# **M3 — Native token support, execution simulator, timelock governance, stronger slippage, & circuit breakers (≤ ~220 steps)**

## **A. Branch, baseline, & environment**

1. Create branch feature/m3-native-sim-timelock.
2. Tag current commit pre-m3-cut (post-M2 green).
3. Add new env vars to .env.example: TIMELOCK\_DELAY\_SEC, TIMELOCK\_ADDRESS (optional), MULTISIG\_PROPOSER, MULTISIG\_EXECUTOR.
4. Add deployments/sepolia.m3.template.json to capture Timelock + updated adapter/executor addresses.
5. Add PNPM script "env:check:m3" that asserts **all** M2 envs plus new M3 ones exist; exit non-zero if missing.
6. Create docs/dev/m3-overview.md (skeleton): purpose, features, flags, links to runbooks.

## **B. Solidity — UniswapV3Adapter native (wrap/unwrap) & surfaces**

### **B1. New Native sentinel & WETH interface**

1. Create contracts/src/common/Native.sol:  
   * library Native { address constant NATIVE = address(0); }
2. Add contracts/src/interfaces/common/IWETH.sol with deposit(), withdraw(uint256), transfer, approve.

### **B2. Adapter constructor & storage**

1. Open contracts/src/venues/adapters/UniswapV3Adapter.sol.
2. Import IWETH and Native.
3. Add immutable IWETH public WETH;.
4. Change constructor to take \_weth and set WETH = \_weth.
5. Extend contract to accept ETH: add receive() external payable {}.

### **B3. Execute: single-hop native in**

1. Modify execute(bytes) signature to be external payable.
2. In **single-hop** path:  
   * Decode params.
   * If sp.tokenIn == Native.NATIVE:  
     + require(msg.value == sp.amountIn, "value/amountIn");
     + WETH.deposit{value: sp.amountIn}();
     + Use address(WETH) as actual input token for approvals.
3. Post-swap delivery for single hop:  
   * If sp.recipient == Native.NATIVE && sp.tokenOut == address(WETH):  
     + WETH.withdraw(amountOut);
     + payable(msg.sender).call{value: amountOut}("") (check success).
   * Else transfer ERC-20 to sp.recipient.
4. Ensure approval hygiene: approve(0) → approve(amount) → approve(0) on tokenIn\_.

### **B4. Execute: multi-hop native edges**

1. In **multi-hop** path:  
   * Decode struct (path, recipient, amountIn, minOut, deadline).
   * If first token sentinel is native:  
     + require(msg.value == mi.amountIn, "value/amountIn");
     + WETH.deposit{value: mi.amountIn}();
     + Ensure path actually starts with WETH address (validation convenience).
2. After exactInput:  
   * If mi.recipient == Native.NATIVE **and** last token is WETH:  
     + WETH.withdraw(amountOut);
     + send ETH to caller with safe call.
   * Else transfer ERC-20 out to recipient.
3. Implement pure helper \_lastToken(bytes memory path) returns (address) that reads final 20-byte slice.
4. Emit existing AdapterExecuted with original tokenIn/tokenOut (pre-wrap/unwrap) in args for observability.

### **B5. Guards & errors alignment**

1. In contracts/src/policy/Errors.sol add error Unauthorized();.
2. Ensure all new failure paths use custom errors where appropriate; keep string literals only for low-risk “internal mismatch” checks.

## **C. Solidity — PolicyGuards stricter slippage & TWAP knobs**

1. Open contracts/src/policy/PolicyGuards.sol.
2. Add state: uint16 public maxSlippageBps = 100; and uint32 public twapWindow = 300;.
3. Add setters: setMaxSlippageBps(uint16), setTwapWindow(uint32) with onlyOwner and events.
4. Keep path shape, fee allowlist, and maxSteps from M2 intact.
5. Document (NatSpec) that **native** is permitted **only at endpoints** (enforced in adapter).

## **D. Solidity — BundleExecutor pause & circuit breaker**

1. Open contracts/src/execution/BundleExecutor.sol.
2. Add bool public paused; and uint256 public maxTotalGas;.
3. Add event Paused(bool value) and event MaxTotalGasSet(uint256 value).
4. Add external setters callable **only by the same owner as PolicyGuards** (use guards.owner() check + Errors.Unauthorized()):  
   * setPaused(bool)
   * setMaxTotalGas(uint256)
5. In execute(...):  
   * if (paused) revert Errors.Unauthorized();
   * Track per-step gas and accumulate totalGas.
   * If maxTotalGas != 0 && totalGas > maxTotalGas → revert "gas breaker".
6. Keep per-step and total minOut logic from M2.

## **E. Governance — Timelock + Multisig handoff**

### **E1. Timelock contract**

1. Add contracts/src/governance/TimelockConfig.sol extending OZ TimelockController.
2. Constructor takes (minDelay, proposers[], executors[]).

### **E2. Deployment scripts (Foundry)**

1. Create script/DeployTimelock.s.sol returning Timelock address.
2. Update script/Deploy.s.sol to:  
   * Deploy Timelock when TIMELOCK\_ADDRESS not provided.
   * Pass M3 WETH into the adapter constructor.
3. Update script/Configure.s.sol to:  
   * guards.setMaxSlippageBps(100);
   * guards.setTwapWindow(300);
   * executor.setMaxTotalGas(5\_000\_000);
4. Create script/OwnershipTransfer.s.sol:  
   * Transfer PolicyGuards and RouteRegistry ownership to **Timelock**.
   * (Executor control via guards.owner() gate is therefore timelocked).
5. Write script/TimelockQueueExecute.s.sol helper that demonstrates queuing/executing:  
   * Example: setMaxSteps(8).

### **E3. Docs & ops**

1. Draft docs/runbooks/m3-ops.md detailing:  
   * Timelock deployment,
   * grant role: PROPOSER\_ROLE to MULTISIG\_PROPOSER; EXECUTOR\_ROLE to MULTISIG\_EXECUTOR,
   * ownership transfer verification,
   * how to queue/cancel operations,
   * emergency pause via timelock.

## **F. TypeScript — deterministic simulator & price guards**

### **F1. Simulator module**

1. Add packages/core-exec/src/sim/simulator.ts with:  
   * type SimInput { steps; amountIn; maxSlippageBps? }
   * type SimResult { expectedOut; perStep; twapOk; slippageOk }
   * simulate(publicClient, input) orchestrating quotes and twap check.
2. Call into quoteMultiHop / quoteSingle (next section).
3. Read policy defaults via loadPolicies().

### **F2. TWAP guard (stubbed wire)**

1. Create packages/core-exec/src/sim/twap.ts exporting getTwap(client, steps): Promise<number>.
2. For MVP, return 0 (no drift) but expose TODO in comment for Uniswap V3 observations integration.
3. Add unit seam so future PR can inject oracle.

### **F3. Quoter helpers**

1. Create packages/core-exec/src/quote/univ3.ts:  
   * quoteMultiHop(client, payload, amountIn) calling QuoterV2.quoteExactInput.
   * quoteSingle(client, payload, amountIn) calling quoteExactInputSingle.
2. Implement minimal ABI decode helpers to extract path/single params from payload (or call shared encoder/decoder).
3. Ensure bigints returned; throw on revert with informative error.

### **F4. Policies reader**

1. Create packages/core-exec/src/util/policies.ts:  
   * loadPolicies() reads config/policies.json,
   * returns typed object { maxSlippageBps, twapWindowSec, circuitBreakers }.

## **G. TypeScript — payloads & native sentinel**

### **G1. Constants**

1. In packages/core-exec/src/encode.ts, export:  
   * TAG\_UNIV3\_SINGLE, TAG\_UNIV3\_MULTI, NATIVE\_SENTINEL = "0x000...000".
2. Document: sentinel allowed only as **first** or **final** token in path.

### **G2. Encoders**

1. Ensure encodeUniV3Path(hops) accepts NATIVE\_SENTINEL at endpoints but internally agents must convert to WETH for router; adapter performs wrap/unwrap so **path must still encode WETH address** (document this).
2. Expose helpers to pack/unpack uint24 fees correctly.

### **G3. Adapter client**

1. In packages/adapters-evm/src/UniswapV3AdapterClient.ts:  
   * Add buildSingleHopNativeIn and buildMultiHopNativeIn/Out convenience wrappers which set recipient/tokenIn with sentinel and encode payload with the correct tag.
2. Validate at client level that native sentinel is only used at endpoints; otherwise throw.

### **G4. RouteBuilder parity**

1. Update packages/core-exec/src/RouteBuilder.ts:  
   * Keep step continuity validation from M2,
   * Allow continuity when endpoint is sentinel ↔️ WETH (treat as equivalent).
2. Expose derivePerStepMinOut(quotes, bps) helper to calculate minOuts from simulator quotes.

## **H. CLI — simulate & execute with simulator-derived minOut**

1. Open packages/cli/src/commands/route.ts.
2. Add subcommand route simulate --route-file <json> [--max-slippage-bps N]:  
   * Reads file, calls simulator, prints structured JSON result.
3. Add subcommand route execute-with-sim --route-file <json>:  
   * Runs simulation,
   * Applies policy bps to compute minTotalOut,
   * Executes route using existing execute flow.
4. Ensure both commands support --json output.
5. Add helpful errors if twapOk === false or if quotes missing.

## **I. Config — venues & policies expansion**

1. Update config/venues.json:  
   * Under each network add "native": "0x000...000" and "weth": "<addr>".
2. Update config/policies.json:  
   * Add "maxSlippageBps": 100,
   * Add "twapWindowSec": 300,
   * Add "circuitBreakers": { "bundle.maxTotalGas": 5000000 }.
3. Add validation script scripts/dev/validate-config.ts:  
   * Ensures weth present when native sentinel configured,
   * Ensures circuit breaker value numeric and > 0.

## **J. Foundry tests — native, breakers, ownership through timelock**

### **J1. Adapter native unit tests**

1. Create contracts/test/venues/UniswapV3Adapter.native.t.sol.
2. Case: single-hop native in → wraps WETH, swaps, transfers ERC-20 out.
3. Case: single-hop native out (WETH out) → unwraps and pays ETH to caller (via recipient sentinel).
4. Case: multi-hop native in → wraps WETH; path starts with WETH.
5. Case: multi-hop native out → unwraps last WETH out to ETH.
6. Case: revert on msg.value != amountIn.
7. Case: approval hygiene check around router calls (zeroed after).

### **J2. Executor breakers unit tests**

1. Create contracts/test/execution/Executor.breakers.t.sol.
2. Verify paused=true reverts with Unauthorized().
3. Verify gas breaker triggers when accumulated per-step gas > limit.

### **J3. Timelock tests**

1. Create contracts/test/governance/Timelock.ownership.t.sol.
2. Deploy timelock, transfer guards ownership to timelock.
3. Queue + execute setMaxSlippageBps(80); assert change applied.
4. Attempt direct EOA setter call; expect revert.

## **K. Vitest (TS) — simulator, CLI, parity**

### **K1. Simulator unit tests**

1. Add packages/core-exec/test/sim.simulator.spec.ts.
2. Mock quoter calls: return deterministic amounts per step.
3. Test perStep array lengths & accumulation to expectedOut.
4. twapOk true for stub (0 drift).
5. Respect custom maxSlippageBps override in input.

### **K2. Quoter helpers on fork**

1. Add packages/core-exec/test/quote.univ3.fork.spec.ts (guarded with @fork):  
   * Use Anvil fork,
   * Run quoteSingle/quoteMultiHop vs chain quoter; compare outputs.

### **K3. CLI simulate + execute**

1. Add packages/cli/test/route.simulate.exec.spec.ts.
2. Provide route json fixture with multi-hop and sentinel on endpoint.
3. Assert simulator JSON includes expectedOut and perStep.
4. Run execute-with-sim dry-run (mock provider); assert minTotalOut computed and passed.

## **L. Observability — metrics & event wiring**

1. Extend packages/core-exec/src/telemetry/metrics.ts with observe(name, value, labels?).
2. Wire packages/core-exec/src/telemetry/indexer.ts to:  
   * Increment bundles\_executed\_total,
   * Sum bundle\_gas\_sum from BundleExecuted.totalGas.
3. Add PNPM script "dev:metrics": "tsx packages/core-exec/src/telemetry/server.ts".
4. Create minimal /metrics server (Prometheus text), gated by TELEMETRY\_BIND env; document dev-only.

## **M. CI pipeline — new jobs & order**

1. Update .github/workflows/ci.yml:  
   * Add contracts-native Foundry job (unit),
   * Add contracts-breakers Foundry job,
   * Add ts-sim Vitest job,
   * Ensure ts-fork-tests (quoter) runs after contract compile but before integration,
   * Keep fork tests non-blocking or quarantine if flaky; report artifacts.
2. Upload artifacts: telemetry/events.ndjson, fork test logs, gas snapshots.

## **N. Staging deploy & smoke**

1. Fill deployments/sepolia.m3.template.json with live addresses after deploy.
2. Deploy M3 on Sepolia: adapter (with WETH), executor unchanged bytecode (if adds), timelock (if needed).
3. Run Configure.s.sol and set:  
   * maxSlippageBps=100, twapWindow=300, maxTotalGas=5\_000\_000.
4. Transfer ownership to Timelock; grant proposer/executor to multisig.
5. Seed staging accounts with ETH + relevant tokens (via faucets/mints).
6. Build a native-in route (ETH→WETH→USDC) and register it.
7. Run CLI route simulate and inspect outputs; persist JSON to ./.out/m3-sim.json.
8. Run CLI route execute-with-sim and capture tx hash.
9. Verify BundleExecuted fires; check metrics counters increased.

## **O. Acceptance tests (objective sign-off)**

1. Create acceptance/m3.md with pass/fail gates:  
   1. Native wrap/unwrap succeeds (both directions).
   2. msg.value mismatch reverts.
   3. Circuit breaker: exceed gas → revert; normal path succeeds.
   4. paused=true blocks bundle.
   5. Simulate→Execute pipeline sets minTotalOut and succeeds on fork/staging.
   6. Timelock controls guards; EOA cannot call setters.
   7. Metrics counters reflect real execution.
   8. Approval hygiene holds post-swap.
2. Run each acceptance step on staging; attach artifacts (tx hashes, logs, screenshots).

## **P. Security hardening (M3 scope)**

1. Re-run Slither (or equivalent) on updated contracts; stash report under artifacts/security/m3-slither.json.
2. Manual review checklist:  
   * Adapter: wrapping scope limited; no arbitrary ETH drains; correct receive().
   * Executor: breakers & pause only via timelock.
   * Guards: setters timelocked; events emitted.
3. Verify **no** externalizable reentrancy on native transfer (use post-state updates before call{value:...} if any state is mutated).
4. Confirm adapter never sends ERC-20 to unexpected recipients; only to specified recipient or caller on native unwrap.
5. Record known limitation: TWAP check is **stubbed**; scheduled for M3.x (or M4).

## **Q. Gas & performance baselines (M3)**

1. Add forge snapshot for:  
   * single-hop native in→ERC20,
   * multi-hop native in→ERC20,
   * native out (WETH→ETH unwrap).
2. Update docs/dev/gas.md with new tables & ±15% guardrails.
3. Set per-test gas caps in GAS\_LIMIT\_JSON for new tests; wire assertions.

## **R. Docs (developer & ops)**

1. Update docs/content/uc4-multiswap-permissions.mdx:  
   * Add section: “Native token edges” with sentinel rules.
   * Diagram: wrap at entry, unwrap at exit.
2. Update README.md:  
   * Add quickstart: building a native-in multi-hop route.
   * New CLI simulate and execute-with-sim snippets.
3. Add docs/runbooks/m3-ops.md (filled earlier) with:  
   * Timelock ops,
   * Emergency pause,
   * Rollback: disable route + pause executor,
   * Resetting breakers.

## **S. Backward compatibility & migrations**

1. docs/dev/migrations/m3.md:  
   * M2 routes continue to run (ERC-20 only).
   * Native sentinel is opt-in; old clients ignore it.
   * Ownership transfer: from multisig (M2) → timelock (M3) with multisig as proposer/executor.
2. CLI behavior:  
   * If user tries native sentinel mid-path, validation fails with actionable message.
   * If policies.json missing new keys, default to safe values and warn.

## **T. Release packaging**

1. Bump versions to 0.3.0-m3 in:  
   * packages/adapters-evm/package.json
   * packages/core-exec/package.json
   * packages/cli/package.json
2. Regenerate ABIs → packages/\*/dist/abi.
3. Create m3-release-notes.md:  
   * Features, configs, governance, known limits (TWAP stub), upgrade steps.
4. Tag m3-native-sim-timelock.
5. Merge feature/m3-native-sim-timelock → main after CI green.

## **U. Copy-pasteable deltas (key snippets)**

### **U1. Adapter constructor change (Solidity)**

constructor(IUniswapV3Router \_router, IQuoterV2 \_quoter, IWETH \_weth) {

router = \_router;

quoter = \_quoter;

WETH = \_weth;

}

### **U2. Native sentinel constant**

library Native { address constant NATIVE = address(0); }

### **U3. Executor breakers**

function setPaused(bool v) external {

if (msg.sender != guards.owner()) revert Errors.Unauthorized();

paused = v; emit Paused(v);

}

function setMaxTotalGas(uint256 v) external {

if (msg.sender != guards.owner()) revert Errors.Unauthorized();

maxTotalGas = v; emit MaxTotalGasSet(v);

}

### **U4. Policy knobs**

uint16 public maxSlippageBps = 100;

uint32 public twapWindow = 300;

function setMaxSlippageBps(uint16 bps) external onlyOwner { maxSlippageBps = bps; emit SlippageBpsSet(bps); }

function setTwapWindow(uint32 sec) external onlyOwner { twapWindow = sec; emit TwapWindowSet(sec); }

### **U5. TS simulate entrypoint**

export async function simulate(client: PublicClient, input: SimInput): Promise<SimResult> {

const policies = await loadPolicies();

const maxBps = input.maxSlippageBps ?? policies.maxSlippageBps ?? 100;

const perStep: bigint[] = [];

let running = input.amountIn;

for (const s of input.steps) {

running = s.kind === "UNIV3\_MULTI"

? await quoteMultiHop(client, s.payload, running)

: await quoteSingle(client, s.payload, running);

perStep.push(running);

}

const twapOk = (await getTwap(client, input.steps)) <= maxBps;

return { expectedOut: running, perStep, twapOk, slippageOk: true };

}

### **U6. CLI simulate & execute**

route.command("simulate")

.requiredOption("--route-file <path>")

.option("--max-slippage-bps <num>")

.action(async (opts) => { /\* read file, run simulate(), print JSON \*/ });

route.command("execute-with-sim")

.requiredOption("--route-file <path>")

.action(async (opts) => { /\* run simulate, derive minTotalOut, execute \*/ });

## **V. “Definition of done” (what success looks like)**

1. Foundry: native, breakers, timelock tests pass locally and in CI.
2. TS: simulator/CLI test suites pass; fork quotes match chain quoter (within tolerance).
3. Staging run shows successful native swap, simulator-driven execution, metrics updated.
4. Timelock fully controls guard knobs; pause & breaker operable via queued ops.
5. Docs & runbooks published; release tagged m3-native-sim-timelock.