

UNIVERSITY OF
CAMBRIDGE
Department of Computer
Science and Technology

Trading Under Delay: Smart-Contract Engineering for Latency-Aware AMMs

Rishi R. Kataria

St. Edmund's College

May 2026

Submitted in partial fulfillment of the requirements for the
Master of Philosophy in Advanced Computer Science

Total page count: 71

Main chapters (excluding front-matter, references and appendix): 51 pages (pp 8–58)

Main chapters word count: 14,998

Methodology used to generate that word count:

```
texcount -inc -total -sum -1 main.tex
```

Declaration

I, Rishi R. Kataria of St. Edmund's College, being a candidate for the Master of Philosophy in Advanced Computer Science, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose. In preparation of this report, I adhered to the Department of Computer Science and Technology AI Policy.

Signed: Rishi R. Kataria

Date: 9 June 2026

Abstract

Decentralised finance protocols on the Ethereum Virtual Machine assume the gap between observing remote state and acting on it is negligible. That assumption fails when execution domains are separated by minutes of propagation delay rather than milliseconds of network latency, the regime imposed by interplanetary links, scheduled satellite contact windows, and intermittently connected terrestrial networks. Earth and Mars are the main case. Intermittent and scheduled networks motivate the setting, but the mechanism and evaluation focus on a fixed one-way delay under assumed contact availability.

This dissertation studies that regime for trading on automated market makers, taking Earth and Mars as the sharpest case. A constant-product AMM reached across the delay is shown to fail in three distinct ways, a stale price, a late execution, and capital stranded in flight, and standard parameter changes, such as tighter slippage bounds, higher priority fees, or faster local blocks, do not remove the propagation interval between observing remote reserves and executing against them. The answer it proposes is the lane option, an original mechanism that gives one trader a private, time-bounded slice of the AMM, sealed so the delay cannot spoil the price they act on, priced by a single premium and closed by a merge and, under the model, returns a non-negative constant-product invariant. A portable form bridges the slice to the trader's own chain so swaps settle locally rather than a delay away.

Both are realised as EVM smart contracts. With no autonomous scheduler and a hard cap on per-transaction work, the lane's merge rides on ordinary trades and a bonded cleanup task, and the portable form carries the slice to a custodian on the trader's chain through a trusted relay.

Both are evaluated against the three failures on StateLab, a purpose-built two-chain framework under an Earth–Mars-scale delay. The AMM-side lane removes the price lag, carrying none of the divergence that costs the baseline from about two percent on a calm pair to past sixty percent on a volatile one. The portable lane also clears the execution lag, a reactive trader filling hundreds of trades to a remote venue's one past half an hour, and turns a per-swap capital cost into a single round trip. Neither is free, a thinner slice slipping more, near sixteen percent on a five-percent slice, so a lane pays where the lag it clears beats the drag and the premium it costs.

Acknowledgements

I would like to express my sincere gratitude to Professor Frank Stajano and Dr Ferdinando Samaria for their supervision and for putting up with me over the course of this project. Trading on a market that sits light-minutes away is about as blue-sky as a project gets, and I am glad they let me chase it. Thank you both for the patience and the steady push.

I am grateful to the University of Cambridge for the best year of my life, and for the people and the place that make it what it is.

To my friends here, thank you for the formals, the company, and the distractions I needed more than I let on. You turned a hard year into a good one.

And to my family, thank you for everything, for the love and the belief that never wavered. None of this happens without you.

Contents

1 Introduction **8**

1.1 Contributions 9

2 Background **11**

2.1 EVM chains and smart contracts 11

2.2 Constant-product AMMs 12

2.3 Cross-chain transport 13

2.4 Interplanetary networking 14

2.5 Related work 15

3 Architecture and problem **16**

3.1 Delay model and notation 16

3.2 Architecture 17

3.2.1 One shared chain 17

3.2.2 Two local chains and messages 18

3.3 Cross-planetary swaps and the three lags 19

3.3.1 Price lag 20

3.3.2 Execution lag 21

3.3.3 Capital lag 21

3.4 Design requirements 22

4 Lane options **23**

4.1 The lane option mechanism 23

4.1.1 The split 24

4.1.2 Trading the lane 25

4.1.3 The merge 25

4.2 Pricing lane options 28

4.3 Where the lanes live 29

4.3.1 The AMM-side lane 29

4.3.2 The portable lane 29

4.4 Lane options under the three lags 31

4.4.1 The AMM-side lane 31

4.4.2 The portable lane . . . 32
4.5 Advantages of lane options . . . 33
4.5.1 Wider uses . . . 33

5 Engineering smart contracts under delay **34**
5.1 A simple AMM contract . . . 34
5.2 The lane options contract . . . 34
5.2.1 Constraint 1, the gas limit . . . 35
5.2.2 Constraint 2, no scheduler . . . 36
5.3 The portable lane options contract . . . 40
5.3.1 The bridge . . . 40
5.3.2 Creating the custodian . . . 41
5.3.3 Moving the slice across . . . 42
5.3.4 Opening and running a lane . . . 42
5.3.5 Returning the lane at expiry . . . 42
5.4 Contract analysis . . . 44
5.4.1 Security analysis . . . 44

6 Evaluation **46**
6.1 StateLab . . . 46
6.1.1 Existing frameworks . . . 46
6.1.2 Implementation . . . 47
6.1.3 Open Source Release . . . 48
6.2 Setup . . . 49
6.2.1 Testbed and trader . . . 49
6.2.2 Price process . . . 49
6.2.3 Monte Carlo . . . 50
6.3 Experiments . . . 50
6.3.1 Price lag . . . 50
6.3.2 Execution lag . . . 51
6.3.3 Capital in transit . . . 52
6.3.4 Split drag . . . 53
6.3.5 Optimal lane size . . . 55
6.4 Discussion . . . 55

7 Conclusion **57**

Bibliography **59**

A Worked examples **64**
A.1 Price lag . . . 64
A.2 Execution lag . . . 65

A.3 Capital lag 66

B Lane merge proofs **68**

B.1 Deriving the merged invariant 68

B.2 The surplus is non-negative 69

C Self-induced divergence **70**

Chapter 1

Introduction

Financial systems implicitly assume the time between observing a remote state and acting on it is short compared to how fast that state changes. Decentralised finance (DeFi), where markets run as public programs on a blockchain rather than inside a bank or an exchange, makes the same assumption [2, 16], and on an ordinary blockchain it is sound. Blocks arrive every few seconds and a trade settles within a handful of them, so a trader reads a market, sends an order, and executes before the quote goes stale, ordinary safeguards holding the small remaining slippage in check [6].

The market studied here is an automated market maker (AMM), a smart contract that holds a pool of two assets and prices each trade off how much of each it holds [11]. Each trade changes those holdings and so the next trade's price, which makes an AMM a sharp test of delay [13, 14].

The assumption breaks once a message between trader and market takes minutes instead of milliseconds. Earth and Mars are the running example and the sharpest case, but the same shape appears wherever local execution continues while outside news arrives late, on satellite, intermittent, or congested networks [39, 40]. The deciding factor is the ratio between the delay and the speed at which prices move, not the distance. A trader on Earth and an AMM on Mars are four to twenty-four minutes of one-way light-time apart [45].

That delay costs the remote trader in three ways. Mars traders can still submit swaps to the same AMM and move its reserves, so the Earth order may reach a market whose price is no longer the one the trader acted on. This is price lag. A Mars-side trader who sees the same opportunity is included in local block time while the Earth order arrives minutes later, so the nearer trader acts first. This is execution lag. And the funds behind the Earth order stay committed the whole time it is in flight. This is capital lag. For one trader each lag is an opportunity cost, and for the system capital lag becomes a liquidity cost.

The safeguards that contain delay on Earth do not reach that far. A slippage bound, the worst price a trader will accept, the order reverting if the fill would be worse, can stop a stale order filling at a bad price [11] but cannot make it arrive earlier, and a reverted order has still spent the time in transit for a round trip to the market. A higher gas fee can shorten the local inclusion wait by paying block builders for faster inclusion [6], and changing local ordering rules can reduce local races [27], but neither can remove the minutes spent in propagation. The problem is not that a price can move, but that observing it, acting on it, and freeing the capital behind it are no longer close in time.

This dissertation answers that with a market designed for the delay rather than in spite of it. It first fixes the architecture, joining two local chains by delayed messages rather than using one shared chain that would leave a side permanently late or slow every local trade to the round trip. Within that architecture sit *lane options*, an original AMM mechanism in which a trader pays a flat premium for the exclusive right to trade a private slice of an AMM for a fixed time. Sealed from public trading for the lease, the slice shields the holder from the price changes the delay would otherwise let through. A portable form carries it to the holder's own chain, so repeated swaps settle locally rather than a delay away. The dissertation then implements both forms on the EVM, within its gas limit and without any scheduler, and evaluates them against the three lags as market conditions vary.

1.1 Contributions

This dissertation makes five contributions.

1. **A decomposition of the trading-under-delay problem.** A two-chain architecture is shown to work under delay where a single shared chain does not. A delay long enough to matter breaks a constant-product AMM in three ways, price lag, execution lag, and capital lag (Chapter 3).

2. **Lane options, an original AMM mechanism.** A trader pays a premium for the exclusive right to trade a private slice of an AMM for a fixed time. The slice is split off from the public pool and merged back at the end, under the model returning liquidity no smaller than it split off. A portable form carries it to the trader's own chain so swaps settle locally. The mechanism, its pricing, and its behaviour against the three lags are developed in full (Chapter 4).

3. **A smart-contract implementation of lane options for the EVM.** Both forms of the lane option are engineered as Solidity smart contracts within the constraints of the EVM (no built-in scheduler and a hard cap on the work one transaction can do). It is paired with a critical security analysis of the attack surface (Chapter 5).

4. **StateLab, a general-purpose framework for testing real smart contracts under controlled time.** Evaluating the contracts needs two chains running real Solidity contracts under a minutes-long delay. Existing tools give a local chain to run real contracts on, but lack a multichain accelerated clock [8, 10]. StateLab was built to fill that gap. It holds one clock outside the chains, advances it from event to event so block time and relay delay scale in proportion, and replays every run deterministically from a seed. It runs any compiled smart contract across any number of chains and is released under the MIT licence (Chapter 6).

5. **An evaluation of lane options across market regimes.** Both lane-option forms and a constant-product AMM as baseline are run as deployed bytecode on StateLab under delay, with metrics defined for each lag and its cost. The results, swept across volatility regimes and key parameters, show which design clears each lag, what premium the lag supports, what split fraction minimises the holder's cost, and where each design wins over the others (Chapter 6).

Chapter 2

Background

2.1 EVM chains and smart contracts

An EVM (Ethereum Virtual Machine) chain is a replicated state machine whose state holds accounts, balances, and the code and storage of smart contracts [1, 3]. Nodes called *validators* agree on an ordered sequence of transactions and execute each deterministically to the same next state [2, 6]. Ethereum is the original such chain, and EVM-compatible networks reuse the model with their own validators [6].

Transactions are grouped into *blocks*, each pointing to its parent, so the chain is an append-only sequence whose order fixes when transactions take effect (Figure 2.1) [2]. The *block time* δ_{block} is the nominal interval between blocks, about twelve seconds on Ethereum mainnet [6]. A transaction's cost is measured in *gas*, paid in the chain's *native token*, and each block carries a *block gas limit* capping the gas its transactions may use [6].

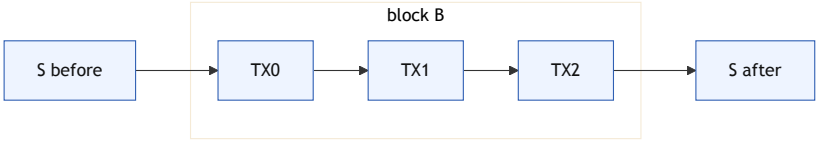


Figure 2.1: An append-only chain of blocks, each extending the last in a single agreed order [2].

A *smart contract* is a program stored at an address on the chain, with its own persistent storage, that runs when a transaction calls one of its functions [6]. A call either *commits* its writes or *reverts*, undoing every change and leaving the state as it was [2, 30], and validators re-run the same code against the same starting state for the same transition [2].

A smart contract is public and its caller untrusted, so anyone may call any function, and a contract restricting an action must check the caller itself [6].

Two properties of the EVM then bound what a smart contract can do.

Constraint 1: the transaction gas limit. Every state-changing call runs inside one transaction, capped by the block gas limit, so an operation whose worst-case cost grows without bound in a contract's state becomes impossible once that state is large enough for no caller to fit the call in a block. The exception is a *read*, a call that only inspects state and returns a value. A read call never enters a block, so the limit does not apply [6].

Constraint 2: no scheduler. The chain is passive. Its state changes only when a transaction is applied, so between transactions no smart contract runs and no balance moves, however much time passes [2, 6]. A contract cannot wake itself or schedule future work, so every change it makes is driven by an external transaction and anything due later waits for the next that calls in.

A *token* is itself a smart contract, used to represent an asset on chain. It keeps a ledger of how much each address holds, and a transfer moves units from one balance to another. Where its code allows, it can also *mint* new units at an address or *burn* units [17].

Smart contracts are the basis for decentralised finance (DeFi), where financial applications run as public on-chain programs [16] that hold token reserves, quote prices from them, and move balances by their own code [13].

2.2 Constant-product AMMs

An automated market maker (AMM) is a smart contract that holds two tokens, together a *liquidity pool*, and lets anyone swap one for the other on demand [14]. The amounts it holds are its *reserves* (X, Y). A *trader* pays one token in and takes the other out, while a *liquidity provider* (LP) deposits both tokens in the current ratio to supply the inventory each swap draws on, collecting the fee each swap pays into the reserves [14]. An LP's risk is *impermanent loss* [20]. When the pool's price drifts from the deposit ratio, the pooled stake is worth less than holding the two tokens, a gap the rule does not compensate, one that reverses only if the price returns. Whether fees leave an LP ahead depends on flow, volatility, and the fee level [19].

The *constant-product* AMM used by Uniswap keeps the product of its reserves fixed in the fee-free model [11, 13],

$$X \cdot Y = k, \tag{2.1}$$

so valid reserve states lie on a hyperbola and every fee-free swap moves the pool along it (Figure 2.2). A trader sending Δx in raises X to $X + \Delta x$, and the contract resets Y to keep the product fixed, paying out its drop,

$$\Delta y = Y - \frac{XY}{X + \Delta x} = \frac{Y \Delta x}{X + \Delta x}. \tag{2.2}$$

The reverse trade follows the same rule with the reserves exchanged [14].

Definition 2.1 (Marginal price)**.** *The* marginal price *of X in units of Y is the rate for an infinitesimal $X \to Y$ swap at the current reserves, the price a reader sees on inspecting them,*

$$P = \frac{Y}{X}. \tag{2.3}$$

Definition 2.2 (Execution price)**.** *The* execution price *of a swap of size Δx is the average output per unit input,*

$$P_{\text{exec}}(\Delta x) = \frac{\Delta y}{\Delta x} = \frac{Y}{X + \Delta x}. \tag{2.4}$$

It is below P for positive Δx and falls as Δx grows.

Definition 2.3 (Slippage)**.** Slippage *is the gap between the marginal and execution prices of a finite trade [14],*

$$P - P_{\text{exec}}(\Delta x) = \frac{Y}{X} - \frac{Y}{X + \Delta x} = \frac{Y\,\Delta x}{X(X + \Delta x)} = \frac{\Delta y}{X}, \tag{2.5}$$

which grows in Δx and vanishes as $\Delta x \to 0$, and shrinks as the reserves grow, so a deeper pool slips less on the same trade.

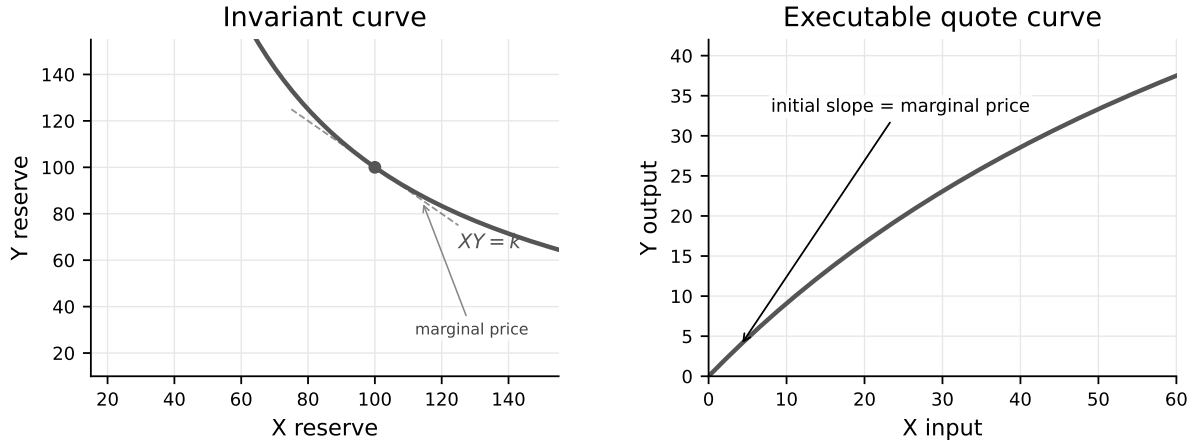


Figure 2.2: The constant-product rule in two views, the invariant curve $XY = k$ and the quote a single trade receives along it [13].

Both prices depend only on the reserves, which by the passive state-machine property of §2.1 move only when a transaction changes the pool. A reader therefore holds a quote good only until the next transaction touches it, so once observation and execution are separated by a delay an intervening transaction can move the pool and the price the trader saw differs from the price the trade settles at [13, 18].

2.3 Cross-chain transport

A cross-chain action connects two chains that share no block sequence. A smart contract sees only its own chain's state, so an action touching both sides cannot be one transaction

and splits into a source-chain transaction that emits a message and a later destination-chain transaction [34]. The message is only data, changing nothing until that destination transaction carries its effect out (Figure 2.3).

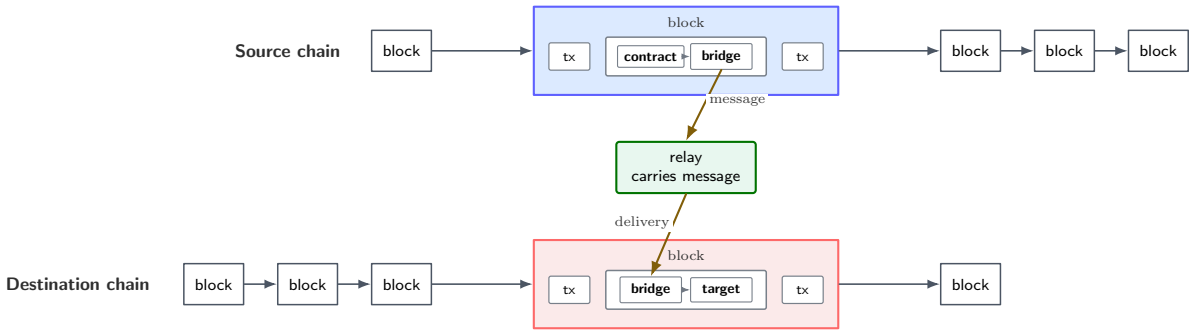


Figure 2.3: A cross-chain message, carried by the relay from one chain to the other across the delay and applied once on arrival.

The machinery that carries this out is a *bridge*, a pair of smart contracts, one on each chain, with a *relay* between them, an off-chain party that sees what the source-side contract emits and calls the destination-side contract, which checks the message and calls the target [33, 34]. A token crosses the same way, since its balances live in one contract and cannot move across directly. A simplified bridge crosses it by *burn-and-mint* [34], burning units on the source chain and minting matching units on the destination.

Bridges differ in how the destination convinces itself a source message is genuine. Bridges range from a trusted relay, which delegates trust to a single off-chain party, to light-client and proof-carrying bridges that check the source chain's consensus data or a cryptographic proof [34, 37], trading more on-chain work for less trust while keeping the shape of a cross-chain action.

2.4 Interplanetary networking

Two limits govern interplanetary communication, but only one is fundamental [39, 40]. The first is *propagation delay*, the time a signal takes to travel. Set by the speed of light, it cannot be engineered away, so an Earth–Mars message takes minutes even on a clear, direct path [40, 45]. The second is *link availability*, whether a usable path exists at a given moment. Mars missions reach Earth through scheduled contact windows on shared infrastructure such as the Deep Space Network, with commanding paused for weeks during solar conjunction, when the planet passes behind the Sun [41, 46]. Availability, by contrast, is an engineering problem. Fall introduced delay-tolerant networking [39], which stores and forwards traffic across intermittent contact rather than assuming an end-to-end path, an approach Burleigh et al. carry to the interplanetary setting [40], and Jesick et al. propose dedicated relay spacecraft in Mars-Trojan orbits off the direct Earth–Mars line that keep a path open even when the two planets are poorly aligned [42].

2.5 Related work

Delay already matters in ordinary electronic trading, where market data reaches participants at different times, so a trader can act on a state another has already seen change. The market microstructure literature studies this as adverse selection [26] and latency arbitrage [24], and Budish, Cramton, and Shim show how continuous-time markets turn small speed advantages into races and propose frequent batch auctions [25].

Blockchains add a timing problem of their own. A sent transaction waits in a public area, the *mempool*, visible to everyone until a block producer fixes its place in a block [18]. Whoever assembles the next block chooses which pending transactions to include, in what order, and whether to insert its own. The value a producer or searcher extracts this way, beyond block reward and fees, is *maximal extractable value* (MEV) [18].

The clearest MEV against an AMM is the *sandwich attack*, where a searcher swaps just before and after a victim, profiting from the price the victim's trade moves [28]. A constant-product pool quotes from its reserves, so when an external price moves first, arbitrageurs trade against the stale pool until its ratio catches up, fees deciding which LPs come out ahead [13, 23]. Stale pricing and ordering advantage already arise in DeFi, where users respond with slippage bounds and priority fees [11].

Trading across a minutes-long propagation delay is not a mempool wait or an MEV race [18]. In a mempool race every participant reads the same live chain state and the only contest is the order in which transactions are included. Under a propagation delay the read is stale and the write delayed. The trader acts on a price already a delay old, and the order then takes a further delay to reach the market [13, 34]. Delay-tolerant payment networks come closest, but a payment's amount is set by the sender before it leaves, so delay cannot change what arrives [43, 44]. An AMM's output is computed by the pool's reserves at the moment of the swap, and those reserves move during the delay [13].

This dissertation occupies the gap between these lines of work. Market-latency work studies speed within one venue [25], blockchain-ordering work studies contention within one chain [18], AMM work studies stale pricing within one pool [13], and delay-tolerant work studies message delivery without the state-dependent execution an AMM trade needs [43]. None covers a trade whose output is fixed only at a destination AMM that has already moved during a minutes-long delay.

Chapter 3

Architecture and problem

The problem needs a precise setting, a clear definition, and a suitable architecture to trade within. A single shared chain across the delay either leaves one side permanently late or slows every local action to the round trip, so the dissertation adopts two local chains joined by delayed messages. Within that architecture, a cross-planetary swap fails in three ways, and those failures become the requirements a fix must meet.

3.1 Delay model and notation

The setting is two locations far enough apart that a noticeable time delay separates any message between them. Earth and Mars are the running example, and nothing in the argument requires these exact planets.

Definition 3.1 (Delivery delay)**.** *The* delivery delay τ *is the one-way time a message takes to travel from one chain to the other.*

For Earth and Mars, τ is the one-way light time, smallest at opposition and largest near solar conjunction, when Mars is behind the Sun,

$$\tau \in [\,4\,\text{min},\ 24\,\text{min}\,] \tag{3.1}$$

over the repeating Earth–Mars cycle [40, 45], as Figure 3.1 shows. Local block times here are seconds, so τ dominates local execution time. The dissertation assumes continuous contact, which §2.4 argues is an infrastructure choice rather than a physical limit, so τ reduces to light time alone. Even the blackout roughly every twenty-six months, when Mars passes behind the Sun, is an engineering constraint rather than a physical one, with relay spacecraft in Mars-Trojan orbits proposed to maintain a continuous path [42].

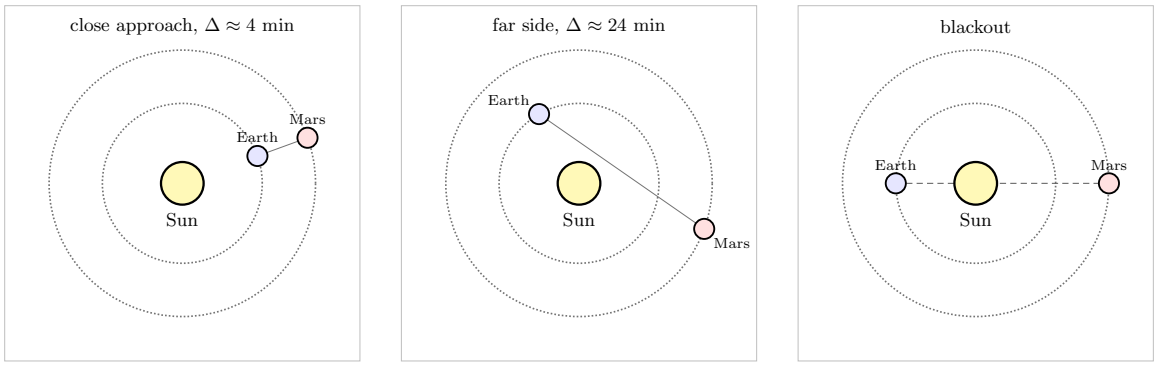


Figure 3.1: Earth–Mars signal geometry, the one-way light time τ ranging with the planets' separation over the synodic cycle.

3.2 Architecture

Two architectures are possible. The two sides can share a single EVM chain, or each can run its own EVM chain with a cross-chain protocol carrying messages between them.

3.2.1 One shared chain

A single shared EVM chain would remove the cross-chain bridge and keep single-chain atomicity. It is unavailable once the delivery delay τ exceeds a chain's block time by orders of magnitude, because security degrades when blocks are produced faster than the network can spread them [5]. Two cases cover the choices.

Case 1, extending an existing Earth chain to Mars. The starting point is an EVM chain that already exists on Earth, with Ethereum as the example. Its validators sit on Earth, its block time is about 12 seconds [6]. Mars participants can join either as users of the Earth-hosted chain or as a validator minority. Both fail.

As users. A Mars-submitted transaction crosses to Earth, waits for the next Earth block, then waits for confirmation to return, a round trip plus one block, while an Earth-side user on the same chain waits only one block. With a one-way delay of 4 to 24 minutes (§3.1) against Ethereum's 12-second block, the round trip dominates, even if a Mars user wants to transact with another Mars user.

As validators. A Mars-proposed block takes minutes to reach the Earth majority, which in that time mines a new block and moves the chain forward. By the fork-choice rule, the shared rule for picking that branch, the Mars block becomes an orphan, a valid block discarded for losing the race (Figure 3.2), dropping any transactions that appear only in it [4, 5].

The argument is symmetric in the host side. With the existing chain on Mars instead, the same gap applies to Earth participants, so the failure is between host and remote, not between planets.

Case 2, a new chain with validators on both sides. The repair gives this new chain

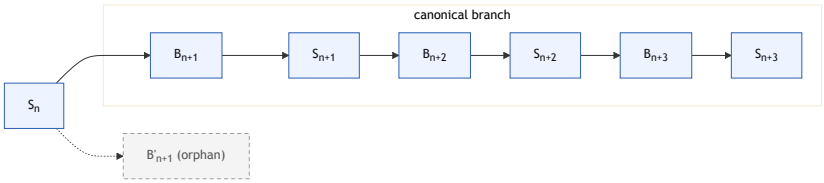


Figure 3.2: An orphan block, a remotely proposed block discarded by the fork-choice rule when the majority chain advances past it during the delay.

a block interval long enough that a candidate block reaches the other side before the next is due, so no side is remote and no proposed block is orphaned [5]. That interval is at least twice the one-way delay, 8 to 48 minutes between consecutive Earth–Mars blocks. The bound defeats both failure modes of Case 1 but carries two consequences.

Local transactions wait the full block time. A local Earth-to-Earth transfer or Mars-to-Mars swap waits the full 8 to 48 minutes for inclusion, even though none of its effect leaves the chain.

External events cannot reach the chain promptly. If an external market price moves or an oracle reading changes, the on-chain reflection waits for the next block, so AMM reserves and oracle values are stale by up to a round trip.

Case 2 removes the asymmetry only by making every participant and state update pay the same long wait, ruling out one chain across both planets.

3.2.2 Two local chains and messages

The two-chain architecture gives each side its own EVM chain, with its own validators, block time, mempool, fee market, and participants. A transaction is local when it reads and writes only its own chain's storage, and local transactions confirm under that chain's ordinary block time, not under the interplanetary delivery delay.

Definition 3.2 (Local chains)**.** *Represent each local EVM chain by its ordered block sequence. The Earth and Mars chains are*

$$C_E \;=\; (B_E^0, B_E^1, B_E^2, \ldots), \qquad C_M \;=\; (B_M^0, B_M^1, B_M^2, \ldots). \tag{3.2}$$

Here B_E^n is the nth block on C_E and B_M^n the nth on C_M. The ordering in each is local, with no shared global block sequence.

Definition 3.3 (Local block times)**.** *The local block time is the expected time from one block on a chain to the next block on the same chain:*

$$\delta_E \;=\; \textit{expected time from } B_E^n \textit{ to } B_E^{n+1}, \tag{3.3}$$

$$\delta_M \;=\; \textit{expected time from } B_M^n \textit{ to } B_M^{n+1}. \tag{3.4}$$

These are local consensus parameters. They may be equal but need not be. Neither includes the interplanetary delivery delay τ.

The two chains share no state, so one can affect the other only through the relay of §2.3, the off-chain party carrying each message across one delivery delay τ.

This dissertation uses a single relay and takes it as trusted, since the trading problem depends only on the delivery delay and the absence of shared state, not on the carrier. The cost of that trust, and the threats a dishonest or broken relay poses, are taken up in the security analysis of §5.4 [34, 35].

This architecture keeps per-side execution but loses cross-chain atomicity (§2.3), so a swap spanning both sides is at least two transactions separated by the delivery delay, with the destination chain continuing to advance between them (Figure 3.3).

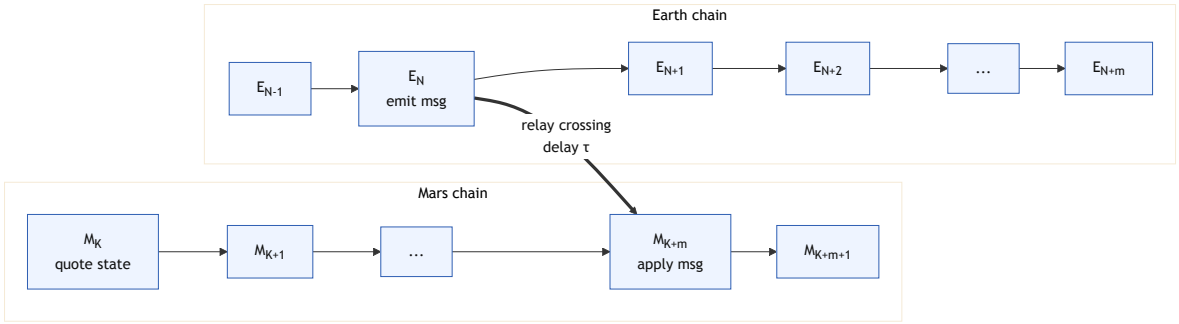


Figure 3.3: A cross-planetary swap from Earth to Mars, two transactions separated by the delay while the destination chain advances between them.

3.3 Cross-planetary swaps and the three lags

A cross-planetary swap is the one trade the two-chain world cannot make in a single step (§3.2.2).

Definition 3.4 (Cross-planetary swap)**.** *A* cross-planetary swap *is a swap against an AMM pool in which the trader and the pool sit on opposite chains. For an Earth-to-Mars swap, the source transaction commits on Earth at block B_E^N, and the destination write can be submitted on Mars only after one delivery delay τ. During that crossing, the Mars chain keeps appending blocks with block time δ_M, including any other swaps against the same pool that move its reserves (X, Y). $N_{\text{Mars}}(\tau) = \lfloor \tau/\delta_M \rfloor$ is a lower bound on the Mars blocks that pass during the crossing, and the swap executes against a later Mars state than the trader observed. The Mars-to-Earth case is symmetric, with $N_{\text{Earth}}(\tau) = \lfloor \tau/\delta_E \rfloor$.*

It fails in three ways. The trader prices against stale state, the action lands too late, and the capital sits idle for the round trip. Appendix A gives worked examples.

3.3.1 Price lag

The trader reads the destination reserves at quote time, computes an expected output, and submits the cross-planetary swap. By the time it arrives, other trades have moved the reserves, so the contract executes against reserves the trader never priced and the delivered output differs from the quote.

Figure 3.4 shows the timing. The delay count depends on how the trader obtained that state. If the trader already holds a fresh destination-chain view, the executed state is τ stale. If the trader first receives a remote observation and then acts, it is 2τ old.

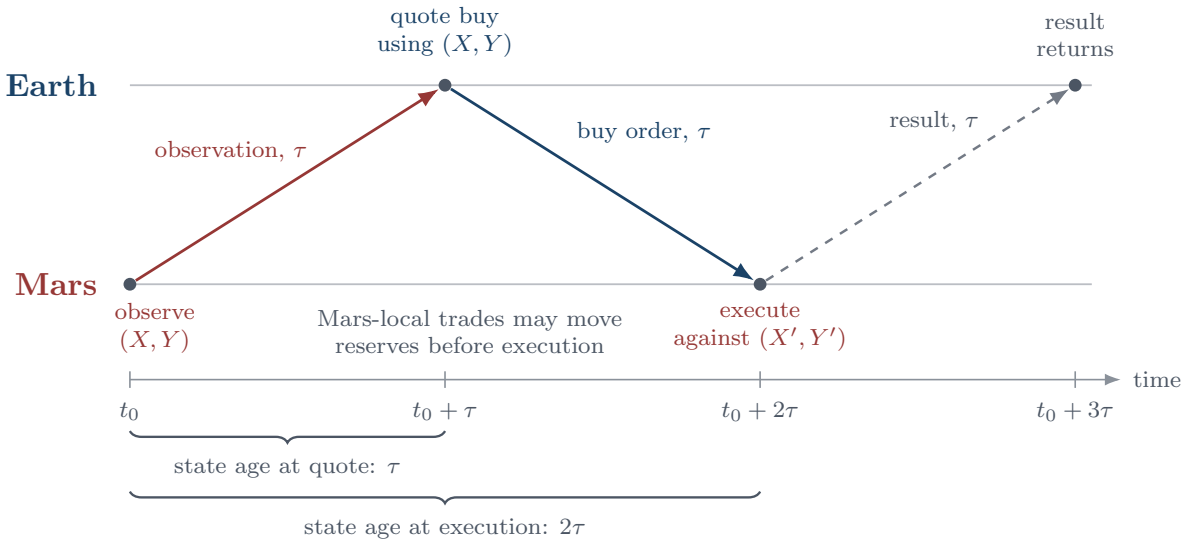


Figure 3.4: Price-lag timeline for an Earth-to-Mars buy, the reserves moving during the delay so the order executes against a state the trader never priced.

Proposition 3.1 (Stale quote)**.** *Let a destination constant-product pool have reserves (X, Y) when the trader forms the quote and reserves (X', Y') when the swap executes. A trader who submits a cross-planetary swap of input size Δx expects output*

$$\Delta y \;=\; \frac{Y\,\Delta x}{X + \Delta x}. \tag{3.5}$$

The contract instead pays the delivered output

$$\Delta y' \;=\; \frac{Y'\,\Delta x}{X' + \Delta x}. \tag{3.6}$$

The price lag *is the gap between the two,*

$$L_{\text{price}} \;=\; \Delta y - \Delta y', \tag{3.7}$$

which vanishes only when

$$\frac{Y'}{X' + \Delta x} \;=\; \frac{Y}{X + \Delta x}, \tag{3.8}$$

and is otherwise nonzero whenever an intervening trade moves the pool along the constant-

product curve, changing the price ratio.

With a prior remote observation (the observe-then-act path of Figure 3.4), the state (X, Y) is itself τ old when the order is formed, so the gap from observation to execution is 2τ, and the proposition applies with (X, Y) read at observation time.

3.3.2 Execution lag

Execution lag is about who reaches the market first. Block inclusion is ordered in time, so where acting sooner is better, the trader whose transaction reaches the chain earlier wins. A cross-planetary trader gives up that priority by a fixed amount [18, 29].

Definition 3.5 (Execution lag)**.** *Let a trader decide at time t to act on the chain where the AMM lives, with local block time δ. A trader already on that chain can be included at about $t + \delta$. A remote trader, whose action must cross the delay first, cannot be included before $t + \tau + \delta$. The* execution lag *is the difference,*

$$L_{\text{exec}} = (t + \tau + \delta) - (t + \delta) = \tau, \tag{3.9}$$

the head start a local trader holds over a remote trader deciding at the same instant.

The lag is decisive whenever acting first wins. Suppose a Mars pool becomes mispriced at time t, and a Mars-local trader and an Earth trader both move to correct it. The Mars trader is included after about δ_M, while the Earth order reaches Mars only after $\tau + \delta_M$, by which time the Mars trader has taken the trade and moved the reserves. The Earth trader does not merely trade against a moved price, which would be price lag, but loses the trade outright, and through the τ interval Mars-local trades keep filling against the mispriced reserves [29].

3.3.3 Capital lag

The trader's source-side funds are pledged when the swap is submitted and stay locked until the outcome returns, redeployable on neither chain for the full round trip 2τ. A reverting swap locks the capital for the same duration.

The cost of that idle time is the return the trader would otherwise have earned. With K the value of the locked assets and r the opportunity rate per unit time, the *capital lag* is the cost over a round trip,

$$L_{\text{cap}} = K\, r\, 2\tau = 2Kr\tau. \tag{3.10}$$

The round trip 2τ is the figure carried forward, since a trader redeploys only against an outcome already seen.

3.4 Design requirements

The three lags and the EVM platform constraints (§2.1) fix what a solution must do. Three requirements come from the lags. It must isolate the trader from reserve changes during the delay, so a quote still holds at execution (price lag). It must let trades settle at local block speed rather than a relay away (execution lag). And it must spare capital a relay crossing on every trade (capital lag). Three more come from the platform. It must preserve local execution, so local activity keeps confirming at local speed. Any cross-chain placement must keep its bridge trust assumption explicit and isolated from the AMM's accounting. And it must run as passive, gas-limited EVM contracts.

Chapter 4

Lane options

Lane options, an AMM mechanism and an original contribution of this dissertation, protect a trader's price from the rest of the public pool's order flow. A lane option leases a trader a private, time-bounded slice of a liquidity pool, leaving the rest to serve everyone else. Measured against the three lags of Chapter 3, an AMM-side lane clears price lag, and a portable lane on the holder's chain clears execution lag as well, while shifting the capital cost to a single round trip of the lane's capital. Chapter 5 gives the EVM implementation of both.

4.1 The lane option mechanism

A lane option gives one trader their own private AMM, split off from a public one for a fixed time. The one idea behind it is to seal that slice from everyone else's order flow, so the price the holder acts on cannot be moved by the trades that cause the price lag. The trader who opens it, the *holder*, trades it as often as they wish in either direction, and it stays theirs alone. It is a right, not an obligation, opening at the parent AMM's current price and afterwards moving only under the holder's own trades, insulated from the public order flow that moves the rest of the AMM. The remainder, the *public pool*, trades on as before. Figure 4.1 follows one lane from split to merge, and the sections below take each stage in turn.

Definition 4.1 (Lane option)**.** *A* lane option *is a private, time-bounded portion of an AMM, bought for an upfront premium. For an AMM with reserves* (X, Y)*, a split fraction* $c \in (0, 1)$*, the share of each reserve moved into the lane, and a lease horizon* T*, a* split *divides the reserves into a* public pool *open to all traders and a private* lane *open only to the holder over the window* $[0, T]$*. The holder pays a premium* Π *at the split, and the lane then runs as an independent constant-product AMM, opening with reserves* (cX, cY)*, until* T*, when it merges back into the public pool.*

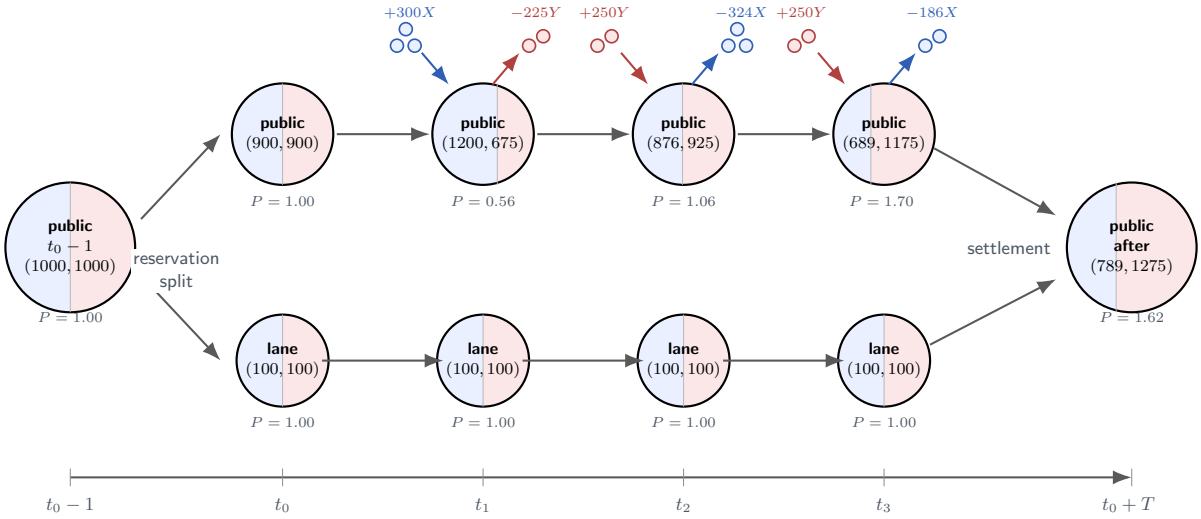


Figure 4.1: One lane over its lease, from the split that opens it, through the holder's private swaps, to the merge that returns it.

4.1.1 The split

The parent AMM has reserves (X, Y), constant-product invariant $k = XY$, and marginal price $P = Y/X$. The split divides these reserves in proportion, the fraction c into the lane and $1 - c$ into the public pool,

$$\text{lane} = (cX,\ cY), \qquad \text{public} = \big((1-c)X,\ (1-c)Y\big). \tag{4.1}$$

Each runs as its own constant-product AMM, with invariant

$$k_{\text{lane}} = (cX)(cY) = c^2 k, \qquad k_{\text{public}} = \big((1-c)X\big)\big((1-c)Y\big) = (1-c)^2 k. \tag{4.2}$$

Taking the same fraction of each reserve leaves both pools at the parent price,

$$\frac{cY}{cX} = \frac{(1-c)Y}{(1-c)X} = P, \tag{4.3}$$

so the lane, the public pool, and the parent all share the one price P, which the split locks in for the holder (Figure 4.2).

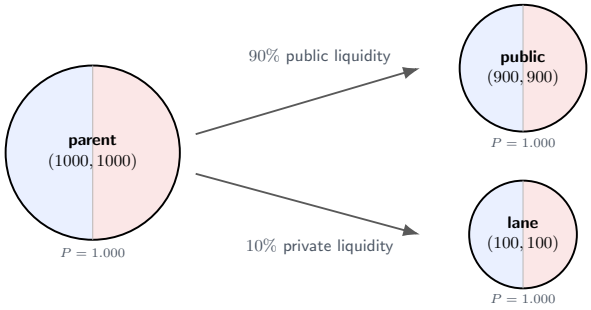


Figure 4.2: The split, carving a fraction c of the pool's reserves into a private lane at the current price and leaving the public pool on a shallower curve.

4.1.2 Trading the lane

During the lease the lane is the holder's alone, priced by the constant-product rule of §2.2. An input Δx against the lane's reserves (X_ℓ, Y_ℓ) returns

$$\Delta y = \frac{Y_\ell \, \Delta x}{X_\ell + \Delta x}, \tag{4.4}$$

where $(X_\ell, Y_\ell) = (cX, cY)$ at the opening trade and evolves with the holder's swaps. Because the lane holds only the fraction c of the reserves, it sits on a shallower curve than the undivided AMM, so the same trade moves its price much further (Figure 4.3). The split conserves the parent's aggregate geometric-mean liquidity $\sqrt{XY}$, the standard measure of a constant-product pool's depth, across the public pool and lane,

$$\sqrt{(1-c)X \, (1-c)Y} + \sqrt{cX \, cY} = (1-c)\sqrt{k} + c\sqrt{k} = \sqrt{k}, \tag{4.5}$$

so the lane and the public pool sit on two shallower curves through the same price P (Figure 4.4). The public pool's tradable liquidity still drops while the lease runs, the fraction c reserved to the lane until merge.

For the same input the lane returns less than the full pool, since it is shallower (Figure 4.4).

Proposition 4.1 (Split drag)**.** *For an input Δx the full pool and the lane, which holds only cX and cY, return*

$$\Delta y_{\text{full}} = \frac{Y \Delta x}{X + \Delta x}, \qquad \Delta y_{\text{lane}} = \frac{cY \Delta x}{cX + \Delta x}, \tag{4.6}$$

the lane the smaller of the two. Each falls below the marginal price by its slippage (§2.2), a fraction of price

$$s_{\text{full}} = \frac{\Delta x}{X + \Delta x}, \qquad s_{\text{lane}} = \frac{\Delta x}{cX + \Delta x}, \tag{4.7}$$

on the full pool and on the lane. The lane is the fraction c as deep, so it slips more, and the increase, the split drag*, is their difference,*

$$s_{\text{lane}} - s_{\text{full}} = \frac{\Delta x}{cX + \Delta x} - \frac{\Delta x}{X + \Delta x}, \tag{4.8}$$

growing as the split c shrinks and vanishing as $c \to 1$.

The holder is never tied to the lane. If public flow has moved the price in their favour they trade the public pool and leave the lane unused, and if it has moved against them the lane still waits at its opening price.

4.1.3 The merge

The lease ends at T, when the lane merges back. Write the public pool at (X_p, Y_p) and the lane at (X_ℓ, Y_ℓ), the split reserves (cX, cY) evolved by the holder's swaps (§4.4.1).

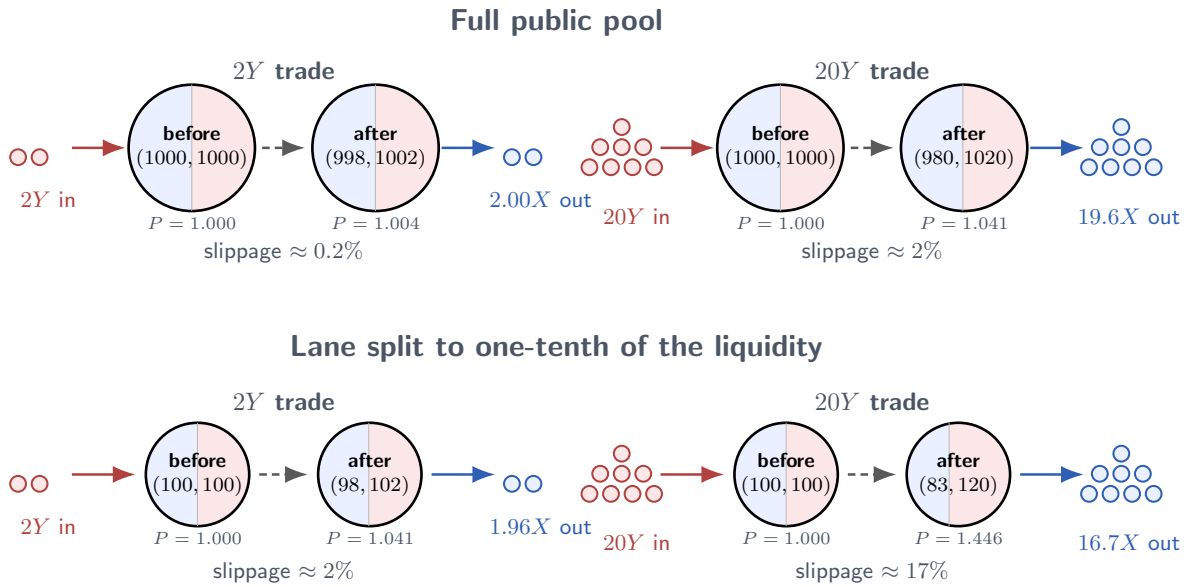


Figure 4.3: The same trade on the public pool and the lane, the thinner lane moving its price further for the equal input.

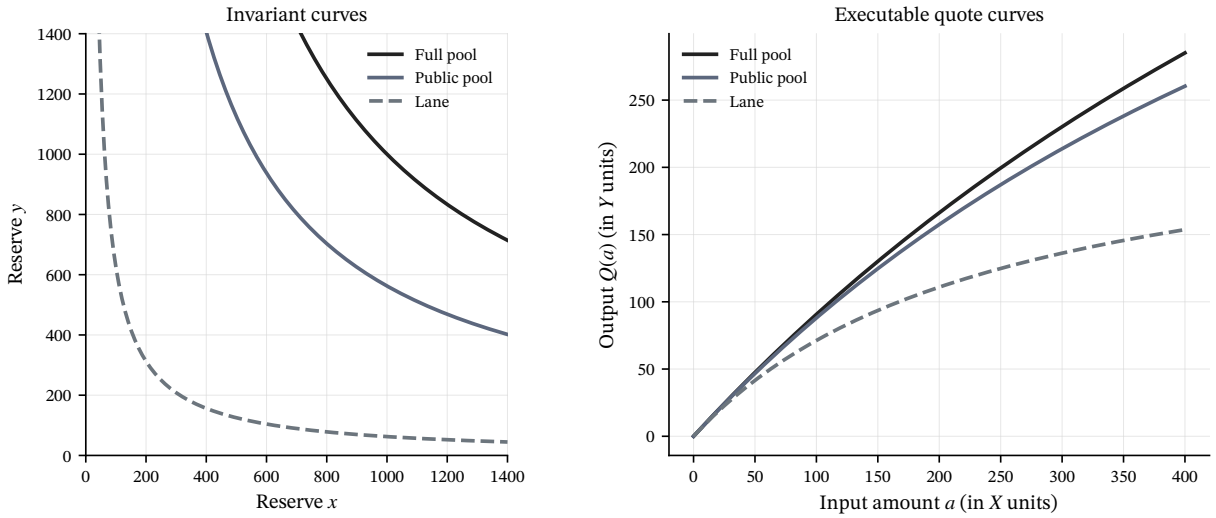


Figure 4.4: Public pool and lane on two shallower curves through the same price, the split conserving the parent's geometric-mean liquidity.

The merge adds the two pools reserve by reserve,

$$X_{\text{merged}} = X_p + X_\ell, \qquad Y_{\text{merged}} = Y_p + Y_\ell, \tag{4.9}$$

into a single pool whose invariant is

$$k_{\text{merged}} = X_{\text{merged}}\, Y_{\text{merged}}. \tag{4.10}$$

Writing

$$\rho \;=\; \frac{P_{\text{lane}}}{P_{\text{public}}} \tag{4.11}$$

for the lane-to-public price ratio at expiry, this evaluates to

$$k_{\text{merged}} = k\Big[(1-c)^2 + c^2 + c(1-c)\big(\sqrt{\rho} + \tfrac{1}{\sqrt{\rho}}\big)\Big], \tag{4.12}$$

a surplus over the parent's k of

$$k_{\text{merged}} - k = c(1-c)\,k\Big(\sqrt{\rho} + \tfrac{1}{\sqrt{\rho}} - 2\Big), \tag{4.13}$$

non-negative since $\sqrt{\rho} + 1/\sqrt{\rho} \geq 2$, and zero only at $\rho = 1$.

Proposition 4.2 (Merge surplus)**.** *$k_{\text{merged}} \geq k$, with equality exactly when the two pools merge at the same price and strict inequality otherwise, and the surplus grows strictly as the two prices diverge (derived and proved in Appendix B).*

The merge is safe for the liquidity providers. Under the model it needs no price oracle and no outside capital. When the holder keeps the lane near the public price, $\rho \approx 1$ and the merge rebuilds the parent almost exactly, while swaps that drive the prices apart grow the surplus, which stays with the providers (Figure 4.5).

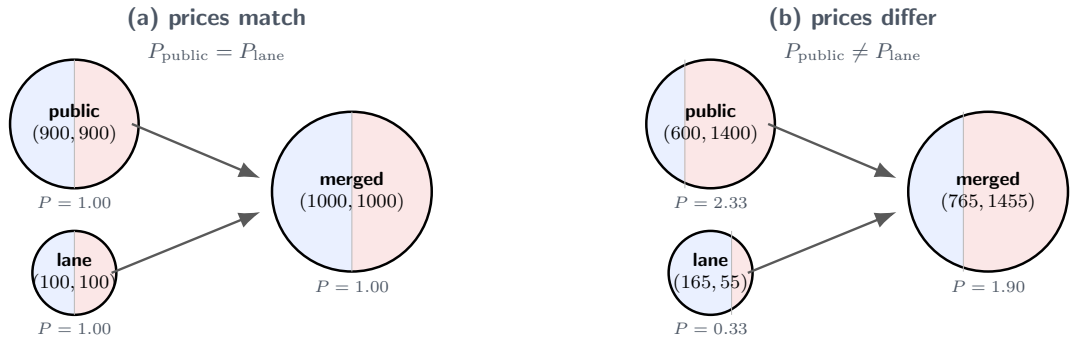


Figure 4.5: The merge at T, the lane's reserves rejoining the public pool at the lease end and the two curves recombining into one.

The merge shock. The merge moves the price as well as the invariant. The merged pool's price is

$$P_{\text{merged}} \;=\; \frac{Y_p + Y_\ell}{X_p + X_\ell}, \tag{4.14}$$

the public and lane prices weighted by their X reserves, so the new price sits between the two and replaces both at T. To a trader watching the public pool this is a discontinuous *price shock* that grows with how far the prices drifted over the lease (Figure 4.6).

The shock is the one risk the merge carries, and it comes from being scheduled. Mechanically the merge is no different from an ordinary swap, but it happens at a predictable block T, so its direction and size can be read in advance, making it easier to front-run or sandwich [28]. Containing this predictable merge is an engineering problem, taken up in Chapter 5.

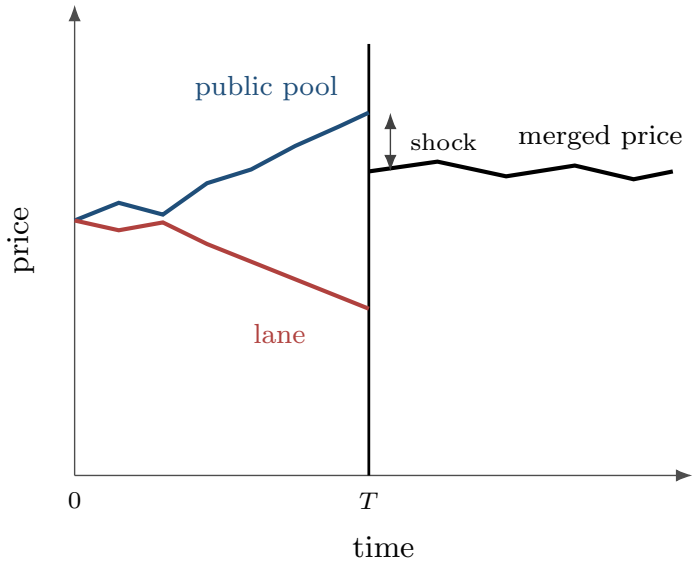


Figure 4.6: The price shock at the merge, the public price stepping as a lane whose price has drifted folds back at the known time T.

4.2 Pricing lane options

The holder pays an upfront amount Π, the *premium*, at the split, in either of the AMM's two reserve tokens. The liquidity provider posts a flat rate per block of lease, the way an AMM posts a swap fee [11, 15], and the contract charges that rate on the leased notional over the whole lease, so the premium grows with both the size of the slice and the number of blocks it is held. The rate itself is posted, not derived on chain from volatility or an option model. Pricing a lane exactly is a complex topic which can be further studied. It inherits the same LP risk as an ordinary AMM, including impermanent loss [13, 20, 23]. The only new cost is the public flows it forgoes for time T, so an LP paid for it does at least as well as holding the position in an open pool.

The premium the LP posts covers three things,

- Π_{opp}, the opportunity cost, the public fees the leased capital forgoes while locked into the lane, rising with the lease length T, the leased notional, and the pair's volatility and flow.
- Π_{base}, a base fee for the fixed cost of splitting the lane off and merging it back, owed even when the opportunity cost is near zero.
- Π_{fee}, a partial credit for the lane swap fees the holder pays back to the LP over the lease, reducing the premium but not below zero.

These combine into the premium

$$\Pi \;=\; \Pi_{\text{opp}} + \Pi_{\text{base}} - \Pi_{\text{fee}}. \tag{4.15}$$

The dissertation does not solve optimal lane pricing. The opportunity cost turns on private knowledge, how an LP values its own flow, so the rate is posted rather than computed, with competition between AMMs disciplining the level.

4.3 Where the lanes live

The two-chain setting of Chapter 3 places the AMM on one EVM chain and the holder on another. Let C_M be the AMM's chain and C_E the holder's (§3.2.2), separated by the delay τ. A lane splits off from the AMM, so it opens on C_M, and can stay there or move to C_E, where the holder already trades. The first placement is the *AMM-side lane*, kept on C_M, and the second the *portable lane*, the bridged form carried to C_E (Figure 4.7).

4.3.1 The AMM-side lane

Definition 4.2 (AMM-side lane)**.** *An* AMM-side lane *is a lane that stays on C_M with the AMM. The holder reaches it from C_E, a delay τ away.*

It is the lane of §4.1 unchanged. The split and the merge run on C_M, and every holder swap is a cross-planetary swap (Definition 3.4) crossing the delay to reach it.

4.3.2 The portable lane

The portable lane moves the lane to C_E, so the holder trades it at home instead of across the delay.

A lane's whole state is its reserves (X_ℓ, Y_ℓ) on the invariant $k_\ell = X_\ell Y_\ell$, the holder allowed to trade it, and the expiry at T that ends the lease, and that is all that crosses, carried by the relay of §3.2.2. The reserves cross by token transfer, burned on C_M and minted on C_E,

$$\mathrm{burn}_{C_M}(X_\ell, Y_\ell) \;\longrightarrow\; \mathrm{mint}_{C_E}(X_\ell, Y_\ell), \tag{4.16}$$

and the invariant, the holder, and the expiry cross alongside in the message. A custodian on C_E mints the reserves and runs the lane until expiry.

The holder opens the lane from C_E with a message to the AMM on C_M a delay τ away, and at expiry the custodian sends the terminal reserves back over τ for the AMM to merge into the public pool. Only the holder's swaps with the lane stay local to C_E and pay its block time.

Definition 4.3 (Portable lane)**.** *A* portable lane *is a lane whose state, the reserves (X_ℓ, Y_ℓ), the invariant k_ℓ, the holder, and the expiry at T, is transferred at the split from C_M to a custodian on C_E, which runs it locally and returns the terminal reserves to C_M to merge.*

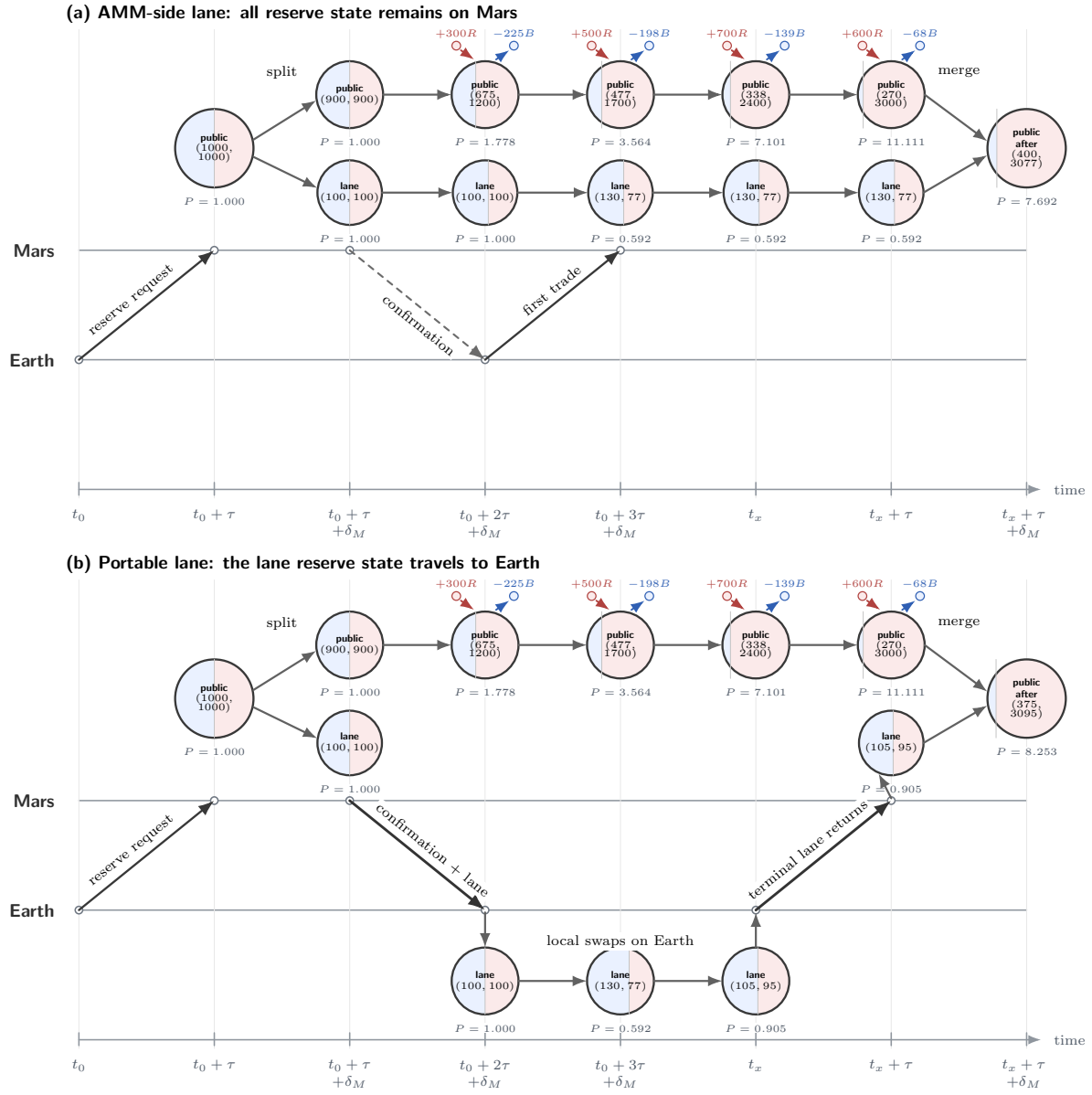


Figure 4.7: The two placements on the Earth–Mars example, time left to right, the AMM and its public pool on Mars (C_M) and the holder on Earth (C_E), one delay τ apart. **(a) The AMM-side lane** stays on Mars, so each holder swap waits τ out and τ back while public trades move the pool. **(b) The portable lane** bridges its reserves to Earth, where the holder trades locally, and only the terminal reserves cross back to Mars to merge.

The lease must leave room for the round trip. Let ε be the local trading window, the time the holder has on C_E to make swaps. With the split at time 0 on C_M, the reserves reach C_E at τ and the custodian must start the return a delay before the merge, so the total lease length is

$$T \;=\; 2\tau + \varepsilon, \tag{4.17}$$

two delays for the round trip and ε for the holder's local window, and the premium of §4.2 prices the lane for that whole span. What the portable lane gives up is the fallback of §4.1, because the public pool stays on C_M and a holder who would rather take the public price must cross the delay to reach it (Figure 4.7).

4.4 Lane options under the three lags

The AMM-side lane keeps the holder a delay from their lane, while the portable lane carries it to the holder. The two agree on price lag and differ on execution lag and capital lag.

4.4.1 The AMM-side lane

Price lag is cleared. The split opens the lane at the AMM's marginal price and no public trade moves it afterwards (Definition 4.1), so a holder swap of input Δx against the lane's reserves (X_ℓ, Y_ℓ) returns

$$\Delta y = \frac{Y_\ell \, \Delta x}{X_\ell + \Delta x}, \tag{4.18}$$

with (X_ℓ, Y_ℓ) starting at (cX, cY) and moving only with the holder's own swaps.

Proposition 4.3 (Lane price-lag isolation)**.** *Writing (X, Y) for the parent reserves at the split, the lane opens at $(X_0^\ell, Y_0^\ell) = (cX, cY)$ and evolves under the holder's i-th swap of input Δx_i as*

$$X_i^\ell = X_{i-1}^\ell + \Delta x_i, \qquad Y_i^\ell = \frac{c^2 XY}{X_i^\ell}, \tag{4.19}$$

holding the lane invariant c^2XY. Each output depends only on the split-time reserves and the holder's own prior swaps. No public swap appears.

What remains is a single price lag at the split, where the lane opens and its premium is priced against an AMM price one delay τ old.

Execution lag is not addressed. The lane is on C_M and the holder on C_E, so a swap sent at time t executes only at $t + \tau$, leaving the execution lag

$$L_{\text{exec}} = \tau \tag{4.20}$$

unchanged from the bare cross-planetary case (§3.3.2). Isolating the price does nothing for the distance the order travels.

Capital lag Writing Π for the premium, N for the number of holder swaps over the lease, each of notional K, and r for the opportunity rate, the capital cost is

$$F_{\text{cap}} = r\Pi\tau + 2KNr\tau = r\tau\big(\Pi + 2KN\big), \tag{4.21}$$

the opportunity cost on the premium for the one delay to open the lane, plus a round trip of 2τ for each of the N swaps.

Verdict The AMM-side lane clears price lag, leaves execution lag unchanged, and worsens capital lag, the holder still a delay away from the lane.

4.4.2 The portable lane

Price lag is cleared exactly as for the AMM-side lane. The split is identical, so Eq. (4.18) and the single residual price lag at the split hold whichever chain hosts the lane. Carrying the lane to C_E moves the chain, not the price.

Execution lag is cleared, though not to zero. Once the lane is on C_E the holder's swaps are local transactions that settle in a block, so L_{exec} falls from τ to a single block time and the holder fits far more trades into the same lease. The AMM-side lane left this lag untouched.

Capital lag is not cleared but traded, because the liquidity itself must travel. The holder's swaps are local, so the per-swap crossing of the AMM-side lane is gone, replaced by one round trip of the lane, out at the split and back at the merge. Writing V for the total value of the lane's two reserves X_ℓ and Y_ℓ that make the crossing, the capital-lag cost is

$$F_{\text{cap}} = r\Pi\tau + 2Vr\,\tau = r\tau\big(\Pi + 2V\big), \tag{4.22}$$

fixed in the swap count, where the AMM-side cost of Eq. (4.21) grows with it. The portable lane is therefore cheaper for a holder who trades often and costlier for one who opens a large lane and trades it once or twice.

Verdict The portable lane clears both price lag and execution lag and turns capital lag into a choice.

4.5 Advantages of lane options

A constant-product AMM (§2.2) on one chain cannot serve a trader on another chain, where the delay leaves every order stale on arrival. The lane option fixes this by splitting a private lane off from the AMM in exchange for a premium, closing it with a merge that, under the model, does not reduce the constant-product invariant. The premium meets the cost of reserving public liquidity.

Where the lane sits decides what it fixes. Table 4.1 sets the two placements against the three lags.

Lag	AMM-side lane	Portable lane
Price	**Cleared.** The split isolates the lane from public flow, so only the split itself is stale, once.	**Cleared.** The split is the same, so isolation holds whichever chain hosts the lane.
Execution	**Not addressed.** The lane is a cross-chain delay away, so every swap pays τ to execute.	**Cleared.** The lane sits with the holder, so swaps settle locally in a block.
Capital	**Worsened.** Every swap crosses the delay, so the cost $r\tau(\Pi + 2KN)$ grows with the swap count N.	**Traded.** Only the lane crosses, at the fixed cost $r\tau(\Pi + 2V)$ rather than a cost that grows with N.

Table 4.1: The two placements against the three lags of Chapter 3.

4.5.1 Wider uses

The interplanetary delay was only the sharpest case of a general need for private, time-bounded liquidity that one party controls, worth paying for wherever the open market works against the holder, not only at a distance.

Resisting MEV. On a public AMM a pending swap is visible before it settles, so others can buy ahead and sell into the price it moves, a *sandwich attack* [28] (§2.5). A lane prevents this inside the lane, because only the holder can trade it, though it does not hide the holder's own-chain transactions.

Cross-chain liquidity. Cross-chain markets already face fragmented liquidity [38]. A chain with little liquidity can borrow from a well supplied one. A portable lane split off from a large AMM is bridged to the thin chain for a lease, so traders there swap against it locally, and at expiry it returns home. The liquidity is rented for a fee rather than moved for good, so the large AMM keeps its position and the thin chain has a working market for the lease.

Chapter 5

Engineering smart contracts under delay

Lane options (Chapter 4) are a mathematical primitive and can be implemented as a Uniswap v4 hook [12], a Solana program, a Move module, or on any other smart-contract platform supporting programmable state transitions [7]. Here it is built as a standalone EVM smart contract, engineered around the two EVM constraints of §2.1, then extended to carry a lane across two chains within the added cross-chain constraints of §2.3, as a portable lane.

5.1 A simple AMM contract

A constant-product AMM is a small smart contract (§2.1). It holds reserves of two ERC-20 tokens [17], X units of one and Y of the other (§2.2), whose product $k = XY$ is the constant-product invariant. A single owner provides the reserves, depositing or withdrawing the two tokens, and any trader swaps against them, the contract taking the input token, keeping a small fee, and returning the other token in the amount the constant-product rule allows, leaving k unchanged [11].

Each of these is a constant-time write, turning on neither the pool's size nor its history, and the reserves can be read off chain at no gas cost. All three functions are implemented as a standard constant-product pool.

5.2 The lane options contract

This section extends the simple AMM of §5.1 with the lane option primitive of Chapter 4. The simple AMM's reserves become the contract's *public pool*, the part anyone may trade, and a lane is a private slice of those reserves, held by one trader for a lease.

Opening a lane moves no tokens. The contract's reserves stay put, and it only records

that a fraction is set aside for the lane and the rest is the public pool. What it holds is always the public reserves plus every unmerged lane's reserves, so splitting a lane off, or merging it back, is bookkeeping within the contract between variables, not a token transfer. This keeps the operations cheap in terms of gas (§2.1).

Against this accounting, opening and trading a lane are easy. The difficulty is the return, folding an expired lane back into the public pool, and it meets both EVM constraints of §2.1. The gas limit (Constraint 1) shapes how a lane is stored, opened, and merged, so the work stays small. The absence of a scheduler (Constraint 2) shapes who merges a lane, and when.

5.2.1 Constraint 1, the gas limit

Every state-changing call runs in one transaction, capped by the block gas limit (§2.1), so no operation may do work that grows without bound. For lanes this creates an issue at the return. Were the contract to loop over every expired lane one at a time, a large enough backlog would exceed the budget and revert the merging trade [6]. This also creates an attack vector: a malicious actor can open many tiny lanes at distinct end-blocks, inflating the backlog until a close exceeds the block gas limit and reverts. The right data structure and storage can help reduce this.

Storing a lane. A lane is a record of its two reserves and the block its lease ends on (Table 5.1). Opening a lane moves no tokens. Only the record and the public totals are updated, never the contract's token balances. The reserves held in unmerged lanes are off limits to the liquidity provider, who may withdraw only the public pool, so a holder cannot have a lane pulled from under them.

The return must fold a whole cohort of lanes back in one step, never one lane at a time. So the contract denormalises the cohort, keeping for each end-block a *block total*, the aggregate sum of the reserves of every live lane that ends there, stored rather than recomputed by scanning.

Definition 5.1 (Block total)**.** *For end-block b, with T_ℓ the end-block of lane ℓ, the block total is*

$$X_b^\Sigma = \sum_{\ell:\, T_\ell = b} X_\ell, \qquad Y_b^\Sigma = \sum_{\ell:\, T_\ell = b} Y_\ell. \tag{5.1}$$

It stays current as lanes change, since each open or trade of a lane ending at b adds its change into the total, a constant-time update. Without the block total, merging L lanes expiring at the same block would cost $O(L)$ at merge time. The aggregator reduces this to $O(1)$ at merge, at the cost of an extra write when a lane opens, so the work is borne by the opener.

Merging every lane that ends at b is then one addition of its block total into the public

Field	Type	Purpose
Lane record (struct Lane)		
`X`, `Y`	uint	The lane's two reserve balances
`endBlock`	uint	Block at which the lease ends
Contract state		
`publicX`, `publicY`	uint	Public pool reserves, tradeable by anyone
`owner`	address	The single liquidity provider
`laneX`, `laneY`	uint	Total reserves across all unmerged lanes; off limits to the liquidity provider
`laneState[id]`	Lane	Reserve record for lane `id`
`laneOwner[id]`	address	Holder of lane `id`
`blockTotal[block]`	(uint, uint)	Aggregate (X_b^Σ, Y_b^Σ) for all lanes ending at `block`
`expiryHeap`	min-heap	Distinct end-blocks in order, earliest first

Table 5.1: The lane options contract state, the public pool and owner, the reserves held across live lanes, each lane's record and holder, and the structures that make the merge cheap.

reserves,

$$(publicX, publicY) \mathrel{+}= (X_b^\Sigma, Y_b^\Sigma), \tag{5.2}$$

the same work whether the cohort ending at b held one lane or a thousand. To find the due blocks without scanning, the distinct end-blocks sit in a binary *min-heap*, soonest at the root, so the earliest is read at once and an insert or a merge costs $O(\log B)$ for B distinct blocks [47]. A second lane on an existing block grows that block's total rather than adding a node, so the heap holds one node per distinct end-block only (Figure 5.1). The merge cost falls from L, the number of unmerged lanes, to B, the number of distinct blocks that contain unmerged lanes. Since each lane expires on exactly one block, $B \leq L$.

Opening costs $O(\log B)$ for the heap insert, not $O(1)$. This is the trade-off the design makes. The opener pays a little more so every merge stays cheap.

Opening a lane. Opening reserves a slice for the holder (Figure 5.2). It charges the premium and the bond (§5.2.2), moves a fraction c of each public reserve into the lane's account, records the lane, and inserts its end-block on the heap if new.

5.2.2 Constraint 2, no scheduler

The chain is passive (§2.1). There is no way to write a scheduled job, a cron that fires at the lease's end to rejoin a lane to the public pool. Every change is driven by an outside transaction, so the merge happens only when someone sends it. The contract can incentivise someone to do this.

The two natural candidates who should return the lane's liquidity are the holder and the liquidity provider, but neither works reliably on its own.

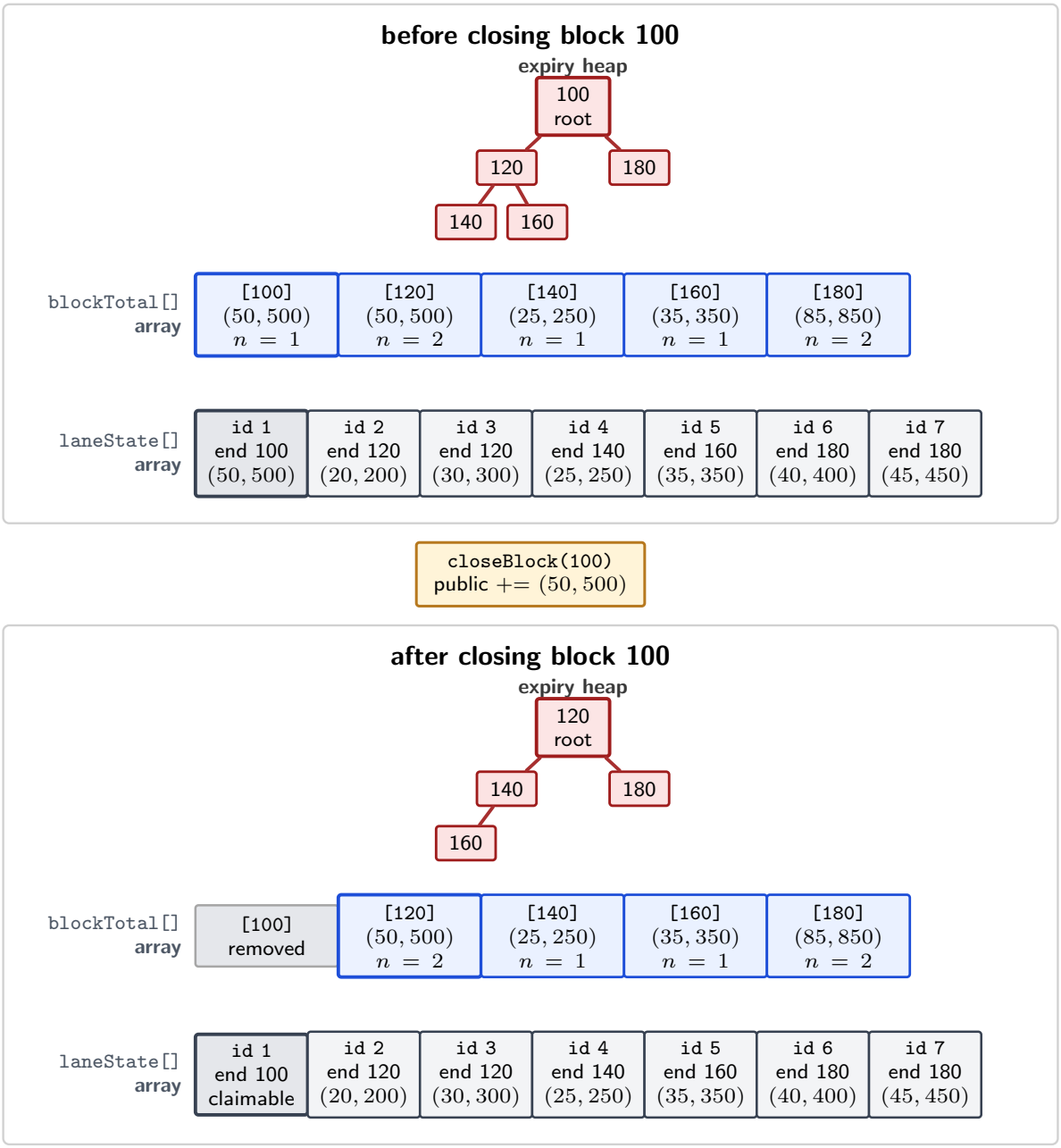


Figure 5.1: The expiry heap and block totals. Blue entries store the aggregate reserve pair $(X_b^{\Sigma}, Y_b^{\Sigma})$ and lane count n for each end-block b. Grey entries store one lane's reserves and the end-block recorded for that lane.

The holder. A rational holder has no incentive to pay gas to close a lane once they have finished using it. One who never closes locks their slice out of the public pool indefinitely. An adversary does so deliberately, opening many small lanes and never returning them.

The liquidity provider. Cleaning up after absent holders is an unbounded, active task. While lanes sit unclosed the LP foregoes the public fees that liquidity would have earned, on top of the premium already collected.

Using a bond as a pseudo-scheduler. The contract has no scheduler, so it pays for the work one would do. To open a lane the holder deposits a small *bond* in the chain's

```
openLane(c, duration):

    # the bond pays whoever later closes the lane
    charge the premium and the bond from the caller

    # reserve a c-fraction of the public pool for the lane
    take c of publicX and c of publicY out of the public pool
    hold them as the lane's reserves

    # record the lane and queue it for the merge
    make a new lane ending after duration, owned by the caller
    add it to its end-block's total and the expiry heap

    return the lane's id
```

Figure 5.2: Opening a lane, charging the premium and bond, moving a fraction c of each reserve into the lane, and queuing its expiry on the heap.

native gas token (§2.1), paid on top of the premium. Unlike the premium, the bond is refundable. The bond is about twice a close's gas, and the contract hands it to whoever runs the close. The merge becomes a paid task anyone may take, a *pseudo-scheduler* that stands in for the timer the EVM lacks. A lane left open past its lease, by an adversary or a forgetful holder alike, is closed by whoever wants its bond, while the one who left it forfeits the deposit.

A close is final. Closing returns the lane's reserves to the public pool and deletes the lane's record together. The deletion matters most on an early close, before the lease ends, since without it a holder could close to reclaim the bond and then keep trading a slice that is already back in the public pool. With it, a swap against a closed lane finds no record and reverts, so each lane is closed once and for all, and the bond leaves exactly once.

Closing takes two forms. A *single-lane close* returns one specific lane early and may only be called by its owner. A *block close* merges every lane expiring at a given block in one step using its block total (§5.2.1). It is the path used by traders and by *watchers*, outside parties who monitor the chain for unclosed expired lanes and run the close to collect the bond.

A lane is closed in exactly one of three cases, and the bond goes to whoever closes it.

Closed early by the holder. Before the lease ends the holder calls a single-lane close on their own lane. The reserves rejoin the public pool, the record is deleted, and the bond is refunded, covering the gas the close cost.

Closed by a trade. The lease has ended and a trade arrives before any watcher does. The trade runs a block close first, merging all lanes due at that block, then swaps against the restored pool. The bond pays the trader, reimbursing the extra gas.

Closed by a watcher. Any outside party acting as a watcher may call the block close directly after expiry to collect the bond.

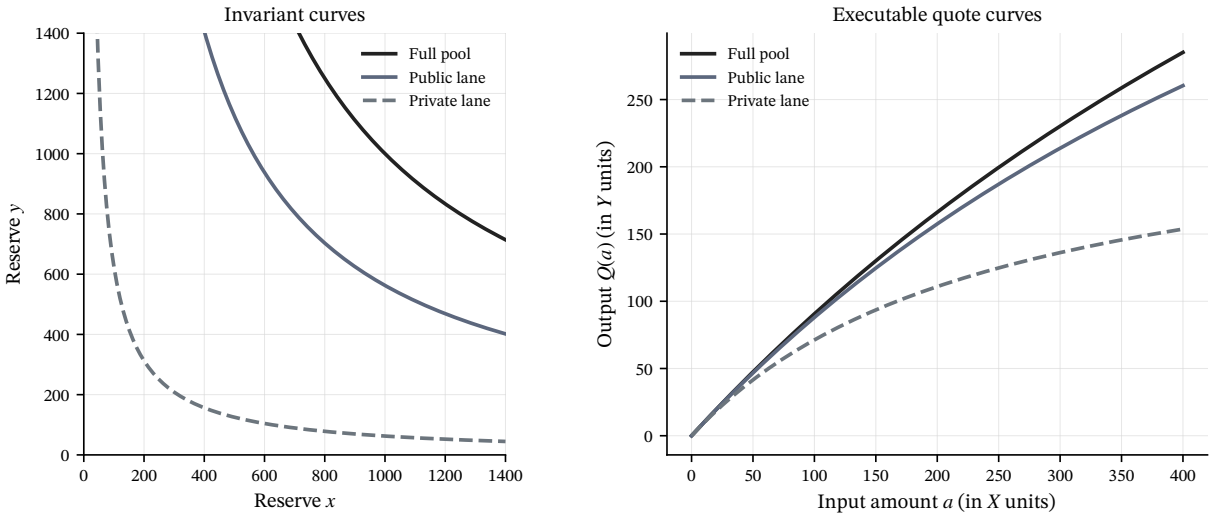


Figure 5.3: A trade meeting an expired but unmerged lane would price against the shallower curve the lane left behind; the lane is closed first so the trade sees the full restored pool.

```
trade(tokenIn, amountIn):

    # first return expired lanes to the public pool,
    # as many as the gas budget allows
    close all due lanes within budget

    # then swap against the whole, restored pool
    swap against the public pool
```

Figure 5.4: A trade clears the expired lanes within its gas budget, then swaps against the restored public pool.

Trading clears first. The last case is how every trade keeps the public pool whole without a scheduler. A trade that met an expired, unmerged lane would price against reserves smaller than the public pool holds (Figure 5.3), so every trade closes the expired lanes before it executes, then swaps against the restored pool (Figure 5.4). One transaction is capped by the block gas limit (Constraint 1, §2.1), so the close cannot run forever. A trade clears at most a fixed *budget* of expired blocks, *maxClose*, set by the caller per transaction, and leaves any beyond it for the next trade or a cleanup call. If the backlog is large, consecutive cleanup calls drain it one block at a time until the pool is fully restored. A read-only quote is computed off-chain and costs no on-chain gas, so it may take $O(B)$ time; it sums the public reserves and the block totals of any lanes due but unmerged, keeping the quoted price correct even during a backlog.

Table 5.2 collects every operation. None grows with the live-lane count, and the gas limit caps B at a number that never binds in practice.

Operation	Kind	Cost	What it does
add liquidity	write	$O(1)$	the owner deposits into the public reserves
remove liquidity	write	$O(1)$	the owner takes from the public reserves only
open a lane	write	$O(\log B)$	post the bond, reserve a slice, add it to its block total, insert its block on the heap
trade a lane	write	$O(1)$	the holder swaps against one lane record
transfer a lane	write	$O(1)$	hand a lane record to a new holder, one write
reclaim a lane	write	$O(1)$	delete a merged-back lane record to free its storage
close own lane	write	$O(\log B)$	the holder merges their live lane back early and reclaims the bond
close a block	write	$O(\log B)$	merge one due block, pay the caller one bond per lane
public swap	write	$O(\mathit{maxClose} \log B)$	clear up to *maxClose* due blocks, then swap against the public reserves
cleanup	write	$O(\mathit{maxClose} \log B)$	clear up to *maxClose* overdue blocks, anyone, for the bond reward
quote a swap	read	$O(B)$	walks the block totals, off chain
quote a premium	read	$O(1)$	the LP-posted rate, off chain

Table 5.2: The cost of every operation. B is the number of distinct end-blocks, never more than the number of live lanes, and *maxClose* the per-call close budget the caller sets.

5.3 The portable lane options contract

The AMM-side lane sits with the AMM, so a holder on the other planet pays the cross-planetary delay on every swap. The portable lane removes that by carrying the slice to the holder's chain. It is the same economic object, split from and merged into the AMM on C_M, but the two chains share no state (§2.3), so the slice and its terms cross as messages to a *custodian* contract on C_E that holds the reserves and serves the holder's swaps locally over the lease.

5.3.1 The bridge

The portable lane rides on the bridge of §2.3, a pair of bridge contracts and a trusted relay. The design uses one minimal primitive, the same on both chains and carrying a single kind of message, a call to run on the destination (Figure 5.5). A contract gives its local bridge a target and the calldata, the relay carries the message across the delay, and the counterpart bridge runs the call, applying it at most once by its source nonce, since the relay may deliver a message more than once and in either order [34, 35]. Tokens and the lane both ride on this primitive.

The relay is trusted. The contracts accept any message it delivers and have no means

to verify it independently. This is the single third-party security dependency this implementation adds beyond the AMM-side contract. In production, a proof-carrying or light-client bridge should replace the trusted relay. A light-client bridge can replace the relay with an on-chain proof of the source chain's state, and a ZK bridge can compress that proof to a single succinct argument [37], at the cost of significantly higher complexity and per-message gas.

```
# source chain, ask the bridge to run a call on the other chain
bridge.send(target, calldata):
    nonce = nextNonce++
    emit Message(nonce, target, calldata)
```

```
# relay
watch Message(nonce, target, calldata)
after delay, call deliver(nonce, target, calldata)
```

```
# destination chain, the counterpart bridge runs the call just once
bridge.deliver(nonce, target, calldata):
    # a redelivery is ignored
    require not applied[nonce]
    applied[nonce] = true
    target.call(calldata)
```

Figure 5.5: The bridge primitive, a contract sends a call, the relay carries it across the delay, and the counterpart bridge runs it once.

5.3.2 Creating the custodian

Each public pool has one *custodian*, a contract on C_E holding its lanes and running each as a local market maker for its holder over the lease (Definition 4.3, Figure 5.6).

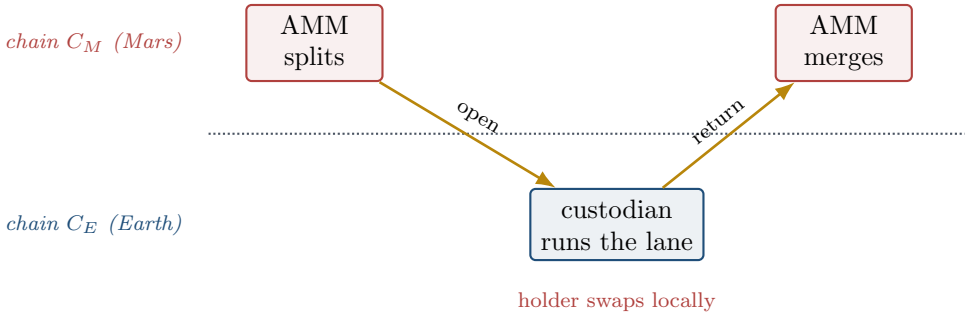


Figure 5.6: A portable lane's life, split on Mars, run by a custodian on Earth over the lease, then returned to Mars to merge.

The custodian is deployed once with the pool, not per lane, so the same custodian holds every lane keyed by its id, and funds leave only on a bridge message from the AMM. Sitting on different chains, the AMM and custodian cannot know each other's address

at deployment, so each starts with its peer empty and is bound once by a one-shot initialization call. The reverse direction uses a second pair.

5.3.3 Moving the slice across

A token lives on one chain (§2.3), so the slice is moved, not copied. A bridgeable token burns a stated amount on the source and mints the same amount to a named recipient on the destination, carried by the bridge call of Figure 5.5 (Figure 5.7) [33, 34]. Between burn and mint the units exist on neither chain, so the lane's reserves are unusable on either chain at that moment. At all other times they are live on exactly one chain.

```
# move an amount of a token from C_M to C_E
transfer(token, amount, recipient):
    # the units leave C_M
    token.burn(amount)
    # a bridge call carries the mint across
    bridge.send(token_on_C_E, mint(amount, recipient))

# on C_E, delivered by the bridge
token_on_C_E.mint(amount, recipient):
    # the units reappear on C_E
    balance[recipient] += amount
```

Figure 5.7: A cross-chain token transfer, the amount burned on the source chain and minted to the recipient on the destination by one bridge call.

5.3.4 Opening and running a lane

A split is one atomic transaction on C_M (Figure 5.8). It burns the slice and sends the two token amounts and the lane terms across as separate bridge messages, which on C_E may land in any order. The custodian treats opening as assembly, holding each message as it arrives and opening the lane once all are in. A redelivery changes nothing, since the bridge has recorded its source nonce.

Opening charges the premium and posts the close bond just as on one chain (§5.2.1). The premium in X and Y stays in the home pool, the bond in the native token is held there, and only the split-off reserves cross. Once open, the custodian serves the holder's swaps locally with no bridge hop per trade. Unlike the AMM-side lane, the custodian has no public pool to route a fee into, so it keeps the fee in the lane. The product grows over the lease, and the terminal balance comes home with the fees inside.

5.3.5 Returning the lane at expiry

At expiry the reserves are still on C_E, and nothing fires on its own there, so the return is transaction-driven like the AMM-side merge (§5.2.2), in two stages. After the lease,

```
# on C_M: one atomic transaction
openPortableLane(c, T, holder h):
    charge the premium and post the bond
    burn c*publicX, c*publicY
    transfer(X, c*publicX, custodian)
    transfer(Y, c*publicY, custodian)
    bridge.send(custodian, open(h, T))

# on C_E: assemble when all three messages arrive
custodian.onMessage():
    accumulate X, Y, and lane terms (h, T)
    once complete: open the lane, serve swaps until T
```

Figure 5.8: Opening a portable lane, the home chain burning the slice and sending the reserves and lease terms as separate messages the custodian assembles.

anyone may call settlement on the custodian, which burns the terminal reserves and sends the return through the bridge. The custodian refuses any swap once the lease ends, so settlement alone moves the reserves. When the return lands on C_M, the terminal reserves are minted by the bridge and merged into the public pool in one step, not the close-a-block-cohort approach of §5.2.2. The accrued fees and the divergence surplus, the extra liquidity the merge returns when the lane price has drifted from the public price (Proposition 4.2), come back inside the terminal balance. The bond posted at open is paid to whoever settles the lane in the same step, just as an AMM-side close pays whoever merges the block.

5.4 Contract analysis

Both contracts run on the EVM, a compute-limited execution environment with no autonomous scheduler and a hard cap on the work one transaction may do (§2.1). With no scheduler to fold an expired lane back, the merge rides on the next ordinary trade or a bonded cleanup task, keeping the public pool whole without a keeper, and every on-chain operation stays bounded, logarithmic in the number of distinct expiries and flat in the number of live lanes. Carrying a lane across two chains adds no work the chain cannot do, and what the contract cannot itself guarantee narrows to the single stated trust in the relay.

A suite of 48 Foundry tests [9], all passing, exercises the lane lifecycle, the closing machinery, and the guards of §5.4.1 over the deployed bytecode.

5.4.1 Security analysis

The lane options contract and its portable form carry the lane records, the close machinery, the custodian, and the bridge, but leave some economic limits to the deployment. Table 5.3 draws the line between what the prototype runs, what it specifies but leaves unenforced, and what it places outside its trust boundary.

Category	**Items**
Implemented in the prototype	lane split and merge accounting, holder-only lane swaps, expiry closure, close-gas bond accounting, the bridge message path, the reentrancy lock [31]
Specified but not enforced	a maximum lease and a live-lane limit, a utilisation-scaled premium
Out of scope or assumed	a trusted relay or proof-carrying bridge, recovery from a halted remote chain

Table 5.3: The prototype-versus-production boundary. The middle row names the guards the reference specifies but leaves to a deployment to enforce.

Threat 1: front-running the merge shock. The merge shock of §4.1.3 reappears here in engineering form. Because a due lane merges only when the next trade or a cleanup call reaches it, an attacker could trade against the smaller, not-yet-merged reserves at a stale price, then merge and trade back [18]. The shock is weak even unguarded. Because a lane can be closed early (§5.2.2), the merge can land on any block before the lease ends, at a time only the holder knows, so a pre-positioned attacker has no block to target. A holder who instead tries to manufacture the shock deliberately, diverging the lane to profit from the merge, loses, since creating the divergence in the thin lane costs more slippage than the shock returns (Appendix C). On the portable lane it is harder still, the lane on a different chain, so an observer cannot see the reserve state or price of the lane and cannot tell which direction the price has moved. Any attempt to front-run is also competing with other closers racing for the bond, making profitable front-running impractical.

Threat 2: capturing or thinning the public pool. A caller may split off almost the whole pool into one long lease for the flat premium, collapsing public liquidity, or open many small lanes at distinct end-blocks, inflating B and slowing the $O(\log B)$ heap operation per new block. Even an honest lane thins the public side, so every other trader and holder slips more, the same swap moving the price further, and the holder pays for their own slice but nothing for the worse market they leave behind. A cap on the fraction of the pool that may be leased, a limit on the number of live lanes, or a dynamic premium that rises with the total fraction already carved each guard against this.

Threat 3: the bond bounty as MEV. Closing a lane pays the bond to whoever sends the closing transaction, and a holder who closes early reclaims their own (§5.2.2). That open bounty is the kind of value a validator (also called a block producer, §2.1) is placed to seize [18]. It can leave out the holder's close transaction, waiting until the lease expires, then insert its own close transaction to take the bond, so the holder loses the deposit having done nothing wrong. And when several watchers race to claim an expired lane's bond, each bids a higher priority fee [18] to land first, and most of the reward flows to the validator as fee rather than to the closer.

Threat 4: the trusted relay. Carrying the lane across the relay adds no new attack surface to the lane mechanism itself. The surface it adds belongs to the bridge (§2.3), the portable lane inheriting the trust the bridge already rests on, the single relay of §3.2.2 being honest. A dishonest relay could forge an open to mint a slice with no matching burn, fabricate a settlement to pull reserves never returned, or drop a message to strand a lane, none of which the contracts detect [35]. Running several relays in parallel removes the *liveness* risk but not the *trust*, since each still asserts the source event on its word. Removing the trust needs a proof-carrying bridge (§2.3, [37]) whose message carries a proof the destination verifies on chain.

Chapter 6

Evaluation

This chapter evaluates lane options by running the lane-option contracts of Chapter 5 on a two-chain testbed under delay and measuring, across the three lags and the costs a lane adds, where each of the simple AMM, the AMM-side lane, and the portable lane wins.

The experiment needs a testbed no existing tool provides (§6.1.1), met here by StateLab (§6.1), an open-source framework built for it [8, 10].

6.1 StateLab

6.1.1 Existing frameworks

The standard ways to run EVM contracts for testing are public test networks and local development nodes. A public test network such as Ethereum Sepolia runs real bytecode on a live chain, but seals its blocks in real time with no fast-forward. With a one-way Earth-to-Mars delay of 24 minutes against a 12-second block time, a single relay crossing spans about 120 blocks (§3.2.2), and a campaign over many seeds and parameters would take hours of wall-clock time to sit through. A public network is also a single chain.

Local development nodes, Anvil and Hardhat, run one EVM chain on a developer's machine and support time fast-forward [8, 10]. Running two instances gives two chains, and a relay process can be written to carry messages between them, so far without difficulty. The difficulty is the fast-forward. Holding the ratio of block time to delay fixed while compressing a 24-minute delay into a few seconds of wall-clock time needs a two-hundredfold speedup, which leaves 60 milliseconds to mine each 12-second block. At that rate the two chains and the off-chain actors fall to the operating-system scheduler, so events land in a non-deterministic order, the chains drift out of sync, and the Python off-chain actors miss their slots and crash.

These shortcomings identify four things the testbed must do.

Real bytecode. The contracts are Solidity, and how they behave depends on the EVM that runs them. A failed call reverts and undoes its writes, a transaction can use only one block's worth of gas, and a token balance moves only through that token's own contract (§2.1). Rewriting the contracts in another language, say a Python model, would lose these and test something other than the contract, so the testbed runs the deployed bytecode itself.

More than one chain. The setting of §3.2.2 is two chains that share no state and no atomic transaction, one delivery delay apart. One chain cannot stand in for two, so the testbed runs at least two, with a relay carrying messages between them.

A delay it can fast-forward. What matters is how many blocks fit inside one delay, $N_{\text{Mars}}(\tau) = \lfloor \tau / \delta_M \rfloor$ blocks per message (§3.2.2), about 120 at an Earth-to-Mars delay, and a run spans many such gaps. Sitting through them in real time is infeasible, and shrinking the delay would change the system, so the testbed runs through the empty blocks fast while holding that ratio fixed.

One clock over chains and off-chain actors. The trader, the relay, and any monitor act between blocks, and when each acts decides which block its action lands in. One clock has to order the chains and these off-chain actors together. Left to the operating system, the order of two due events depends on its scheduler, and the same script puts a trade in a different block on the next run.

None of the existing tools satisfies all four. StateLab does.

6.1.2 Implementation

StateLab is one Python process that owns a single clock, a plain integer counter in the chains' own time unit, and advances every chain and off-chain actor from it. Figure 6.1 puts the clock at the centre, the chains with their mining switched off on one side, and the off-chain trader, monitor, and relay on the other, each driven by the clock and ordered by the same process.

Real bytecode. Each chain is an ordinary Anvil node [8], the local Ethereum node Foundry ships, so the contracts run exactly as deployed, with the same storage, gas, reverts, and logs (§6.1.1). StateLab writes no copy of the contract. It loads the compiled interface as a Python object whose methods call into the deployed code, so the logic that runs is the contract's own.

Fast-forwarded time. A node mines a block only when StateLab asks (§6.1.1), so time moves only when the loop steps it. Each step jumps the clock to the next block due on any chain, mines it, and lets each off-chain actor act in a fixed order (Algorithm 6.1). The

$N_{\text{Mars}}(\tau) = \lfloor \tau/\delta_M \rfloor$ blocks that separate two cross-chain events (§3.2.2), 120 at $\tau = 24$ minutes and $\delta_M = 12$ seconds, are mined one after another, so a run crosses the delay in the time it takes to mine the empty blocks. Skipping the idle wait this way runs the experiment 150 to 250 times faster than real time, so a scenario that would take hours of wall-clock settles in seconds.

The schedule. Because one process orders every block, every action, and every message, a run is one fixed sequence of steps, each a clock time, the chain or off-chain actor that acts, and the action it takes. A seed fixes every random draw, every block timestamp comes from the one clock, and the EVM runs each transaction deterministically (§2.1), so the same seed reproduces the run step for step. That sequence is the full record of a run, and every number in this chapter is read from it.

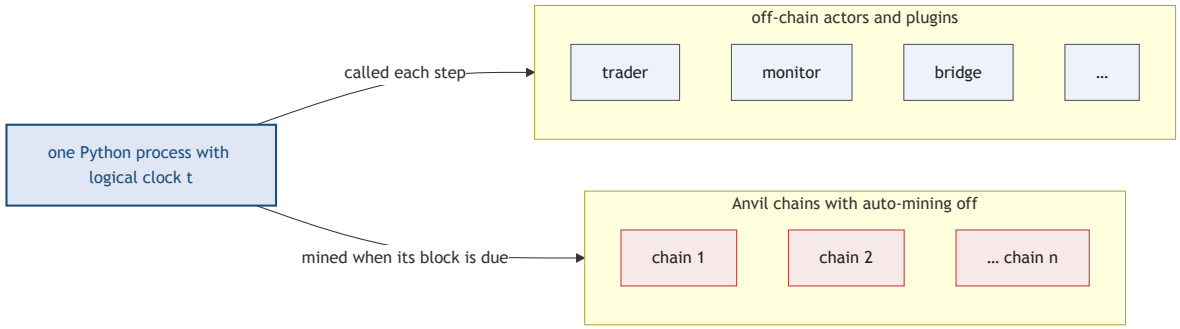


Figure 6.1: StateLab architecture, one Python process driving a real Anvil node per chain under a single shared clock.

Algorithm 6.1 The stepped main loop, advancing the shared clock one scheduled step at a time, deterministic and faster than real time.

```
clock := 0
schedule each chain's first block
while clock < end:
    clock := earliest due block time across the chains
    mine each chain whose block is due now
    let each off-chain actor act, in a fixed order
```

6.1.3 Open Source Release

The engine leaves the number of chains, the set of off-chain actors, and the contracts to the experiment. It is not tied to lane options or this specific two-chain setup. The tokens, the AMM, the bridge, and the lane options of Chapter 4 are each a plugin, and a new study writes its own without changing the engine. Each plugin does two things, defining what it runs on every clock step and handling its own setup, including deploying any contracts it needs. An experiment is a Python script that composes plugins, fixes a seed, and drives the deployed contracts through Solidity-to-Python bindings, giving full

programmatic control over every on-chain and off-chain action and so a deterministic way to test the contracts. The release ships built-in plugins for ERC-20 tokens [17], a simple AMM [11], a cross-chain bridge [34], and the lane options of Chapter 4. This makes StateLab a general-purpose framework for deterministic testing of EVM contracts and the off-chain actors around them, released as an open-source Python package under the MIT licence at `https://github.com/Unoptimized5/statelab`.

6.2 Setup

The setup fixes the scenario and the price process once, then each experiment takes the lag or cost it measures and reports where a lane overtakes the baseline and where it does not.

6.2.1 Testbed and trader

The testbed is the two-chain setting of §3.2.2, built with StateLab (§6.1). A constant-product pool lives on Mars, the trader sits a relay delay τ away on Earth, and a relay carries every cross-chain message one delay later (Figure 6.2). The three contracts under test are the baseline (§2.2), the AMM-side lane (§5.2), and the portable lane (§5.3), each run as deployed bytecode against the same pool, trader, and workload, so only the contract changes. The relay delay τ is the link's one-way latency, fixed within a run but swept across runs, the axis the price-lag experiment varies. The other experiments vary trade size, trade count, split fraction, and premium.

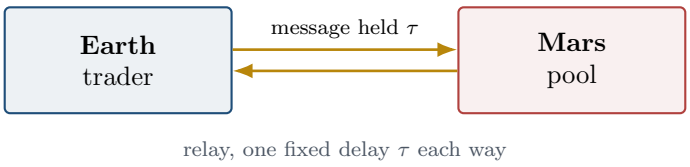


Figure 6.2: The testbed, an Earth trader and a Mars pool joined by a relay that carries each message after one fixed delay τ.

6.2.2 Price process

A pool has no price of its own beyond its reserves, so each run draws a market price for it to track, a driftless geometric Brownian motion [21]. Over one block of length Δt years (the 12-second block over a year) the price P_b moves from P_{b-1} to

$$P_b \;=\; P_{b-1}\exp\!\Big(-\tfrac{1}{2}\sigma^2\,\Delta t \;+\; \sigma\sqrt{\Delta t}\,Z_b\Big), \tag{6.1}$$

with Z_b a fresh standard normal draw each block and σ the volatility, the one knob separating the four pairs of Table 6.1. Each block this path sets the target the order flow drives the baseline pool to, applied alike to all three designs.

Regime	Modelled pair	Effective volatility σ
Stable	USDC-USDT	2.0
Major	USDC-BTC	13.0
Volatile	USDC-ETH	26.0
Meme	USDC-PEPE	66.0

Table 6.1: The four volatility regimes, named after the pair each models. The σ values are synthetic and time-compressed, not annualised estimates. The drift is zero throughout.

6.2.3 Monte Carlo

A single run is one price path and gives a noisy figure, so each operating point is run over many independent seeds and the metric reported as the mean over them with its standard error [48]. StateLab makes a run deterministic in its seed (§6.1.2), so the averages are reproducible and the same seeds drive the three contracts at each point for a paired comparison. The capital and execution results use sixteen seeds, the price-lag and split-drag results eight. The capital result of §6.3.3 fixes the one-way delay at a mid-range $\tau = 12$ minutes.

6.3 Experiments

6.3.1 Price lag

The price lag of §3.3.1 is the gap between the price an order is quoted at when it is decided and the price it is given when it executes one relay delay τ later. Let P_b be the pool price at block b and $P_{b-\tau}$ its price τ blocks earlier, the price the order was quoted against. The relative move over that window, $|P_b - P_{b-\tau}|/P_{b-\tau}$, is the fraction of its value an order loses to the delay, and the divergence is its mean over a run of B blocks,

$$D(\tau) \;=\; \frac{1}{B-\tau} \sum_{b=\tau+1}^{B} \frac{|P_b - P_{b-\tau}|}{P_{b-\tau}}. \tag{6.2}$$

It reads as a percent of price, so it sits on the same axis as the premium a holder pays to escape it. We sweep τ over a wide range, well past the delay a lane would usually face, to trace how the divergence grows with it.

The baseline carries the full D and both lanes read zero (Figure 6.3, Table 6.2). D grows with the delay and with volatility, from half a percent on the stable pair to two-thirds of price on the meme pair at the longest delay. Against it runs the premium, a flat one-hundredth of a percent of the slice per block, which over the window climbs from a fifth of a percent to over three percent across the sweep.

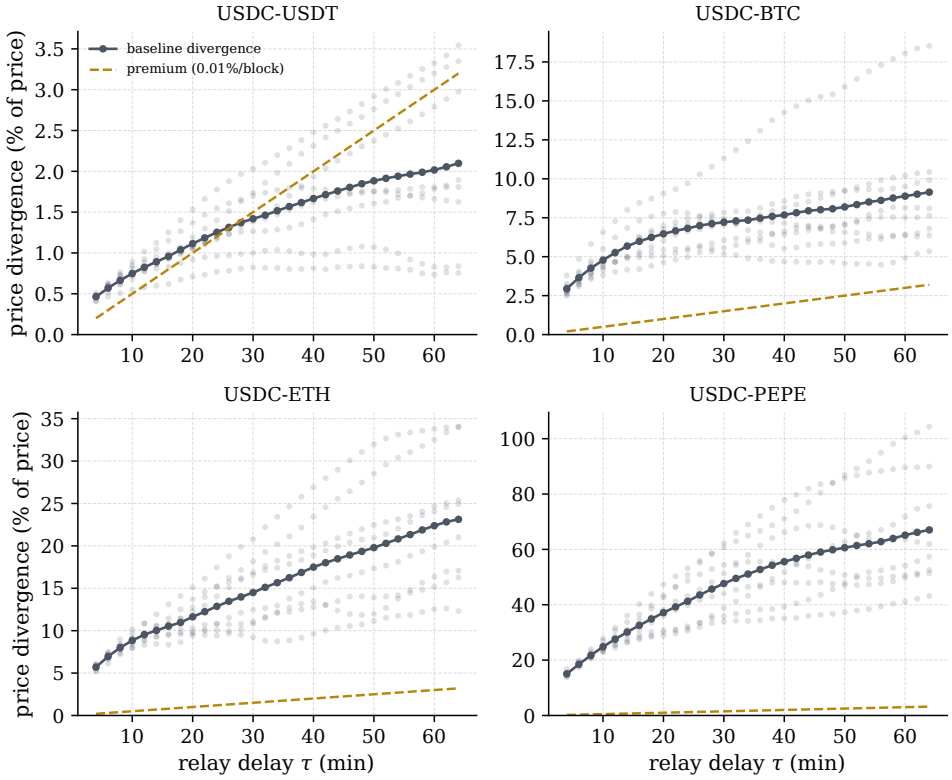


Figure 6.3: The price divergence D, the mean relative price move over the delay as a percent of price, against the relay delay τ, one panel per pair, with the flat per-block premium accrued over the same window (dashed). The baseline is the mean over the seeds, each per-seed run a faint point. The lane's own divergence is zero and is omitted.

Pair	τ=4	τ=8	τ=16	τ=32	τ=64
USDC-USDT	0.46 ± 0.01	0.66 ± 0.03	0.96 ± 0.06	1.46 ± 0.14	2.10 ± 0.36
USDC-BTC	2.94 ± 0.14	4.27 ± 0.24	5.98 ± 0.46	7.29 ± 0.68	9.14 ± 1.38
USDC-ETH	5.70 ± 0.10	8.00 ± 0.19	10.53 ± 0.47	15.11 ± 1.52	23.13 ± 2.66
USDC-PEPE	15.03 ± 0.29	21.77 ± 0.38	32.55 ± 1.47	49.52 ± 3.93	67.04 ± 6.99

Table 6.2: Baseline price divergence D (mean $\pm$ standard error, $n = 8$) at five relay delays τ in minutes, as a percent of price. The lane reads zero at every cell, so only the baseline is tabulated.

6.3.2 Execution lag

The execution lag of §3.3.2 is the time to fill one trade and hear back. A holder who reacts to each fill before placing the next is paced by that round trip, so over a horizon H it places

$$M(H) = \left\lfloor \frac{H}{t_{\mathrm{rt}}} \right\rfloor \quad \text{trades}, \qquad t_{\mathrm{rt}} = 2\beta\tau + 2\delta, \tag{6.3}$$

where $M(H)$ is the number of trades the horizon allows, H is the horizon, two hours here, and t_{rt} is the time one trade takes to send and confirm. That round trip is a relay crossing each way, 2τ at the relay delay τ of §3.2.2, plus two local blocks 2δ at the block time δ of Definition 3.3. The switch β is one when the fill is a relay away, the baseline and AMM-side lane, and zero when it is local, the portable lane.

Over a two-hour horizon (§6.2) the portable lane and the remote designs split sharply (Figure 6.4, Table 6.3). The portable lane reacts in a single local round trip at every delay, so a reactive holder completes about three hundred trades. The baseline and the AMM-side lane both sit a relay away, so each reaction waits a full round trip and the count falls from fourteen trades at a four-minute relay to a single trade once the delay passes half an hour.

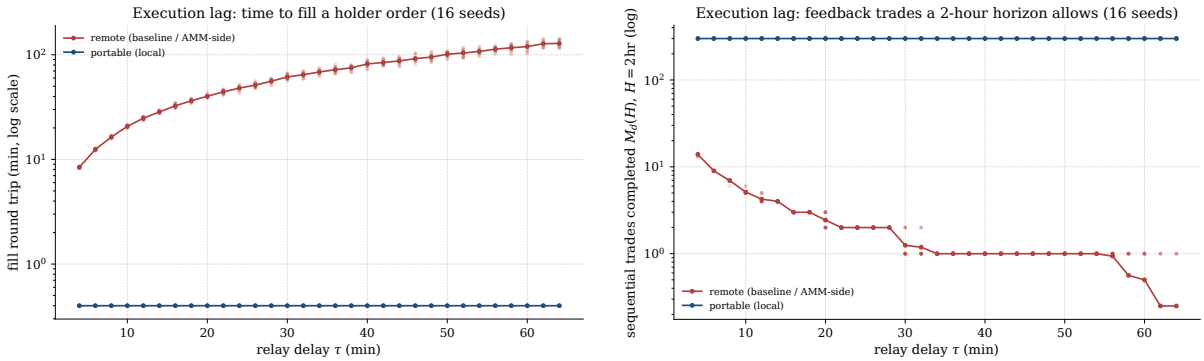


Figure 6.4: Fill latency by design across the relay delay τ (left, log scale), and the sequential feedback trades $M(H)$ a two-hour horizon allows (right). The sixteen per-seed runs are faint scatter around each mean line. The portable lane stays flat at one block while the remote designs track the delay.

Relay delay τ	**Remote** $M(H)$	**Portable** $M(H)$
4 min	13.88 ± 0.08	300 ± 0
12 min	4.25 ± 0.11	300 ± 0
24 min	2.00 ± 0.00	300 ± 0
50 min	1.00 ± 0.00	300 ± 0

Table 6.3: Sequential feedback trades $M(H)$ a two-hour horizon allows, by relay delay, mean $\pm$ standard error over sixteen seeds that jitter each relay hop by up to $\pm 15\%$. The baseline and AMM-side lane share the remote round trip (one "remote" column), the portable lane fills locally. The closed form of Eq. (6.3) is the expectation the seeds scatter around.

6.3.3 Capital in transit

The capital lag of §3.3.3 idles funds while they cross the relay. A crossing of value K is locked for the round trip 2τ, costing $K \cdot 2\tau$ dollar-days (§3.3.3). The capital in transit C is that cost summed over every crossing an episode makes,

$$C \;=\; 2\tau \sum K. \tag{6.4}$$

The baseline and the AMM-side lane cross the value of each trade, so over an episode their crossings total the turnover θ of the pool, the traded volume as a fraction of the pool's value. The portable lane crosses its slice once, so its crossings total the carve c of the pool. Both scale with the pool's value, which drifts as the price moves [13], so each

reported figure is a mean over the seeds.

Sweeping the turnover θ from zero to twice the pool at $\tau = 12$ minutes, with a one-million-dollar pool and a $c = 10\%$ slice (§6.2), gives Figure 6.5 and Table 6.4. The three meet at the crossover $\theta = c$, the portable lane locking less above it and the gap widening with every further round trip.

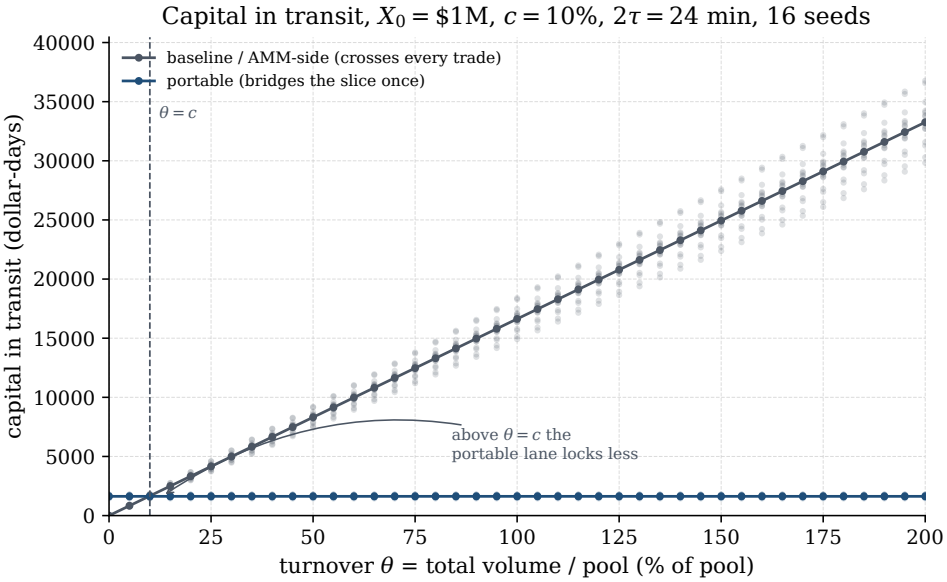


Figure 6.5: Capital in transit against turnover. The baseline and AMM-side lane coincide on the rising line, the portable lane is flat, and the three cross at $\theta = c$, the per-seed runs faint scatter around each mean.

Turnover θ	**Baseline / AMM-side**	**Portable**
10%	$1{,}663 \pm 23$	$1{,}628 \pm 23$
25%	$4{,}157 \pm 59$	$1{,}628 \pm 23$
50%	$8{,}315 \pm 117$	$1{,}628 \pm 23$
100%	$16{,}629 \pm 235$	$1{,}628 \pm 23$
200%	$33{,}259 \pm 469$	$1{,}628 \pm 23$

Table 6.4: Capital in transit over an episode (dollar-days), mean ± standard error over sixteen seeds, for a one-million-dollar pool with a $c = 10\%$ slice at $\tau = 12$ minutes. The baseline and AMM-side lane rise with turnover θ and coincide, the portable lane flat at the slice round trip. The two are level at the crossover $\theta = c = 10\%$ and the portable lane cheaper above it.

6.3.4 Split drag

A lane holds only the fraction c of the reserves, so it sits on a shallower curve than the full pool and a holder swap slips further on it (Proposition 4.1). The split-drag metric is that extra slippage,

$$s_{\text{drag}} \;=\; s_{\text{lane}} - s_{\text{full}}, \tag{6.5}$$

the slippage s_{lane} a swap pays on the lane over the slippage s_{full} it pays on the full pool, both the price-side cost of §2.2. Proposition 4.1 gives them in the reserves, $s_{\text{full}} = \Delta x/(X+\Delta x)$ on the full pool and $s_{\text{lane}} = \Delta x/(cX+\Delta x)$ on the lane, for an input Δx against a reserve X.

Firing the same trade on the full pool and on a lane of it, varying the split fraction c from five to ninety-five percent across the stable and major regimes, volatilities of two and thirteen (§6.2), gives Figure 6.6 and Table 6.5. The measured drag is the exact split impact of Eq. (4.8), which its first-order limit $s_{\text{full}}(1/c-1)$ overstates the more the slice thins, the same on both pairs despite their differing volatility, near fifteen percent of price on a five-percent slice, about one percent on a half-pool slice, and almost nothing once the slice is most of the pool. Set against the divergence each pair saves over a twelve-minute delay, the net price saving crosses zero at a break-even split that falls with volatility, near fifty-three percent of the pool for the stable pair and fifteen for bitcoin (Figure 6.6, right), nine for the more volatile ether, while the meme pair saves across the whole sweep, its break-even near three percent falling below the thinnest slice shown.

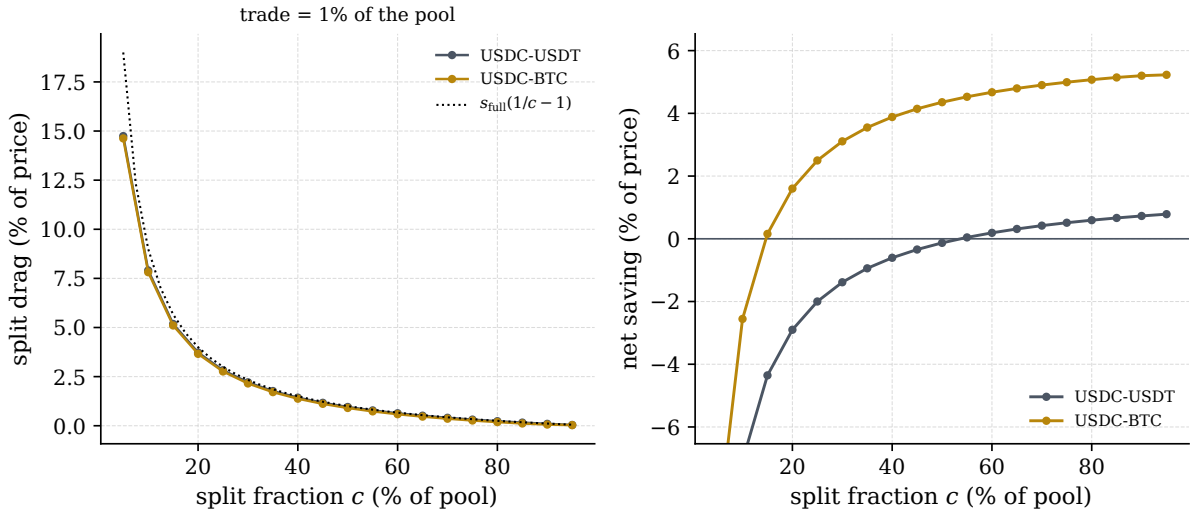


Figure 6.6: The split drag against the split fraction c (left), the stable and major regimes overlying on one curve, with the first-order law $s_{\text{full}}(1/c-1)$ for reference, and the net price saving (right), the divergence removed minus the drag before any premium, each pair crossing zero at its break-even split.

Split fraction c	**Split drag** s_{drag}
0.05	14.68 ± 0.05
0.10	7.86 ± 0.04
0.25	2.79 ± 0.03
0.50	0.93 ± 0.03
0.90	0.08 ± 0.02

Table 6.5: Split drag s_{drag} (percent of price, mean $\pm$ standard error, $n = 4$) against the split fraction c, pooled over the stable and major regimes and two seeds each, the two volatilities confirming the drag turns on the split, not volatility.

6.3.5 Optimal lane size

A holder rents the slice c once for a fixed lease and works it with many small trades, so its cost reads per unit of volume. A trade that builds a position one way pays the split drag of §6.3.4, $s_{\text{full}}(1/c - 1)$ at the small per-trade slippage s_{full}, while the premium and the capital carry spread over the directional turnover θ, so the cost is

$$s_{\text{full}}\left(\frac{1}{c} - 1\right) \;+\; (p + 2\tau r)\frac{c}{\theta}, \tag{6.6}$$

with p the total premium paid on the leased slice over the fixed lease and r the cost of capital of §3.3.3. The split drag is a one-way cost. A holder who round-trips the slice recovers it, since a fee-free reversal returns the pool, and pays only the premium and the capital while still escaping the price and capital lags, so for that holder the thinnest workable slice is cheapest. For the directional holder the two terms pull against each other and the cost traces a U in c (Figure 6.7), least at the slice $\sqrt{s_{\text{full}}\,\theta/(p + 2\tau r)}$, which grows with the turnover as $\sqrt{\theta}$ and shrinks as the premium rises, sliding from a twenty-one-percent slice at a one-percent premium to an eight-percent slice at an eight-percent premium. Capital in transit holds near the axis, about 7.7% of the cost at a two-percent premium.

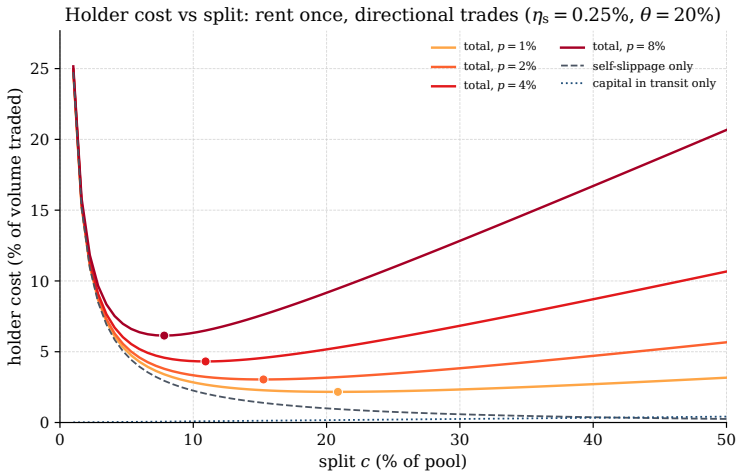


Figure 6.7: Holder cost per unit of volume against split fraction for four premiums at a per-trade slippage $s_{\text{full}} = 0.25\%$ and turnover $\theta = 20\%$, each a U with an optimum sliding left as the premium rises and capital in transit hugging zero.

6.4 Discussion

The three lags compose into one choice that turns on five levers, with three further conditions that scale the result without moving the boundary between the simple AMM, the AMM-side lane, and the portable lane.

Volatility sets how much there is to clear. A stable pair barely drifts, so the baseline serves it and a lane saves too little to bother, while a volatile or meme pair diverges by

tens of percent over the relay window, so a lane that fills against an untouchable slice earns its keep. Where volatility is low the question ends at the baseline.

Relay delay sets how long the lag runs. A short link leaves little for a lane to clear and favours the baseline, while a long link makes a lane worthwhile on price and, because the execution lag it opens is the cross-chain round trip the portable lane settles at home, tilts the placement toward the portable lane.

Split fraction trades drag against cost. A thin slice slips more, near sixteen percent of price on a five-percent slice, while a fat slice locks proportionally more capital and premium, so an optimal split sits where the slippage saved by widening stops covering the premium that widening costs (§6.3.5), sliding smaller as the premium rises and wider with turnover.

Turnover sets how the capital cost accrues. The baseline and the AMM-side lane cross the relay on every trade, so their capital in transit grows without bound, while the portable lane bridges its slice once and stays flat. The three are level when turnover equals the split, and past it the portable lane locks less.

Workload sets which lag the holder feels. A holder who reacts to each fill is gated by the round trip, the execution lag the portable lane settles in one local block, so a reactive workload favours the portable lane heavily, while a holder firing independent trades does as well on the AMM-side lane and spares the bridge.

Trade size, the lease term, and pool size scale the figures without moving the boundary, since the analysis prices small repeated trades, the lease runs about twice the relay delay plus the holder's decision time, and a deeper pool eases the slippage while raising the standing cost of the slice.

The five levers settle into a three-way recommendation. The simple AMM suits a small lag, where a lane saves less than it costs. The AMM-side lane suits a holder escaping the price lag on modest independent volume who does not need the bridge. The portable lane suits a heavy or reactive trader, whose turnover outruns the split or who is gated by the round trip, because it clears the execution and capital lags the AMM-side lane leaves in place.

Costs. Each gain is paid for. The premium buys the lane, and while it is open the holder trades a thinner public pool, the split drag of §6.3.4. The portable placement adds the capital cost of bridging the slice out and back (§6.3.3), and the trust the bridge rests on (§5.4), which the AMM-side placement avoids.

Chapter 7

Conclusion

Trading an automated market maker across chains fails for a structural reason that careful execution cannot remove. The three things a trader depends on are each separated from them by propagation delay. Observation is delayed, because the pool's price is read across the relay and is stale on arrival. Execution is delayed, because filling an order costs a full crossing each way, so a trader reacting to each fill is beaten to the next by a local one who settles in a block. Capital release is delayed, because funds committed to a remote venue are stranded for a full round trip before redeployment. This dissertation named these fundamental frictions the price, execution, and capital lags, and took them as a problem to engineer against rather than price around.

Lane options address the delay not by making the relay faster but by changing what the trader depends on. A holder splits a fixed fraction of a remote constant-product pool into a private sub-pool called a lane, reserves it for a lease term, and trades that slice instead of the public pool [11]. The mechanism comes in two placements. An AMM-side lane leaves the slice on the pool's own chain, where the holder trades it across the relay but free of everyone else's order flow. A portable lane bridges the slice to the holder's own chain and is traded locally. In both cases the holder trades liquidity they control on terms fixed at the open, and the slice merges back into the public pool at expiry.

What was built to support this is deliberately small, passive gas-limited EVM contracts with no scheduler or keeper and every operation bounded by the gas limit [2, 6]. The liquidity provider posts the premium as a fixed rate the holder pays upfront, not a charge skimmed from volume. The cross-chain behaviour rests on a single assumption, a relay that carries each message across the delay and is trusted to do so, the cost of that trust made explicit in the threat model [34, 36]. The evaluation ran on StateLab, a single-process framework that drives one or more real EVM chains and the off-chain actors around them from a single logical clock, skips the idle delay between blocks, and replays every run deterministically from a seed. The engine fixes neither the number of chains nor the contracts under test, so the same framework serves any time-dependent or cross-chain

contract study. The framework and contracts are released, and reproducing a measured table needs its run configuration, seeds, and pinned tool versions.

The evaluation separated the three lags and found the two placements clear different lags. An AMM-side lane spares the holder the price lag carried by public order flow, the dominant cost on a volatile pair, but does nothing for execution. A fill still costs a relay crossing, so a holder reacting to each fill completes one trade once the delay passes half an hour, against the hundreds a local venue completes. The portable lane removes that execution lag by filling on the holder's chain in one block, and amortises capital by bridging its slice once rather than on every trade. Neither is free. Both pay split drag, the extra slippage of a thin slice, which shrinks as the split widens, and both pay the premium. A lane is worth opening only where the lag it clears outweighs the drag it adds, the AMM-side placement serving a light independent trader and the portable placement clearing the execution and capital lags.

Future work

A lane inherits the LP risk of the underlying AMM, including impermanent loss [20], and the per-block rate captures the opportunity cost of leased liquidity. Pricing the rate to also reflect the pair's volatility and the fraction of the pool already leased is an open problem, as these are the factors that determine the option value of the public fallback the holder keeps [21, 22].

It would also be interesting to compare lane options against other approaches to MEV reduction and fairer execution, such as introducing a random delay on every public trade to reduce the timing advantage of co-located actors [18, 28], or batch clearing, where orders accumulate over a window and clear at a uniform price [25, 27].

The contracts were evaluated on common, benign order flow. Studying their behaviour under adversarial conditions, including simultaneous lane opening by many holders, bursty contact windows, and flow that deliberately trades against an open lane, is a natural next step. A pen-testing and formal-verification pass [31, 32] would give stronger production guarantees than the unit test suite alone.

Carrying the design to a platform where the block gas limit is a softer constraint, whether through more expressive execution environments or layer-2 settlement, would allow the implementation to drop several of the engineering trade-offs the EVM forces and may open the door to richer on-chain premium computation.

A lane could also serve as a composability primitive. A smart contract that opens a lane acquires a private slice of liquidity it fully controls for the lease term, and within that slice could run its own pricing logic or act as the pool for a secondary AMM [14]. This would let liquidity be rented from a well-supplied pool and redeployed on a thinner market in a trustless, on-chain way, with the original pool recovering its full reserves at expiry.

Bibliography

[1] Buterin, V.: A Next-Generation Smart Contract and Decentralized Application Platform. White paper (2014). Available at: https://ethereum.org/whitepaper/

[2] Wood, G.: Ethereum: A Secure Decentralised Generalised Transaction Ledger. Ethereum Project Yellow Paper (2014, Shanghai version 2025). Available at: https://ethereum.github.io/yellowpaper/paper.pdf

[3] Herlihy, M.: Blockchains from a Distributed Computing Perspective. Communications of the ACM **62**(2), 78–85 (2019). Available at: https://doi.org/10.1145/3209623

[4] Decker, C., Wattenhofer, R.: Information Propagation in the Bitcoin Network. In: IEEE Thirteenth International Conference on Peer-to-Peer Computing, pp. 1–10 (2013). Available at: https://doi.org/10.1109/P2P.2013.6688704

[5] Gervais, A., Karame, G.O., Wüst, K., Glykantzis, V., Ritzdorf, H., Capkun, S.: On the Security and Performance of Proof of Work Blockchains. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 3–16 (2016). Available at: https://doi.org/10.1145/2976749.2978341

[6] Antonopoulos, A.M., Wood, G., Parisi, C., Mazza, A., Pozzolini, N.: Mastering Ethereum: Implementing Smart Contracts. 2nd edn. O'Reilly Media, Santa Rosa, CA (2025). Available at: https://www.oreilly.com/library/view/mastering-ethereum-2nd/9781098168414/

[7] Bartoletti, M., Benetollo, L., Bugliesi, M., Crafa, S., Dal Sasso, G., Pettinau, R., Pinna, A., Piras, M., Rossi, S., Salis, S., Spanò, A., Tkachenko, V., Tonelli, R., Zunino, R.: Smart contract languages: A comparative analysis. Future Generation Computer Systems **164**, art. 107563 (2025). Available at: https://doi.org/10.1016/j.future.2024.107563

[8] Foundry: anvil. Official documentation (accessed 9 June 2026). Available at: https://www.getfoundry.sh/reference/anvil/anvil

[9] Foundry: forge. Official documentation (accessed 9 June 2026). Available at: https://www.getfoundry.sh/forge/

[10] Hardhat: Hardhat 3. Official documentation (accessed 9 June 2026). Available at: https://hardhat.org/docs/getting-started

[11] Adams, H., Zinsmeister, N., Robinson, D.: Uniswap v2 Core. Technical whitepaper (2020). Available at: https://app.uniswap.org/whitepaper.pdf

[12] Adams, H., Salem, M., Zinsmeister, N., Reynolds, S., Adams, A., Pote, W., Toda, M., Henshaw, A., Williams, E., Robinson, D.: Uniswap v4 Core. White paper (2024). Available at: https://app.uniswap.org/whitepaper-v4.pdf

[13] Angeris, G., Kao, H.-T., Chiang, R., Noyes, C., Chitra, T.: An Analysis of Uniswap Markets. Cryptoeconomic Systems **1**(1) (2020). Available at: https://doi.org/10.21428/58320208.c9738e64

[14] Xu, J., Paruch, K., Cousaert, S., Feng, Y.: SoK: Decentralized Exchanges (DEX) with Automated Market Maker (AMM) Protocols. ACM Computing Surveys **55**(11), 1–50 (2023). Available at: https://doi.org/10.1145/3570639

[15] Brönnimann, W., Egloff, P., Krabichler, T.: Automated market makers and their implications for liquidity providers. Digital Finance **6**(3), 573–604 (2024). Available at: https://doi.org/10.1007/s42521-024-00117-0

[16] Werner, S.M., Perez, D., Gudgeon, L., Klages-Mundt, A., Harz, D., Knottenbelt, W.J.: SoK: Decentralized Finance (DeFi). In: 4th ACM Conference on Advances in Financial Technologies (AFT 2022), pp. 30–46 (2022). Available at: https://doi.org/10.1145/3558535.3559780

[17] Vogelsteller, F., Buterin, V.: ERC-20: Token Standard. Ethereum Improvement Proposals, no. 20 (2015). Available at: https://eips.ethereum.org/EIPS/eip-20

[18] Daian, P., Goldfeder, S., Kell, T., Li, Y., Zhao, X., Bentov, I., Breidenbach, L., Juels, A.: Flash Boys 2.0: Frontrunning in Decentralized Exchanges, Miner Extractable Value, and Consensus Instability. In: 2020 IEEE Symposium on Security and Privacy, pp. 910–927 (2020). Available at: https://doi.org/10.1109/SP40000.2020.00040

[19] Milionis, J., Moallemi, C.C., Roughgarden, T., Zhang, A.L.: Automated Market Making and Loss-Versus-Rebalancing. arXiv:2208.06046 (2022, revised 2024). Available at: https://doi.org/10.48550/arXiv.2208.06046

[20] Milionis, J., Moallemi, C.C., Roughgarden, T., Zhang, A.L.: Quantifying Loss in Automated Market Makers. In: Proceedings of the 2022 ACM CCS Workshop on Decentralized Finance and Security, pp. 71–74 (2022). Available at: https://doi.org/10.1145/3560832.3563441

[21] Hull, J.C.: Options, Futures, and Other Derivatives. 11th edn. Pearson (2022).

[22] Black, F., Scholes, M.: The Pricing of Options and Corporate Liabilities. Journal of Political Economy **81**(3), 637–654 (1973). Available at: https://doi.org/10.1086/260062

[23] Milionis, J., Moallemi, C.C., Roughgarden, T.: Automated Market Making and Arbitrage Profits in the Presence of Fees. In: Financial Cryptography and Data Security, LNCS 14745, pp. 159–171 (2025). Available at: https://doi.org/10.1007/978-3-031-78676-1_9

[24] Aquilina, M., Budish, E., O'Neill, P.: Quantifying the High-Frequency Trading Arms Race. The Quarterly Journal of Economics **137**(1), 493–564 (2022). Available at: https://doi.org/10.1093/qje/qjab032

[25] Budish, E., Cramton, P., Shim, J.: The High-Frequency Trading Arms Race: Frequent Batch Auctions as a Market Design Response. The Quarterly Journal of Economics **130**(4), 1547–1621 (2015). Available at: https://doi.org/10.1093/qje/qjv027

[26] Glosten, L.R., Milgrom, P.R.: Bid, Ask and Transaction Prices in a Specialist Market with Heterogeneously Informed Traders. Journal of Financial Economics **14**(1), 71–100 (1985). Available at: https://doi.org/10.1016/0304-405X(85)90044-3

[27] Canidio, A., Fritsch, R.: Batching Trades on Automated Market Makers. In: 5th Conference on Advances in Financial Technologies (AFT 2023). Leibniz International Proceedings in Informatics (LIPIcs), vol. 282, art. 24 (2023). Available at: https://doi.org/10.4230/LIPIcs.AFT.2023.24

[28] Yan, T., Tessone, C.J.: Sandwich attacks. In: Elgar Encyclopedia of Cryptocurrencies, Blockchain and DLT, pp. 195–197. Edward Elgar Publishing (2026). Available at: https://doi.org/10.4337/9781035339969.00045

[29] Zhou, L., Qin, K., Ferreira Torres, C., Le, D.V., Gervais, A.: High-Frequency Trading on Decentralized On-Chain Exchanges. In: 2021 IEEE Symposium on Security and Privacy, pp. 428–445 (2021). Available at: https://doi.org/10.1109/SP40001.2021.00027

[30] Solidity: Error handling: Assert, Require, Revert and Exceptions. Solidity documentation (accessed 9 June 2026). Available at: https://docs.soliditylang.org/en/latest/control-structures.html#error-handling-assert-require-revert-and-exceptions

[31] Atzei, N., Bartoletti, M., Cimoli, T.: A Survey of Attacks on Ethereum Smart Contracts (SoK). In: Principles of Security and Trust, LNCS 10204, pp. 164–186 (2017). Available at: https://doi.org/10.1007/978-3-662-54455-6_8

[32] Luu, L., Chu, D.-H., Olickel, H., Saxena, P., Hobor, A.: Making Smart Contracts Smarter. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 254–269 (2016). Available at: https://doi.org/10.1145/2976749.2978309

[33] Belchior, R., Vasconcelos, A., Guerreiro, S., Correia, M.: A Survey on Blockchain Interoperability: Past, Present, and Future Trends. ACM Computing Surveys **54**(8), 1–41 (2021). Available at: https://doi.org/10.1145/3471140

[34] Zamyatin, A., Al-Bassam, M., Zindros, D., Kokoris-Kogias, L., Moreno-Sanchez, P., Kiayias, A., Knottenbelt, W.J.: SoK: Communication Across Distributed Ledgers. In: Financial Cryptography and Data Security, LNCS 12675, pp. 3–36 (2021). Available at: https://doi.org/10.1007/978-3-662-64331-0_1

[35] Augusto, A., Belchior, R., Correia, M., Vasconcelos, A., Zhang, L., Hardjono, T.: SoK: Security and Privacy of Blockchain Interoperability. In: 2024 IEEE Symposium on Security and Privacy (SP), pp. 3840–3865 (2024). Available at: https://doi.org/10.1109/SP54263.2024.00255

[36] Lee, S.-S., Murashkin, A., Derka, M., Gorzny, J.: SoK: Not Quite Water Under the Bridge: Review of Cross-Chain Bridge Hacks. In: 2023 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), pp. 1–14 (2023). Available at: https://doi.org/10.1109/ICBC56567.2023.10174993

[37] Xie, T., Zhang, J., Cheng, Z., Zhang, F., Zhang, Y., Jia, Y., Boneh, D., Song, D.: zkBridge: Trustless Cross-Chain Bridges Made Practical. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, pp. 3003–3017 (2022). Available at: https://doi.org/10.1145/3548606.3560652

[38] Gogol, K.M., Schneider, M., Tessone, C.J., Livshits, B.: Liquidity Fragmentation or Optimization? Analyzing Automated Market Makers Across Ethereum and Rollups. In: Financial Cryptography and Data Security. FC 2025 International Workshops, LNCS 15753, pp. 94–110 (2026). Available at: https://doi.org/10.1007/978-3-032-00492-5_7

[39] Fall, K.: A Delay-Tolerant Network Architecture for Challenged Internets. In: Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, pp. 27–34 (2003). Available at: https://doi.org/10.1145/863955.863960

[40] Burleigh, S., Hooke, A., Torgerson, L., Fall, K., Cerf, V., Durst, B., Scott, K., Weiss, H.: Delay-Tolerant Networking: An Approach to Interplanetary Internet. IEEE Communications Magazine **41**(6), 128–136 (2003). Available at: https://doi.org/10.1109/MCOM.2003.1204759

[41] Morabito, D.D., Hastrup, R.: Communicating with Mars during periods of solar conjunction. In: 2002 IEEE Aerospace Conference Proceedings, vol. 3, pp. 3-1271–3-1281 (2002). Available at: https://doi.org/10.1109/AERO.2002.1035260

[42] Jesick, M.: Mars Trojan Orbits for Continuous Earth-Mars Communication. The Journal of the Astronautical Sciences **67**, 902–931 (2020). Available at: https://doi.org/10.1007/s40295-019-00195-y

[43] Hu, Y., Manzoor, A., Ekparinya, P., Liyanage, M., Thilakarathna, K., Jourjon, G., Seneviratne, A.: A Delay-Tolerant Payment Scheme Based on the Ethereum Blockchain. IEEE Access **7**, 33159–33172 (2019). Available at: https://doi.org/10.1109/ACCESS.2019.2903271

[44] Zi, L., Cong, X.: A blockchain transaction mechanism in the delay tolerant network. Journal of Network and Computer Applications **231**, 103998 (2024). Available at: https://doi.org/10.1016/j.jnca.2024.103998

[45] Schauer, K.: Space Communications: 7 Things You Need to Know. NASA (2020). Available at: https://www.nasa.gov/missions/tech-demonstration/space-communications-7-things-you-need-to-know/

[46] Baldwin, P.: Deep Space Network Scheduling: Principles, Challenges, and Path Forward. NASA NTRS 20250004124 (2025). Available at: https://ntrs.nasa.gov/citations/20250004124

[47] Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. 4th edn. MIT Press (2022).

[48] Glasserman, P.: Monte Carlo Methods in Financial Engineering. Springer (2003). Available at: https://doi.org/10.1007/978-0-387-21617-1

Appendix A

Worked examples

These examples use one fee-free constant-product pool throughout. The Mars pool starts with reserves

$$(X, Y) = (100, 1000), \qquad k = XY = 100{,}000,$$

and the Earth trader wants to swap $\Delta x = 10$ units of X for Y. The output function is

$$Q((X, Y), \Delta x) = \frac{Y \Delta x}{X + \Delta x}. \tag{A.1}$$

The one-way delivery delay is $\tau = 24\,\text{min}$. For local Earth execution, take $\delta_E = 12\,\text{s}$. For the lane examples, the lane splits off $c = 1/2$ of the reserves. This is larger than a normal production lane, but it keeps the arithmetic visible.

A.1 Price lag

Simple AMM. At quote time, Earth sees the Mars pool at $(100, 1000)$. The expected output is

$$Q((100, 1000), 10) = \frac{1000 \cdot 10}{100 + 10} = 90.91. \tag{A.2}$$

During the delivery delay, Mars-local flow moves the pool to $(200, 500)$. The invariant is still $200 \cdot 500 = 100{,}000$, but the delivered output is

$$Q((200, 500), 10) = \frac{500 \cdot 10}{200 + 10} = 23.81. \tag{A.3}$$

The trader expected 90.91 units of Y and receives 23.81. The shortfall is 67.10 units of Y.

AMM-side lane option AMM. An AMM-side lane request also reaches Mars after one delivery delay. If Mars has already moved to $(200, 500)$, the lane is split off from that

state, not from the state Earth saw. With $c = 1/2$, the lane starts as

$$(X^{\text{lane}}, Y^{\text{lane}}) = (100, 250).$$

The holder's first lane quote is therefore

$$Q((100, 250), 10) = \frac{250 \cdot 10}{100 + 10} = 22.73. \tag{A.4}$$

The lane has not recovered the original 90.91 quote. Its benefit begins after the split. If later Mars-local public flow moves the public sub-pool to $(150, 166.67)$, the public output for the same trade falls to

$$Q((150, 166.67), 10) = 10.42.$$

The lane output remains 22.73 until the holder trades against it. The AMM-side lane pays price lag once at split time, then isolates the holder from later public-reserve movement.

Portable lane option AMM. The portable lane is split off from the same Mars state $(200, 500)$, so the initial lane state is again $(100, 250)$. The difference is where that state is active. Mars exports the lane to Earth, and the Earth-side contract executes holder swaps against the exported state. The first local holder trade still quotes

$$Q((100, 250), 10) = 22.73.$$

Mars public flow after export cannot change this Earth-side number. The portable lane therefore has the same initial price lag as the AMM-side lane, but no later Mars-side price lag while the lane is active on Earth.

A.2 Execution lag

Simple AMM. At $t = 0$, Earth submits the swap to Mars. At $t = 24\,\text{min}$, Mars receives it and evaluates the current pool state. If the pool is $(200, 500)$, the executable output is 23.81. If the trader set $minOut = 90$, the swap reverts at Mars after the full delivery delay. If the trader set $minOut \leq 23.81$, the swap fills at Mars after the same delay. In both cases, Earth learns the result only after the return message, at about $t = 48\,\text{min}$.

AMM-side lane option AMM. The AMM-side lane prices a delayed claim, but holder actions still land on Mars. The first acknowledgement-informed lane use has the trace

$$\begin{aligned} t = 0\,\text{min} &\quad \text{Earth sends the lane request,} \\ t = 24\,\text{min} &\quad \text{Mars splits off the lane,} \\ t = 48\,\text{min} &\quad \text{Earth receives confirmation,} \\ t = 72\,\text{min} &\quad \text{Mars receives the first lane swap,} \\ t = 96\,\text{min} &\quad \text{Earth receives the result.} \end{aligned}$$

For the state above, that first lane swap can pay 22.73 if the holder's minimum-output check allows it. The lane removes public-reserve interference after the split, but the Earth-originated execution still waits one τ to reach Mars.

Portable lane option AMM. The portable lane keeps the same reservation round trip but moves the active lane state to Earth:

$$\begin{aligned} t = 0\,\text{min} &\quad \text{Earth sends the lane request,} \\ t = 24\,\text{min} &\quad \text{Mars splits off and exports the lane,} \\ t = 48\,\text{min} &\quad \text{Earth activates the lane,} \\ t = 48\,\text{min}\,12\,\text{s} &\quad \text{the first local holder swap can be included.} \end{aligned}$$

The first holder execution happens about 23 min 48 s earlier than under the AMM-side remote-controlled lane. Later holder swaps are local Earth transactions. They wait for local blocks, not for another Mars delivery delay.

A.3 Capital lag

Let the value of the locked assets be $K = \$100{,}000$, the annual opportunity rate $r = 20\%$, and the one-way delivery delay $\tau = 24\,\text{min}$. Two costs follow from §3.3.3. $F_{\text{execution}} = Kr\tau$ is the cost of holding the assets for one crossing, and $F_{\text{confirmation}} = Kr \cdot 2\tau$ is the cost over the full round trip until the outcome is confirmed and the assets are redeployable:

$$F_{\text{execution}} \;=\; K\,r\,\tau \;=\; \$100{,}000 \times \frac{0.20}{365 \times 24 \times 60\,\text{min/yr}} \times 24\,\text{min} \;=\; \$0.91\,, \tag{A.5}$$

$$F_{\text{confirmation}} \;=\; 2\,K\,r\,\tau \;=\; \$1.83\,. \tag{A.6}$$

Simple AMM. The trade capital K is pledged at submission and unavailable until either the fill is confirmed on the source chain (one round trip) or the refund crosses back (also one round trip). In both cases the capital lag cost is $F_{\text{confirmation}} = \$1.83$.

AMM-side lane. The holder commits the premium Π at the split, and its opportunity cost over the delay is $r\Pi\tau$. Each subsequent swap sends K across the relay, locking it for

a round trip until confirmed, a cost of $F_{\text{confirmation}} = \$1.83$. Over N swaps the capital-lag cost accumulates as in Eq. (4.21), $F_{\text{cap}} = r\Pi\tau + 2KNr\tau$.

Portable lane. The lane's reserves bridge out to C_E and back at expiry, a one-time round trip of the lane value $V \approx cK$. After that, the holder's individual swap notional K stays local and is confirmed in one block $\delta_{\text{block}} = 12\,\text{s} = 0.2\,\text{min}$. The per-swap execution lock falls to

$$F_{\text{exec,local}} = K\, r\, \delta_{\text{block}} \approx \$0.0076.$$

Capital-lag cost for the first local swap after the lane arrives: $r\Pi\tau + F_{\text{conf,bridge}} + F_{\text{exec,local}}$, where $F_{\text{conf,bridge}} = 2V\, r\, \tau \approx 2 \times cK \times F_{\text{execution}}/K$. The saving comes from replacing repeated $F_{\text{confirmation}} = \$1.83$ crossings with $F_{\text{exec,local}} \approx \$0.008$ per trade. Table A.1 collects the three lags side by side.

Lag	**Simple AMM**	**AMM-side lane**	**Portable lane**
Price	Quote 90.91, delivery 23.81 after Mars moves.	Stale lane gives lane output 22.73, then isolates later public flow.	Same stale lane output 22.73, active locally after export.
Execution	Executes on Mars after 24 min, result known after 48 min.	First acknowledgement-informed holder swap executes on Mars after 72 min.	First local holder swap can execute on Earth after 48 min 12 s.
Capital	$F_{\text{confirmation}} = \$1.83$ per swap (fill or revert).	$r\Pi\tau + F_{\text{confirmation}} = r\Pi\tau + \$1.83$ per swap.	$r\Pi\tau + F_{\text{conf,bridge}} + F_{\text{exec,local}} \approx r\Pi\tau + 2cKr\tau + \$0.008$ for the first local swap.

Table A.1: Worked comparison using $(X, Y) = (100, 1000)$, $\Delta x = 10$, $\tau = 24\,\text{min}$, $\delta_E = 12\,\text{s}$, $K = \$100{,}000$, and $r = 20\%$ per year.

Appendix B

Lane merge proofs

This appendix derives the merged pool's invariant and proves Proposition 4.2, that merging an expired lane back into its public pool never lowers the invariant and raises it strictly the further the two sub-pools have drifted apart.

Fix the split fraction $c \in (0,1)$. At the open the parent (X, Y) with $k = XY$ divides into a public pool $(1-c)(X,Y)$ and a lane $c(X,Y)$, so by Eq. (4.2) their invariants are $k_{\text{public}} = (1-c)^2 k$ and $k_{\text{lane}} = c^2 k$. Each sub-pool is constant-product, so in this model trading over the lease moves its marginal price but leaves its invariant fixed. At expiry write the public reserves (X_p, Y_p) and the lane reserves (X_ℓ, Y_ℓ), the marginal prices $P_{\text{public}} = Y_p/X_p$ and $P_{\text{lane}} = Y_\ell/X_\ell$, and their ratio $\rho = P_{\text{lane}}/P_{\text{public}}$.

B.1 Deriving the merged invariant

Lemma B.1 (Reserves from invariant and price)**.** *A constant-product pool with invariant $\kappa > 0$ at marginal price $P > 0$ holds reserves*

$$X = \sqrt{\kappa/P}, \qquad Y = \sqrt{\kappa P}.$$

Proof. The reserves satisfy $XY = \kappa$ and $Y/X = P$. Multiplying the two gives $Y^2 = \kappa P$ and dividing them gives $X^2 = \kappa/P$. Reserves are positive, so each is the positive root. □

Lemma B.2 (Cross term)**.** *At expiry, free of P_{public},*

$$X_p Y_\ell + X_\ell Y_p = c(1-c)\, k\big(\sqrt{\rho} + 1/\sqrt{\rho}\big).$$

Proof. Apply Lemma B.1 to each sub-pool, using $\sqrt{k_{\text{public}}} = (1-c)\sqrt{k}$, $\sqrt{k_{\text{lane}}} = c\sqrt{k}$, and $P_{\text{lane}} = \rho\, P_{\text{public}}$,

$$X_p = (1-c)\sqrt{\tfrac{k}{P_{\text{public}}}}, \quad Y_p = (1-c)\sqrt{k\, P_{\text{public}}}, \quad X_\ell = c\sqrt{\tfrac{k}{\rho P_{\text{public}}}}, \quad Y_\ell = c\sqrt{k\,\rho P_{\text{public}}}.$$

Then $X_pY_\ell = c(1-c)\,k\sqrt{\rho}$ and $X_\ell Y_p = c(1-c)\,k/\sqrt{\rho}$, in each of which P_{public} cancels. Adding them gives the claim. □

The merge adds the reserves (Eq. (4.9)), so expanding the merged invariant,

$$k_{\text{merged}} = (X_p + X_\ell)(Y_p + Y_\ell) = X_pY_p + X_\ell Y_\ell + \big(X_pY_\ell + X_\ell Y_p\big).$$

Trading over the lease leaves each sub-pool's invariant fixed, so $X_pY_p = k_{\text{public}} = (1-c)^2k$ and $X_\ell Y_\ell = k_{\text{lane}} = c^2k$, and the cross term is Lemma B.2. Substituting,

$$k_{\text{merged}} = (1-c)^2k + c^2k + c(1-c)\,k\big(\sqrt{\rho} + \tfrac{1}{\sqrt{\rho}}\big) = k\Big[(1-c)^2 + c^2 + c(1-c)\big(\sqrt{\rho} + \tfrac{1}{\sqrt{\rho}}\big)\Big],$$

which is Eq. (4.12). Since $(1-c)^2 + c^2 - 1 = -2c(1-c)$, the surplus is

$$k_{\text{merged}} - k = c(1-c)\,k\,g(\rho), \qquad g(\rho) = \sqrt{\rho} + \tfrac{1}{\sqrt{\rho}} - 2. \tag{B.1}$$

B.2 The surplus is non-negative

Lemma B.3 (The bracket)**.** *For $\rho > 0$ set $g(\rho) = \sqrt{\rho} + 1/\sqrt{\rho} - 2$. Then $g(\rho) \geq 0$ with equality only at $\rho = 1$, the value is symmetric under $\rho \mapsto 1/\rho$, and g is strictly decreasing on $(0,1)$ and strictly increasing on $(1,\infty)$.*

Proof. Put $u = \sqrt{\rho} > 0$. Then

$$g = u + \frac{1}{u} - 2 = \frac{(u-1)^2}{u} \geq 0,$$

which is zero only at $u = 1$, that is at $\rho = 1$. Replacing ρ by $1/\rho$ interchanges $\sqrt{\rho}$ and $1/\sqrt{\rho}$ and so leaves g unchanged. Differentiating,

$$g'(\rho) = \tfrac{1}{2}\rho^{-1/2} - \tfrac{1}{2}\rho^{-3/2} = \frac{\rho - 1}{2\,\rho^{3/2}},$$

which is negative for $\rho < 1$ and positive for $\rho > 1$. □

In Eq. (B.1) the factor $c(1-c)$ is strictly positive on $(0,1)$, and by Lemma B.3 $g(\rho) \geq 0$ with equality only at $\rho = 1$. Hence $k_{\text{merged}} \geq k$, strict whenever the pools merge at different prices, which is the bound and its equality case. By the monotonicity and symmetry of Lemma B.3 the surplus depends only on the magnitude of the drift and grows strictly as ρ moves away from 1 either way. For a fixed drift it is largest at the even split $c = \frac{1}{2}$, where $c(1-c)$ is maximal. This proves Proposition 4.2.

Appendix C

Self-induced divergence

A holder might hope to profit from the merge they cause, driving their lane's price away from the public pool's, letting the lane merge back to shock the public price, then trading the public pool on that shock. This appendix shows the round trip loses, because creating the divergence in the thin lane costs more slippage than the shock returns.

Setup. A pool holds reserves (X, Y) at price $P = Y/X$. A lane splits off a fraction c, so the lane holds (cX, cY) and the public pool $(1-c)(X, Y)$, both at P. The lane's reserves belong to the provider, and the holder only has the right to trade against them over the lease. Take the fee-free case, which is the most favourable to the attacker.

The round trip. The holder trades the lane to displace it, then trades the public pool after the merge. Every trade in a constant-product pool returns strictly less than the marginal price promises, by the slippage the trade itself causes, and the merge is not a trade the holder can price. It merges the lane's reserves back at whatever ratio the holder left them. So the holder pays slippage on the lane leg and on the public leg and is handed nothing in between. The lane leg is the expensive one, because the lane is thin. Moving a fraction-c pool by a given amount costs slippage in inverse proportion to c. A smaller lane lets the holder manufacture a larger shock, but makes them pay more per unit to manufacture it, and the two do not cross.

A worked example. Take $(X, Y) = (1000, 1000)$, so $P = 1$, and $c = 0.1$, so the lane is $(100, 100)$ and the public pool $(900, 900)$. The holder buys 50 of X from the lane. To keep the lane's product at 100×100 it must pay in Y until the lane is $(50, 200)$, so it pays 100 of Y for 50 of X, an average of 2 per unit against a marginal price of 1, the slippage of moving a thin pool. The lane now prices X at $200/50 = 4$, far above the public 1. At expiry the lane merges into the public pool, which becomes $(950, 1100)$ at price ≈ 1.16, the shock. The holder unwinds, selling its 50 of X back into the shocked pool, and receives $1100 \times 50/(950 + 50) = 55$ of Y. Across the round trip the holder's X

is unchanged (+50 then −50) and its Y is $-100 + 55 = -45$. The holder is 45 of Y in the hole, and a fee on either leg only deepens it.

Why it cannot be rescued. Front-running the holder's own merge, buying public X before the shock and selling after, trades in the profitable direction, so it offsets the cost of manufacturing the divergence rather than adding to it. The offset is at best exact. The gain from buying low and selling into the shock just cancels the slippage paid to displace the thin lane, at one optimal front-run size and strictly less at any other, so the round trip reaches break-even and goes no further. The extra liquidity the merge releases is the non-negative surplus that stays with the provider (Proposition 4.2), not the holder, so a holder trading only against the pools can at most break even. A holder is therefore never paid to diverge their own lane.