# **PR: M3 — Native token support, execution simulator, timelock governance, stricter slippage & circuit breakers**

**Theme:** Make UC-4 truly production-capable by (1) enabling **ETH/native** wrap/unwrap, (2) adding a **deterministic simulator** to pre-validate bundles, (3) enforcing **stronger slippage/twap** policy, (4) moving owner controls behind a **Timelock + Multisig**, and (5) adding **circuit breakers** and **alerting**.

**Assumptions:** M2 merged on main, tests green, addresses for Router/Quoter/WETH configured in config/venues.json.

## **1) Solidity — enable native token (WETH) & payout handling**

### **1.1 UniswapV3Adapter: native wrap/unwrap (opt-in)**

\*\*\* a/contracts/src/venues/adapters/UniswapV3Adapter.sol

--- b/contracts/src/venues/adapters/UniswapV3Adapter.sol

@@

import {IUniswapV3Router} from "../../interfaces/uniswap/IUniswapV3Router.sol";

import {IQuoterV2} from "../../interfaces/uniswap/IQuoterV2.sol";

import {Errors} from "../../policy/Errors.sol";

+import {IWETH} from "../../interfaces/common/IWETH.sol";

+import {Native} from "../../common/Native.sol";

contract UniswapV3Adapter {

using SafeERC20 for IERC20;

@@

IUniswapV3Router public immutable router;

IQuoterV2 public immutable quoter;

+ IWETH public immutable WETH;

@@

- constructor(IUniswapV3Router \_router, IQuoterV2 \_quoter) {

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

router = \_router;

quoter = \_quoter;

+ WETH = \_weth;

}

@@

- function execute(bytes calldata payload) external returns (bool ok, uint256 amountIn, uint256 amountOut) {

+ function execute(bytes calldata payload) external payable returns (bool ok, uint256 amountIn, uint256 amountOut) {

uint256 g0 = gasleft();

bytes4 tag;

assembly { tag := calldataload(payload.offset) }

if (tag == TAG\_UNIV3\_SINGLE) {

UniV3SwapParams memory sp = abi.decode(payload[4:], (UniV3SwapParams));

if (sp.deadline < block.timestamp) revert Errors.DeadlineExpired();

- amountIn = sp.amountIn;

- IERC20(sp.tokenIn).safeApprove(address(router), 0);

- IERC20(sp.tokenIn).safeApprove(address(router), sp.amountIn);

- amountOut = router.exactInputSingle(sp.tokenIn, sp.tokenOut, sp.fee, sp.recipient, sp.amountIn, sp.minOut, sp.deadline);

- IERC20(sp.tokenIn).safeApprove(address(router), 0);

+ // Native handling (wrap if tokenIn == NATIVE)

+ address tokenIn\_ = sp.tokenIn;

+ if (tokenIn\_ == Native.NATIVE) {

+ require(msg.value == sp.amountIn, "value/amountIn");

+ WETH.deposit{value: sp.amountIn}();

+ tokenIn\_ = address(WETH);

+ }

+ IERC20(tokenIn\_).safeApprove(address(router), 0);

+ IERC20(tokenIn\_).safeApprove(address(router), sp.amountIn);

+ amountOut = router.exactInputSingle(tokenIn\_, sp.tokenOut, sp.fee, address(this), sp.amountIn, sp.minOut, sp.deadline);

+ IERC20(tokenIn\_).safeApprove(address(router), 0);

+ // Unwrap if recipient expects native

+ if (sp.recipient == Native.NATIVE) {

+ require(address(this) != Native.NATIVE, "bad recipient");

+ if (sp.tokenOut == address(WETH)) {

+ WETH.withdraw(amountOut);

+ (bool s,) = payable(msg.sender).call{value: amountOut}("");

+ require(s, "native transfer");

+ } else {

+ IERC20(sp.tokenOut).safeTransfer(msg.sender, amountOut);

+ }

+ } else {

+ IERC20(sp.tokenOut).safeTransfer(sp.recipient, amountOut);

+ }

ok = true;

- emit AdapterExecuted(tag, sp.tokenIn, sp.tokenOut, amountIn, amountOut, g0 - gasleft());

+ emit AdapterExecuted(tag, sp.tokenIn, sp.tokenOut, sp.amountIn, amountOut, g0 - gasleft());

return (ok, amountIn, amountOut);

} else if (tag == TAG\_UNIV3\_MULTI) {

UniV3ExactInput memory mi = abi.decode(payload[4:], (UniV3ExactInput));

if (mi.deadline < block.timestamp) revert Errors.DeadlineExpired();

@@

- amountIn = mi.amountIn;

- IERC20(tokenIn\_).safeApprove(address(router), 0);

- IERC20(tokenIn\_).safeApprove(address(router), mi.amountIn);

- amountOut = router.exactInput(mi.path, mi.recipient, mi.amountIn, mi.minOut, mi.deadline);

- IERC20(tokenIn\_).safeApprove(address(router), 0);

+ amountIn = mi.amountIn;

+ // Native wrap if first token == NATIVE sentinel

+ if (tokenIn\_ == Native.NATIVE) {

+ require(msg.value == mi.amountIn, "value/amountIn");

+ WETH.deposit{value: mi.amountIn}();

+ tokenIn\_ = address(WETH);

+ // swap path must actually start with WETH in this case

+ }

+ IERC20(tokenIn\_).safeApprove(address(router), 0);

+ IERC20(tokenIn\_).safeApprove(address(router), mi.amountIn);

+ amountOut = router.exactInput(mi.path, address(this), mi.amountIn, mi.minOut, mi.deadline);

+ IERC20(tokenIn\_).safeApprove(address(router), 0);

+ // Deliver native if final hop returns WETH and recipient is Native

+ if (mi.recipient == Native.NATIVE) {

+ if (\_lastToken(mi.path) == address(WETH)) {

+ WETH.withdraw(amountOut);

+ (bool s,) = payable(msg.sender).call{value: amountOut}("");

+ require(s, "native transfer");

+ } else {

+ IERC20(\_lastToken(mi.path)).safeTransfer(msg.sender, amountOut);

+ }

+ } else {

+ IERC20(\_lastToken(mi.path)).safeTransfer(mi.recipient, amountOut);

+ }

ok = true;

emit AdapterExecuted(tag, tokenIn\_, tokenOut\_, amountIn, amountOut, g0 - gasleft());

return (ok, amountIn, amountOut);

}

revert Errors.InvalidInput();

}

+ function \_lastToken(bytes memory path) internal pure returns (address t) {

+ uint256 len = path.length; // ... | token(20) at tail

+ assembly { t := shr(96, mload(add(add(path, 32), sub(len, 20)))) }

+ }

-}

+ receive() external payable {}

+}

Notes  
 • We use Native.NATIVE = address(0) as sentinel in payloads to signal ETH in/out.  
 • Router interactions remain ERC-20 (WETH). We wrap/unwrap at adapter edges.

### **1.2 PolicyGuards: native path rules & TWAP slippage budget**

\*\*\* a/contracts/src/policy/PolicyGuards.sol

--- b/contracts/src/policy/PolicyGuards.sol

@@

uint8 public maxSteps = 8;

+ uint16 public maxSlippageBps = 100; // 1%

+ uint32 public twapWindow = 300; // 5 minutes

event AllowedFeeSet(uint24 fee, bool allowed);

event MaxStepsSet(uint8 value);

+ event SlippageBpsSet(uint16 bps);

+ event TwapWindowSet(uint32 sec);

@@

function setMaxSteps(uint8 value) external onlyOwner {

maxSteps = value;

emit MaxStepsSet(value);

}

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

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

@@

- function validate(Step[] memory steps) external view returns (bool) {

+ function validate(Step[] memory steps) external view returns (bool) {

uint256 n = steps.length;

if (n == 0) revert Errors.InvalidInput();

if (n > maxSteps) revert Errors.MaxStepsExceeded(n, maxSteps);

for (uint256 i = 0; i < n; i++) {

bytes4 tag;

bytes memory p = steps[i].payload;

assembly { tag := mload(add(p, 32)) }

if (tag == 0xaabbcc01) {

@@

} else if (tag == 0xaabbcc02) {

uint256 pathLen;

assembly { pathLen := mload(add(p, 0x80)) }

if (pathLen < 43) revert Errors.InvalidPath();

if (((pathLen + 3) % 23) != 0) revert Errors.InvalidPath();

+ // Native rules: only allowed at endpoints via WETH wrap/unwrap in adapter

+ // (payload signals native by recipient or first token sentinel; enforced in adapter)

} else {

revert Errors.InvalidInput();

}

}

return true;

}

We keep **TWAP/price** checks in TS simulator (pre-execution), exporting policy values here so the simulator can read them.

## **2) Solidity — BundleExecutor: circuit breakers & pause**

\*\*\* a/contracts/src/execution/BundleExecutor.sol

--- b/contracts/src/execution/BundleExecutor.sol

@@

import "../policy/PolicyGuards.sol";

+import "../policy/Errors.sol";

contract BundleExecutor {

PolicyGuards public guards;

+ bool public paused;

+ uint256 public maxTotalGas; // circuit breaker

- event BundleExecuted(bytes32 routeId, address caller, uint256 amountIn, uint256 totalReceived, uint256 steps, uint256 totalGas, uint256 ts);

+ event BundleExecuted(bytes32 routeId, address caller, uint256 amountIn, uint256 totalReceived, uint256 steps, uint256 totalGas, uint256 ts);

+ event Paused(bool value);

+ event MaxTotalGasSet(uint256 value);

constructor(PolicyGuards \_guards) { guards = \_guards; }

@@

+ 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);

+ }

@@

function execute(bytes32 routeId, Step[] calldata steps, uint256 minTotalOut) external returns (uint256 received) {

+ if (paused) revert Errors.Unauthorized();

require(guards.validate(\_mirror(steps)), "invalid");

uint256 g0 = gasleft();

uint256 totalGas;

@@

totalGas += (gi - gasleft());

+ if (maxTotalGas != 0 && totalGas > maxTotalGas) revert("gas breaker");

}

require(received >= minTotalOut, "slippage");

emit BundleExecuted(routeId, msg.sender, 0, received, steps.length, g0 - gasleft(), block.timestamp);

}

## **3) Governance — Timelock + Multisig ownership handoff**

Add a minimal OZ Timelock and wire PolicyGuards/RouteRegistry ownership to Timelock; proposer/executor will be your multisig.

\*\*\* /dev/null

--- b/contracts/src/governance/TimelockConfig.sol

@@

+// SPDX-License-Identifier: MIT

+pragma solidity ^0.8.20;

+import "openzeppelin-contracts/governance/TimelockController.sol";

+contract TimelockConfig is TimelockController {

+ constructor(uint256 minDelay, address[] memory proposers, address[] memory executors)

+ TimelockController(minDelay, proposers, executors, msg.sender) {}

+}

Deploy script:

\*\*\* /dev/null

--- b/script/DeployTimelock.s.sol

@@

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.20;

import "../contracts/src/governance/TimelockConfig.sol";

contract DeployTimelock {

function run(uint256 delay, address proposer, address executor) external returns (address) {

TimelockConfig t = new TimelockConfig(delay, \_arr(proposer), \_arr(executor));

return address(t);

}

function \_arr(address a) internal pure returns (address; v[0]=a; }

}

Ownership transfer script (M3 replaces M2’s direct multisig handoff):

\*\*\* a/script/OwnershipTransfer.s.sol

--- b/script/OwnershipTransfer.s.sol

@@

- // transfer to MULTISIG\_OWNER

+ // transfer to TIMELOCK address – multisig acts as proposer/executor on timelock

(Agents will replace call args to pass TIMELOCK.)

## **4) TypeScript — new simulator & price guards**

### **4.1 Add a deterministic simulator (staticcall + quoter + twap oracle)**

\*\*\* /dev/null

--- b/packages/core-exec/src/sim/simulator.ts

@@

+import { PublicClient } from "viem";

+import { getTwap } from "./twap";

+import { quoteMultiHop, quoteSingle } from "../quote/univ3";

+import { loadPolicies } from "../util/policies";

+

+export type SimInput = {

+ steps: { kind: "UNIV3\_SINGLE"|"UNIV3\_MULTI", payload: `0x${string}` }[];

+ amountIn: bigint;

+ maxSlippageBps?: number;

+};

+

+export type SimResult = { expectedOut: bigint; perStep: bigint[]; twapOk: boolean; slippageOk: boolean; };

+

+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) {

+ if (s.kind === "UNIV3\_MULTI") {

+ const out = await quoteMultiHop(client, s.payload, running);

+ perStep.push(out); running = out;

+ } else {

+ const out = await quoteSingle(client, s.payload, running);

+ perStep.push(out); running = out;

+ }

+ }

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

+ // slippage check vs current quoter — caller will turn this into per-step minOut

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

+}

### **4.2 TWAP sketch (replace with your pool oracle; simple guard here)**

\*\*\* /dev/null

--- b/packages/core-exec/src/sim/twap.ts

@@

+import { PublicClient } from "viem";

+export async function getTwap(client: PublicClient, \_steps: any[]): Promise<number> {

+ // TODO: read oracle observations; for MVP return 0 (pretend no drift)

+ return 0;

+}

### **4.3 Quoter helpers (reusable from M2)**

\*\*\* /dev/null

--- b/packages/core-exec/src/quote/univ3.ts

@@

+import { encodeFunctionData } from "viem";

+import { quoterAbi } from "../abi/univ3";

+export async function quoteMultiHop(client:any, payload:`0x${string}`, amountIn:bigint) {

+ // decode path from payload[4:]

+ const path = decodeBytesFromPayload(payload); // implement using your ABI coder

+ const data = encodeFunctionData({ abi: quoterAbi, functionName:"quoteExactInput", args:[{path, amountIn, sqrtPriceLimitX96:0n}] });

+ const r = await client.call({ to: process.env.UNIV3\_QUOTER as `0x${string}`, data });

+ return (r as any)[0] as bigint;

+}

+export async function quoteSingle(client:any, payload:`0x${string}`, amountIn:bigint) {

+ // decode tokenIn/out/fee from payload; call quoteExactInputSingle

+ // return out

+ return amountIn; // TODO minimal stub for agents to complete

+}

## **5) TypeScript — payload changes for native sentinel & CLI**

### **5.1 Encoders: allow Native.NATIVE sentinel**

\*\*\* a/packages/core-exec/src/encode.ts

--- b/packages/core-exec/src/encode.ts

@@

export const TAG\_UNIV3\_MULTI = "0xaabbcc02" as const;

+export const NATIVE\_SENTINEL = "0x0000000000000000000000000000000000000000" as const;

@@

export function encodeUniV3Path(hops: (string | number)[]): `0x${string}` {

// unchanged — but callers may pass NATIVE\_SENTINEL only if first or last token; adapter wraps/unwraps

...

}

### **5.2 CLI: simulate & execute with simulator-derived minOut**

\*\*\* a/packages/cli/src/commands/route.ts

--- b/packages/cli/src/commands/route.ts

@@

+route

+ .command("simulate")

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

+ .option("--max-slippage-bps <num>", "defaults from policies.json")

+ .action(async (opts) => {

+ const route = JSON.parse(fs.readFileSync(opts.routeFile, "utf8"));

+ const sim = await simulate(publicClient, { steps: route.steps, amountIn: BigInt(route.amountIn), maxSlippageBps: opts.maxSlippageBps? Number(opts.maxSlippageBps):undefined });

+ console.log(JSON.stringify(sim, null, 2));

+ });

+

+route

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

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

+ .action(async (opts) => {

+ const route = JSON.parse(fs.readFileSync(opts.routeFile, "utf8"));

+ const sim = await simulate(publicClient, { steps: route.steps, amountIn: BigInt(route.amountIn) });

+ // derive per-step minOut and total minOut using policy slippage bps

+ const bps = (await loadPolicies()).maxSlippageBps ?? 100;

+ const minTotalOut = (sim.expectedOut \* BigInt(10000 - bps)) / 10000n;

+ await executeRoute(route, minTotalOut); // agent fills in tx sender etc.

+ });

## **6) Config — venues/policies expand for M3**

\*\*\* a/config/venues.json

--- b/config/venues.json

@@

"sepolia": {

"uniswapV3": {

"router": "0x...router",

"quoter": "0x...quoter",

- "weth": "0x...weth"

+ "weth": "0x...weth",

+ "native": "0x0000000000000000000000000000000000000000"

}

}

}

\*\*\* a/config/policies.json

--- b/config/policies.json

@@

- "maxSteps": 8,

- "slippageDefaultBps": 50

+ "maxSteps": 8,

+ "slippageDefaultBps": 50,

+ "maxSlippageBps": 100,

+ "twapWindowSec": 300,

+ "circuitBreakers": { "bundle.maxTotalGas": 5\_000\_000 }

Sync script delta:

\*\*\* a/scripts/dev/sync-policies.ts

--- b/scripts/dev/sync-policies.ts

@@

// read on-chain guards.maxSteps, maxSlippageBps, twapWindow and write into policies.json

## **7) Deployment — wire WETH & Timelock; configure breakers**

\*\*\* a/script/Deploy.s.sol

--- b/script/Deploy.s.sol

@@

- new UniswapV3Adapter(router, quoter);

+ new UniswapV3Adapter(router, quoter, IWETH(cfg.weth));

\*\*\* a/script/Configure.s.sol

--- b/script/Configure.s.sol

@@

function run(address guardsAddr, address executorAddr) external {

PolicyGuards g = PolicyGuards(guardsAddr);

g.setMaxSteps(8);

g.setAllowedFee(500, true);

g.setAllowedFee(3000, true);

g.setAllowedFee(10000, true);

+ g.setMaxSlippageBps(100);

+ g.setTwapWindow(300);

+ (bool ok, ) = executorAddr.call(abi.encodeWithSignature("setMaxTotalGas(uint256)", 5\_000\_000));

+ require(ok, "setMaxTotalGas");

}

Timelock handoff:

\*\*\* a/script/OwnershipTransfer.s.sol

--- b/script/OwnershipTransfer.s.sol

@@

- (bool ok1,) = guardsAddr.call(abi.encodeWithSignature("transferOwnership(address)", newOwner));

- (bool ok2,) = registryAddr.call(abi.encodeWithSignature("transferOwnership(address)", newOwner));

+ (bool ok1,) = guardsAddr.call(abi.encodeWithSignature("transferOwnership(address)", timelock));

+ (bool ok2,) = registryAddr.call(abi.encodeWithSignature("transferOwnership(address)", timelock));

## **8) Tests — unit, integration, fork**

### **8.1 Foundry (Solidity)**

\*\*\* /dev/null

--- b/contracts/test/venues/UniswapV3Adapter.native.t.sol

@@

// Tests:

// - exactInputSingle with tokenIn == Native.NATIVE wraps WETH

// - multi-hop with first/last native sentinel

// - revert on msg.value mismatch

// - delivers ETH to caller when recipient == Native.NATIVE and last token is WETH

\*\*\* /dev/null

--- b/contracts/test/execution/Executor.breakers.t.sol

@@

// Tests:

// - paused blocks execution

// - gas breaker triggers when accumulated gas exceeds limit

### **8.2 Vitest (TS)**

\*\*\* /dev/null

--- b/packages/core-exec/test/sim.simulator.spec.ts

@@

// Tests:

// - simulate returns perStep outputs

// - respect policies.maxSlippageBps

// - twapOk true for stubbed 0

\*\*\* /dev/null

--- b/packages/cli/test/route.simulate.exec.spec.ts

@@

// CLI simulate -> execute-with-sim uses result to set minTotalOut

## **9) Observability — alerts & Prometheus counters**

Add counters and simple alert hooks.

\*\*\* a/packages/core-exec/src/telemetry/metrics.ts

--- b/packages/core-exec/src/telemetry/metrics.ts

@@

export function inc(name:string, labels:Record<string,string> = {}) {

const key = name + JSON.stringify(labels);

counters[key] = (counters[key] ?? 0) + 1;

}

+export function observe(name:string, value:number, labels:Record<string,string> = {}) {

+ const key = name + JSON.stringify(labels);

+ counters[key] = (counters[key] ?? 0) + value; // simple sum; agents can extend to hist

+}

Hook executor event (indexer) to bump counters:

\*\*\* a/packages/core-exec/src/telemetry/indexer.ts

--- b/packages/core-exec/src/telemetry/indexer.ts

@@

watchContractEvent(client, { address: addresses, abi, onLogs: (logs) => {

for (const l of logs) { appendFileSync(OUT, JSON.stringify(l) + "\n"); }

+ for (const l of logs) {

+ if (l.eventName === "BundleExecuted") {

+ inc("bundles\_executed\_total", { network: process.env.NET ?? "local" });

+ observe("bundle\_gas\_sum", Number(l.args.totalGas), {});

+ }

+ }

}});

Docs:

\*\*\* a/docs/dev/telemetry.md

--- b/docs/dev/telemetry.md

@@

- New counters in M3:

- bundles\_executed\_total

- bundle\_gas\_sum

- Example Prometheus scrape: `http://localhost:9464/metrics`

## **10) CLI/Docs — native token examples & timelock**

Readme/runbook snippets:

\*\*\* a/README.md

--- b/README.md

@@

+### Native ETH swaps (M3)

+Build a path with native on the left or right using the `0x000...000` sentinel and the adapter will wrap/unwrap WETH at the edges:

+```

+pnpm cli route build-multihop \

+ --adapter $UNIV3\_ADAPTER \

+ --hops '["0x0000000000000000000000000000000000000000",500,"0xWETH",3000,"0xUSDC"]' \

+ --amount-in 10000000000000000 --recipient 0xYOU --deadline $DL

+```

Ops runbook:

\*\*\* /dev/null

--- b/docs/runbooks/m3-ops.md

@@

+# M3 Ops

+- Deploy timelock, set multisig as proposer/executor.

+- Transfer ownership of Guards/Registry to timelock.

+- Configure breakers: `setMaxTotalGas`.

+- Enable/disable pause via timelock schedule.

+- Rollback: pause executor, disable route in registry.

## **11) CI pipeline bumps**

\*\*\* a/.github/workflows/ci.yml

--- b/.github/workflows/ci.yml

@@

contracts-integration:

steps:

- name: Foundry (integration – native)

run: forge test -vv --match-contract UniswapV3Adapter.native

ts-fork-tests:

steps:

- name: Vitest simulate & CLI

run: pnpm -w vitest run --dir packages/core-exec --dir packages/cli

## **12) Acceptance checklist (M3)**

Create acceptance/m3.md with objective gates:

1. Native input (ETH→WETH) wraps and executes; output paid as ETH when requested.
2. Revert on msg.value ≠ amountIn.
3. Circuit breaker halts bundle when totalGas exceeds configured threshold.
4. paused=true blocks execution and emits event.
5. Simulator produces expectedOut; CLI uses it to set minTotalOut and execution succeeds on fork within slippage budget.
6. Timelock owns PolicyGuards and RouteRegistry; multisig can propose/execute config changes.
7. Metrics counters increment on real executions.
8. Docs updated and examples runnable.

## **13) Versioning**

\*\*\* a/packages/adapters-evm/package.json

--- b/packages/adapters-evm/package.json

@@

- "version": "0.2.0-m2",

+ "version": "0.3.0-m3",

\*\*\* a/packages/core-exec/package.json

--- b/packages/core-exec/package.json

@@

- "version": "0.2.0-m2",

+ "version": "0.3.0-m3",

\*\*\* a/packages/cli/package.json

--- b/packages/cli/package.json

@@

- "version": "0.2.0-m2",

+ "version": "0.3.0-m3",

## **14) Known limitations (explicit, to unblock ship)**

* **TWAP** function is stubbed to 0 drift; agents should wire pool oracle (Uniswap V3 observations) in follow-up.
* **quoteSingle** decoder in TS is minimal; agents to complete ABI field extraction.
* **Permit/Permit2** not included in M3 scope; add in M4 if needed.

## **15) Done-ness (merge gates)**

* Solidity compiles; unit+integration (native & breakers) passing.
* TS simulator & CLI simulate/execute e2e pass on Sepolia fork.
* Timelock deployed; ownership transferred; functional check with a queued setMaxSteps change.
* Circuit breaker and pause verified.
* Docs & runbooks updated.
* Version bumps & release notes m3-release-notes.md added.