPC projection meaning the intent is not currently available on this node. It makes no global statement.

---

6. Data Model and Deterministic ID Derivation

6.1 FastIntent Fields

A FastIntent contains:

- `base_tx`
- `intent_nonce: u64`
- `client_created_at_ms: u64`
- `max_fee: u64`

6.2 Deterministic Derivation

1. Normalize `base_tx` into `consensus_core::tx::Transaction`.
2. Compute `base_tx_fingerprint = hashing::tx::hash(base_tx, false)`.
3. Compute:

```
intent_id = H("FAST_INTENT_V1" || network_id || base_tx_fingerprint ||
intent_nonce_le || client_created_at_ms_le || max_fee_le)
```

6.3 Additional Rules

- `mass` is not part of `intent_id`.
- `tx.id()` is not the sole normative basis of `intent_id`.
- Interoperability test vectors for RPC, P2P, and storage are mandatory.

---

7. Time, Spendability, and Dependency Rules

7.1 Time Rules

A FastIntent is only admissible if:

```
abs(local_receive_ms - client_created_at_ms) <= 500
```

The effective local TTL is:

```
expires_at_ms = local_receive_ms + 3000
```

7.2 Spendability Rule

A FastIntent is only admissible if all inputs of `base_tx` are fully resolvable against the local **confirmed spendable** chain view at admission time.

7.3 Forbidden Dependency Forms

The following are disallowed:

- inputs from unconfirmed parents,
- fast-on-fast dependencies,
- inputs resolvable only through mempool dependency graphs.

If inputs are not resolvable, the result is `missing_inputs` or `invalid_base_tx`, depending on the validation path.

---

8. Public API Specification

8.1 `submit_fast_intent(base_tx, intent_nonce, client_created_at_ms, max_fee)`

Success Response

Returns:

- `intent_id`
- `status`
- `node_epoch`
- `expires_at_ms`
- `retention_until_ms`
- optional `cancel_token`

Rejection Response

Returns:

- `status = rejected`
- `reason`
- `node_epoch`
- optional `retention_until_ms`

Idempotency

Resubmitting the same `intent_id` must not change state. The response must return the complete current local view:

- `status`
- optional `reason`
- `node_epoch`
- optional `expires_at_ms`
- optional `retention_until_ms`
- optional `cancel_token`, but only if still valid in the current node-local context

---

8.2 `get_fast_intent_status(intent_id, client_last_node_epoch?)`

Known Intent Response

Returns:

- `status`
- optional `reason`
- `node_epoch`
- optional `expires_at_ms`
- optional `retention_until_ms`

Unknown Intent Response

Returns:

- `status = unknown_intent`
- `node_epoch`
- optional `epoch_changed`

Normative Meaning

`unknown_intent` means only that the intent is not currently known on this node. It may indicate:

- never seen locally,
- locally evicted,
- seen on another backend node,
- lost across restart,
- expired retention.

It is not a global statement.

8.3 `cancel_fast_intent(intent_id, cancel_token, node_epoch)`

Success Response

Returns:

- `status = cancelled`
- `reason = cancelled_by_user`
- `node_epoch`

Error Response for Known Active Intent

Returns:

- `status = rejected`
- `reason = cancel_token_invalid | cancel_unauthorized | node_restart`
- `node_epoch`

Response for Known Retained Terminal Intent

If the intent is already terminal but still retained, the RPC returns the existing terminal state unchanged:

- `status = expired | dropped | rejected | cancelled`
- optional `reason`
- `node_epoch`

Unknown Response

Returns:

- `status = unknown_intent`
- `node_epoch`
- optional `epoch_changed`

Normative `epoch_changed` Semantics for Cancel

For `cancel_fast_intent`, `epoch_changed = true` means that the `node_epoch` supplied in the request does not match the current local `node_epoch`.

Cancel Semantics

Cancel is node-local only. It does not imply reversal of any Basechain transaction and is not propagated through P2P.

---

9. Public Error Classes

9.1 Fast RPC Reason Codes

- `temporarily_rejected_fast_lock_conflict`
- `clock_drift_exceeded`
- `fee_rate_below_floor`
- `max_fee_exceeded`
- `ttl_expired`
- `rail_overloaded`
- `rail_disabled`
- `node_unsynced`
- `node_restart`
- `invalid_base_tx`
- `duplicate_intent`
- `duplicate_base_tx_active`
- `missing_inputs`
- `nonstandard_base_tx`
- `base_tx_already_in_mempool`
- `base_tx_already_onchain`
- `invalidated_by_chain_update`
- `cancel_token_invalid`
- `cancel_unauthorized`
- `cancelled_by_user`

9.2 Normal-Path External Error Class

`normal_policy_reject_fast_lock_conflict`

This is a stable external error class for normal transactions rejected because of an active fast lock. It is **not** part of the Fast RPC reason catalogue.

---

10. Pre-Lock State Visibility

Because `received` and `validated` are public statuses, the implementation MUST make them externally observable on a best-effort basis.

This may be implemented either by:

- treating pre-lock entries as part of the logical FastStateStore, or
- maintaining a separate bounded pre-commit status store.

In either case:

- `received` and `validated` are public statuses,
- they are best-effort visible through `get_fast_intent_status`,
- they remain bounded and may be evicted under pressure.

---

11. Admission Rules and Ordering

11.1 Mandatory Admission Sequence

The implementation MUST process fast admission in exactly this order:

1. Check rail enabled state, mode, sync state, and global breaker.
2. Check structure, count, size, and mass limits.
3. Check duplicate, seen-cache, and fingerprint short-circuits.
4. Check drift and time rules.
5. Check fee floor and policy preconditions.
6. Perform full base transaction validation.
7. Perform arbiter conflict check.
8. Acquire atomic input locks.
9. Write state to FastStateStore.
10. Optionally hand off to local microbatching.

This order is security-relevant and DoS-relevant and MUST NOT be reordered.

11.2 Admission Cutoff

- Before successful lock commit, mode changes may still yield `rejected(rail_disabled | rail_overloaded)`.
- After lock commit, the state MUST NOT be reprojected back to `rejected`.
- After `locked`, only normal follow-up transitions are allowed: `fast_confirmed`, `expired`, `dropped`, or `cancelled`.

---

12. ConflictArbiter Specification

12.1 Responsibilities

The arbiter only:

- checks conflicts,
- atomically reserves inputs or rejects,
- returns a result.

12.2 Forbidden Work

The arbiter MUST NOT:

- perform signature validation,
- perform script validation,
- perform disk I/O,
- perform network I/O,
- scan unrelated global pending state,
- perform nested callbacks into foreign subsystems.

12.3 Complexity and Fairness

- Worst-case per operation MUST be `O(inputs)`.
- No arbiter operation may be `O(total_pending)`.
- FIFO is required within each lane.
- Separate lanes exist for normal path and fast path.
- The normal lane MUST never be delayed behind the fast lane.

12.4 Queue Overflow

- The fast arbiter queue is bounded.
- On fast queue overflow, new fast submissions are immediately rejected with `rail_overloaded`.
- The normal path MUST NOT wait for fast queue drain.
- Sustained fast-lane saturation triggers mode transitions to `degraded` and then `paused`.

---

13. State Transition Model

Transitions not listed below are invalid and result in no-op plus error telemetry.

13.1 Valid Transitions

- `received -> rejected` on parse or precheck failure
- `received -> received` on duplicate intent, returning idempotent snapshot
- `received -> rejected` on duplicate active base fingerprint
- `received -> validated` when full validation succeeds

- `validated -> rejected` on mode disallow before commit
- `validated -> rejected` on arbiter conflict
- `validated -> locked` on successful lock acquisition
- `locked -> fast_confirmed` on local microblock inclusion after full local possession and validation
- `locked -> expired` when TTL elapses
- `locked -> dropped` on chain invalidation
- `locked -> cancelled` on valid cancel
- `fast_confirmed -> expired`
- `fast_confirmed -> dropped`
- `fast_confirmed -> cancelled`
- terminal retained states return themselves on cancel requests
- terminal retained states may later be internally evicted and thereafter project as `unknown_intent`

---

14. Duplicate and Conflict Matrix

- same `intent_id` again → idempotent, return current snapshot
- same `base_tx_fingerprint`, different nonce, active intent present → `duplicate_base_tx_active`
- same input set, different transaction content → `temporarily_rejected_fast_lock_conflict`
- base transaction already in normal mempool → `base_tx_already_in_mempool`
- base transaction already on-chain → `base_tx_already_onchain`
- normal claim active on an input → `temporarily_rejected_fast_lock_conflict`
- active fast lock versus competing normal transaction → `normal_policy_reject_fast_lock_conflict`
- after eviction or restart → `unknown_intent`

`base_tx_already_onchain` is an admission-only rejection reason and is not reused for later revalidation projection.

---

15. node_epoch and Restart Semantics

- `node_epoch` is generated exactly once per process start.
- It does not change on pause, resume, enable, disable, or mode transition.
- Cancel requires exact equality with current `node_epoch`.
- Epoch mismatch on cancel returns `rejected(node_restart)`.
- After restart, old fast state is typically lost and requests will return `unknown_intent`.

Clients SHOULD supply their last known `node_epoch` when querying status so they can detect local context change.

---

16. Mode Controller and Thresholds

Evaluation interval is 1 second using a 10-second sliding window.

16.1 `normal -> degraded`

Transition occurs if, for at least 3 consecutive seconds, any one of the following holds:

- `fast_arbiter_queue_ratio >= 0.70`
- `fast_validation_queue_ratio >= 0.80`
- `p95_fast_arbiter_wait_ms >= 15`
- `fast_worker_cpu_ratio >= 0.75`
- `fast_overload_reject_rate_10s >= 3%`
- `fast_pull_fail_rate_10s >= 10%`

16.2 `degraded -> paused`

Transition occurs if, for at least 2 consecutive seconds, any one of the following holds:

- `fast_arbiter_queue_ratio >= 0.90`
- `fast_validation_queue_ratio >= 0.95`
- `p95_fast_arbiter_wait_ms >= 40`
- `fast_worker_cpu_ratio >= 0.90`
- `basechain_latency_delta_p95 >= 10%`
- `revalidation_backlog_seconds >= 5`
- `node_synced = false` immediately

16.3 Resume Hysteresis

- `paused -> degraded` only after all pause triggers remain below threshold for 30 seconds
- `degraded -> normal` only after all degrade triggers remain below threshold for 60 seconds

Low-water marks:

- queue ratios `<= 0.40`
- wait `<= 8 ms`
- CPU ratio `<= 0.50`
- overload reject rate `<= 0.2%`
- pull fail rate `<= 2%`

16.4 Mode Effects

- `normal`: standard quotas, `microblock_interval = 100 ms`
- `degraded`: stricter quotas, more aggressive early rejects, `microblock_interval = 200 ms`
- `paused`: no new fast submits accepted, no microblock building, Basechain continues normally

---

17. P2P and Pull-on-Miss

- Fast P2P is enabled only when both peers advertise capability.
- Remote microblocks are never blindly trusted.
- Missing intents are retrieved best-effort via bounded `RequestFastIntents`.
- No explicit `FastIntentNotFound` exists.
- When retry or wait budgets are exhausted, the affected microblock context is dropped internally.

This path remains intentionally bounded and subordinate to node protection.

---

18. Retention and Storage Limits

- Terminal states are kept best-effort for a bounded time.
- `retention_until_ms` is a planned retention horizon, not a guarantee.
- Hard limits on entry count and byte size are mandatory.
- Eviction order is:
    1. expired terminal entries first,
    2. then oldest terminal entries.

Required telemetry includes:

- terminal entry count,
- terminal byte usage,
- terminal evictions by cause.

---

19. Wallet and Client Guidance

Wallets and clients MUST represent fast-path semantics accurately:

- `fast_confirmed` must be displayed as local and non-final
- `unknown_intent` must never be described as global non-existence
- `cancelled` must be described as a local fast-context result
- node-affine workflows SHOULD be used for submit, status, and cancel
- sticky routing or node-affine sessions are recommended behind load balancers

Recommended labels:

- `locked`: “Fast path active (local)”
- `fast_confirmed`: “Fast confirmed (local, not final)”
- `unknown_intent`: “Currently unavailable on this node”
- `cancelled`: “Local fast path ended”

---

20. Security, Abuse Resistance, and Stability

Mandatory safeguards include:

- per-peer fast message rate limits,
- per-session or per-auth RPC quotas,
- bounded in-flight submissions,
- bounded queues for all fast entry points,
- separate fast workers with CPU budget cap,
- cheap prechecks before expensive validation,
- bounded validation caches with TTL and LRU eviction,
- coalesced and budgeted revalidation,
- stable telemetry and alerts.

The protection model is explicit: under stress, the fast rail is reduced or paused before the Basechain is allowed to degrade.

---

21. Observability

Required metrics include at minimum:

- `fast_active_locks`
- `fast_pending_intents`
- `fast_submit_total{source,result,reason}`
- `fast_reject_total{reason}`
- `fast_drop_total{reason}`
- `fast_expired_total`
- `fast_terminal_entries`
- `fast_terminal_bytes`
- `fast_terminal_evictions_total{cause}`
- `fast_arbiter_queue_len`
- `fast_arbiter_wait_ms`
- `fast_arbiter_hold_ms`
- `fast_pull_miss_total`
- `fast_pull_fail_total{reason}`
- `fast_mode_transition_total{from,to}`
- `basechain_block_latency_ms`
- `basechain_latency_delta_vs_baseline_ms`

Required alerts include:

- Basechain latency delta above threshold,
- fast rail stuck in `paused`,
- arbiter queue sustained near limit,
- terminal evictions rising sharply,
- pull-on-miss failure rate increasing sharply.

---

22. Implementation Work Packages

1. Protocol and data types
2. FastStateStore, ConflictArbiter, AdmissionPipeline
3. RPC endpoints and stable error projection
4. P2P flows and capability negotiation
5. ModeController and resource budgets
6. RetentionStore and eviction
7. Telemetry and alerts
8. Test, fuzzing, and abuse harnesses

---

23. Test Strategy

Required test groups:

- deterministic vector tests for canonicalization and ID derivation,
- transition table tests,
- duplicate and conflict matrix tests,
- restart and `node_epoch` tests,
- cancel semantics tests,
- P2P fuzzing,
- arbiter property tests,
- abuse tests for flood, expensive valid intents, and pull-on-miss exhaustion,
- wallet contract tests for public status and error semantics.

Acceptance criteria:

- no Basechain stalls under fast-path attack,
- no unbounded growth,
- no double input claims,
- deterministic `intent_id`,
- consistent RPC semantics,
- correct `unknown_intent` behavior after restart and retention expiry.

---

24. Rollout Plan

Phase A0

Feature flag, API contracts, metrics, no-op path

Phase A1

Admission, arbiter, locking, TTL, retention

Phase A2

Local fast microblocks, capability-gated P2P, pull-on-miss

Phase A3

DoS hardening, circuit breaker, load tests

Phase A4

Testnet rollout

Phase A5

Optional mainnet enablement, default remains `fast_enabled = false`

Rollback

Fast rail may be paused or disabled via configuration. No consensus migration is required. Basechain continues unchanged.

---

25. Configuration Defaults

- `fast_enabled = false`
- `fast_ttl_ms = 3000`
- `clock_drift_max_ms = 500`
- `microblock_interval_ms_normal = 100`
- `microblock_interval_ms_degraded = 200`
- `max_inputs_per_intent = 32`
- `max_outputs_per_intent = 64`
- `fast_min_feerate_floor = 2x minimum_relay_feerate`
- `max_pending_intents = 50_000`
- `max_locks = 200_000`
- `max_terminal_entries = 200_000`
- `max_terminal_bytes = 128 MiB`
- `terminal_retention_ms = 120_000`
- `seen_cache_ttl_ms = 600_000`
- `normal_arbiter_queue_capacity = 4096`
- `fast_arbiter_queue_capacity = 4096`

---

26. Assumptions

1. HF-A prioritizes liveness and robustness over maximum flexibility.
2. Local acceleration is the product goal, not global fast finality.
3. Node operators explicitly accept that under pressure the fast function yields before the chain does.
4. Wallets and clients can treat `node_epoch` as node-local session context.
5. Node clocks are sufficiently synchronized for the configured drift window.

---

27. Conclusion

HF-A, in this form, is a local, decentralized, strictly bounded fast transfer rail that can significantly improve transfer UX without modifying the safety, settlement, or finality guarantees of the Basechain.

The architecture is intentionally conservative. That constraint is exactly what makes HF-A robust, operable, and realistically implementable on top of the existing node architecture.