CenturionDEX v2: Worked Examples with Real Numbers

Constant-Product AMM, Swaps, Fees, LP Tokens, and the Price Oracle

Centurion Labs*

Abstract

This note presents a self-contained collection of fully worked numerical examples for the core operations of a CenturionDEX v2 pool. The subject is the classical two-asset constant-product automated market maker: one invariant $x \cdot y = k$, one fixed swap fee $\varphi = 0.3\%$, and one fungible CRC-20 liquidity token whose pro-rata claim is uniform across the price axis $(0, \infty)$. We work every example numerically with exact rationals where possible and four-significant-figure decimals otherwise, and we carry a single canonical pool $(x_0, y_0) = (1000$ CTN, 2,025,000 USDC) with $k_0 = 2{,}025{,}000{,}000$, $p_0 = 2025$, and $L_0 = \sqrt{k_0} = 45{,}000$ throughout, so that every reserve update, swap output, and LP-token issuance is reproducible by hand. The content covers: minting (first and subsequent) with the `MINIMUM_LIQUIDITY` dust burn, pro-rata burning, position value $V(p) = 2\sqrt{kp}$ tabulated across a half-decade of prices, the closed-form impermanent loss $\mathrm{IL}(\alpha) = 2\sqrt{\alpha}/(1+\alpha) - 1$, and the four canonical swap cases (exact-in sell, exact-in buy, exact-out sell, exact-out buy) with the fee-in-reserve bookkeeping that makes $\sqrt{k}$ monotone. We then develop price impact and slippage in closed form, the growth of $\sqrt{k}$ as the sufficient statistic for accrued fees, and the optional 1/6 protocol-fee split with two independent numerical reconciliations against the limit derivation. The block-level arithmetic-mean TWAP oracle is presented with a three-segment numerical window, a wrap-safe computation that crosses $\tau = 2^{32}$, and the internals of the UQ112.112 accumulator representation. Flash swaps are covered in two flavors—in-kind repayment and a two-pool arbitrage with explicit feasibility check—and multi-pool routing is compared in a concrete two-hop example with compounded fee $\gamma^2 = 0.994009$. A dedicated section traces one hypothetical LP through mint, fee accrual, and burn over thirty days, and an explicit fungibility-vs-NFT contrast with the concentrated-liquidity design of v3 is drawn so that the reader can see why v2 needs no per-position state at all. Two appendices collect proofs of the identities used in the body and the fixed-point arithmetic of the UQ112.112 accumulator, including the rounding-in-favor-of-pool convention and the 224-bit overflow accounting. The v3 design appears only as a reference frame for orientation and never enters any v2 computation.

Reader's guide

The presentation is self-contained but technical. Readers unfamiliar with constant-product automated market makers (AMMs) may prefer to read this guide first; each subsequent section also opens with a short pedagogical note before proceeding to formal statements.

Conceptual orientation

A CenturionDEX v2 pool is a two-asset automated market maker in which every liquidity provider (LP) deposits both tokens in proportion to the prevailing reserve ratio and receives a fungible CRC-20 liquidity token LPT representing a pro-rata claim on the pool. Reserves evolve under the *constant-product invariant* $x \cdot y = k$, an equality (up to fees and rounding) that is enforced by the pair contract at every swap and restored by every mint or burn. The spot price is $p = y/x$; the pool's depth, measured by $L := \sqrt{xy} = \sqrt{k}$, is uniform across all prices in $(0, \infty)$, so every LP is exposed to the full curve on equal terms. Swap fees are charged by withholding a fraction $\varphi = 0.003$ of the input token from the swap formula and leaving it in the

*https://centurionchain.org

reserves, which makes $\sqrt{k}$ monotonically non-decreasing in time and folds accrued fees directly into every LP token without any per-position bookkeeping.

Relation to concentrated liquidity. The v2 design is the canonical single-curve constant-product AMM: one invariant, one fee tier $\varphi = 0.3\%$, uniform depth on $(0, \infty)$, fungible LP tokens, and deposits in 50/50 value split by construction. The concentrated-liquidity design of Centurion v3 generalizes this by allowing each LP to restrict capital to a chosen price interval $[p_a, p_b]$, which multiplies local depth but introduces non-fungibility, out-of-band fee accounting, and a discrete family of fee tiers. In this paper v3 plays the role of an asymptotic reference only: every formula and every number below is a strictly v2 quantity, and the degeneration of a v3 position with $p_a \to 0$ and $p_b \to \infty$ recovers the v2 full-range LP that is our object of study. Readers who begin with v3 can view v2 as the single limit in which all of v3's machinery—ticks, $\sqrt{p}$ state, NFTs, outside-growth accumulators—collapses to one scalar pair (x, y) and one scalar invariant k.

Notation and terminology

Symbol / term	Definition
Assets and prices	
CTN, USDC	The two pool assets; X = CTN (base), Y = USDC (quote).
p	Spot price, $p = y/x$, in USDC per CTN.
$\bar{p}$	Average execution price of a swap, $\bar{p} = \lvert\Delta y\rvert/\lvert\Delta x\rvert$.
p_0	Canonical reference price 2025 USDC/CTN.
Reserves and depth	
$x,\ y$	On-chain reserves of CTN and USDC, in native units.
k	Constant-product invariant, $k = x \cdot y$.
L	Pool depth, $L = \sqrt{k} = \sqrt{xy}$.
Swaps and fees	
$\Delta x_{\text{in}}, \Delta y_{\text{in}}$	Gross input amounts (pre-fee) of a swap.
$\Delta x_{\text{out}}, \Delta y_{\text{out}}$	Output amounts received by the swapper.
φ	Swap fee fraction; $\varphi = 0.003$ in v2.
γ	Fee retention $\gamma := 1 - \varphi = 0.997$.
ϕ_{p}	Optional protocol-fee share; 1/6 of φ when on, 0 when off.
LP tokens and oracle	
S	Total supply of the CRC-20 LP token.
s	Balance of LP tokens held by a specific LP.
$\sqrt{k}_{\text{last}}$	Snapshot of $\sqrt{k}$ taken at the last liquidity event (protocol-fee basis).
$c_0^t,\ c_1^t$	Price-cumulative accumulators, encoded in UQ112.112 fixed-point.
τ	Block timestamp, uint32, wraps at 2^{32} seconds.

Canonical running example

Throughout this note we carry a single pool configuration to keep every numerical chain of reasoning anchored:

$$x_0 = 1000 \text{ CTN}, \qquad y_0 = 2{,}025{,}000 \text{ USDC}, \qquad k = 2{,}025{,}000{,}000, \qquad L = \sqrt{k} = 45{,}000.$$

The spot price is $p_0 = y_0/x_0 = 2025$ USDC/CTN, and $\sqrt{p_0} = 45$ so that all square roots are clean rationals. The swap fee is $\varphi = 0.003$, the protocol fee is assumed off unless stated, and the minimum-liquidity burn at first mint is fixed at 1000 wei-LP.

Suggested reading path

A reader who wants only the swap mechanics can read Sections 2, 7, 8, and 9. A reader focused on the LP experience should read Sections 3, 4, 5, 6, 10, and 16. A reader focused on the oracle and protocol-level economics should read Sections 11, 12, and 13. The theoretical Section 18, the edge-case catalogue in Section 19, and the two appendices are recommended on a second pass.

1 Notation, state, and conventions

Beginner note. This section fixes symbols, units, and sign conventions. All later sections use the notation introduced here without further comment. Skimmable on a first read; essential on a second.

A CenturionDEX v2 pair contract stores the pool state in a single packed storage slot together with two auxiliary accumulators:

`reserve0`	112-bit reserve of token 0 (CTN in our running example).
`reserve1`	112-bit reserve of token 1 (USDC in our running example).
`blockTimestampLast`	32-bit timestamp of the last `sync` or mutating call.
`price0CumulativeLast`	UQ112.112 accumulator $\int (y/x)\, d\tau$.
`price1CumulativeLast`	UQ112.112 accumulator $\int (x/y)\, d\tau$.
`kLast`	Snapshot of $x \cdot y$ used by the protocol fee.

All reserve and token quantities are non-negative integers; prices and ratios are computed in user-facing presentation at 18 decimals for CTN and 6 decimals for USDC in practice, but in this note we drop decimal scaling and treat 1 CTN and 1 USDC as atomic units whose smallest subdivision is the wei. Throughout, Δx and Δy are *signed* when discussing reserve changes (a sell of 10 CTN in returns $\Delta x = +10$ for the pool and $\Delta y < 0$ for the pool), but *unsigned* when reported as swap inputs and outputs.

Example 1.1 (Packing the storage slot)**.** *Storing* $(x, y, \tau) = (1000 \cdot 10^{18},\ 2{,}025{,}000 \cdot 10^{6},\ 1{,}700{,}000{,}000)$ *uses* $112 + 112 + 32 = 256$ *bits. The* CTN *reserve* $10^{21} \approx 2^{70}$ *and the* USDC *reserve* $2.025 \cdot 10^{12} \approx 2^{41}$*, both well inside the* 112*-bit ceiling of* $2^{112} - 1 \approx 5.19 \cdot 10^{33}$*. A single* `SLOAD` *fetches the full triple, which is why every read of* (x, y) *in v2 is cheap enough to run on-chain without caching.*

Remark. *The* 112*-bit reserve width is a hard cap inherited from the* UQ112.112 *accumulator arithmetic: the product* $y \cdot 2^{112}/x$ *must fit in* 224 *bits before being added to a* 224*-bit cumulative, which forces both reserves to fit in* 112 *bits simultaneously. No pool has ever approached this bound in production.*

2 The constant-product invariant

Beginner note. The invariant $x \cdot y = k$ is the single equation that governs every swap. It is the v2 analogue of a pricing curve: the pool will always quote whatever swap keeps the post-trade reserves on the same hyperbola (up to the explicit fee carve-out). This section explains how that curve encodes a price, depth, and inventory simultaneously.

Definition 2.1 (Invariant)**.** *A v2 pool is said to satisfy the constant-product invariant at state* (x, y) *if* $k := xy > 0$*. A swap from state* (x_1, y_1) *to state* (x_2, y_2) *is* fee-free admissible *iff* $x_2 y_2 = x_1 y_1$*, and* fee-charging admissible *iff* $x_2 y_2 \geq x_1 y_1$*, with equality up to the rounding in favor of the pool documented in Appendix B.*

Example 2.1 (Invariant at the running state)**.** $x_0 = 1000$*,* $y_0 = 2{,}025{,}000$ *yields* $k_0 = 2.025 \cdot 10^9$*,* $L_0 = \sqrt{k_0} = 45{,}000$*,* $p_0 = y_0/x_0 = 2025$*, and* $\sqrt{p_0} = 45$*. A hypothetical fee-free sell of* $\Delta x_{\text{in}} = 10$ CTN *would land the pool at* $x_1 = 1010$*, forcing* $y_1 = k_0/x_1 = 2{,}005{,}940.594\overline{059}\ldots$ *and* $\Delta y_{\text{out}} = y_0 - y_1 = 19{,}059.406\ldots$ USDC*. The realized price is* $\bar{p} = \Delta y_{\text{out}}/\Delta x_{\text{in}} = 1905.94\ldots$*, strictly below* $p_0 = 2025$ *because the curve bends: bigger trades pay more slippage.*

Proposition 2.2 (Spot price and local slope)**.** *On any admissible hyperbola $xy = k$, the marginal price is*

$$p(x) \;=\; -\frac{dy}{dx} \;=\; \frac{y}{x} \;=\; \frac{k}{x^2}, \qquad \frac{dp}{dx} \;=\; -\frac{2k}{x^3} \;=\; -\frac{2p}{x}.$$

Example 2.2 (Price sensitivity to a single CTN)**.** *At $(x_0, y_0) = (1000, 2{,}025{,}000)$ and $p_0 = 2025$, a one-CTN perturbation yields $|\Delta p| \approx 2p_0/x_0 = 4.05$ USDC/CTN. This is purely a marginal-price statement; the average price paid on a* 1-CTN *trade differs from p_0 by only half that amount (see §8).*

3 Minting LP tokens

Beginner note. Adding liquidity is always a two-sided operation. Except for the very first deposit, the required ratio of CTN to USDC is fixed by the current reserves. The LP token you receive is a pro-rata claim on the pool; there is no per-LP state beyond the CRC-20 balance.

Let $S \geq 0$ denote the current total supply of the pool's LP token and let (x, y) be the current reserves.

First mint ($S = 0$). If the pool is being bootstrapped, the first LP deposits arbitrary positive amounts $(\Delta x, \Delta y)$ and mints

$$s_1 \;=\; \sqrt{\Delta x \cdot \Delta y} \;-\; \texttt{MINIMUM\_LIQUIDITY}, \qquad \texttt{MINIMUM\_LIQUIDITY} = 1000 \text{ wei-LPT}.$$

The 1000 wei are permanently locked to the zero address, which prevents an adversary from inflating the LP token price of an empty pool.

Subsequent mint ($S > 0$). A later LP deposits $(\Delta x, \Delta y)$ and mints

$$\Delta s \;=\; \min\!\Big(\tfrac{\Delta x \cdot S}{x},\; \tfrac{\Delta y \cdot S}{y}\Big).$$

The min is the contract's defense against an LP who tries to deposit off-ratio: the minority side sets the LP-token grant, and the majority side is effectively a donation.

Example 3.1 (Bootstrap of the canonical pool)**.** *Alice bootstraps the pool with $\Delta x = 1000$* CTN *and $\Delta y = 2{,}025{,}000$* USDC*. Then $\sqrt{\Delta x \Delta y} = \sqrt{2.025 \cdot 10^9} = 45{,}000$, so the geometric-mean LP-token quantity is* 45,000 LPT $= 45{,}000 \cdot 10^{18}$ *wei*-LPT*. Of this,* `MINIMUM_LIQUIDITY` $= 1000$ *wei*-LPT *is minted to the zero address, and Alice receives $s_1 = 45{,}000 \cdot 10^{18} - 1000$ wei-LPT $= 44{,}999{,}999{,}999{,}999{,}999{,}999{,}000$ wei-LPT, which in decimal* LPT *is $45{,}000 - 10^{-15} = 44{,}999.999\,999\,999\,999\,999$* LPT *(fifteen nines after the decimal). The total supply, including the locked dust, is $S = 45{,}000$* LPT*. After this mint, $L = \sqrt{k} = 45{,}000$, which is why we chose these reference numbers: LP token supply and pool depth coincide numerically at initialization.*

Example 3.2 (On-ratio subsequent mint)**.** *Bob deposits $\Delta x = 10$* CTN *and $\Delta y = 20{,}250$* USDC *at state $(1000,\ 2{,}025{,}000,\ S = 45{,}000)$. The ratio is the pool ratio. He mints*

$$\Delta s \;=\; \min\!\Big(\tfrac{10 \cdot 45{,}000}{1000},\; \tfrac{20{,}250 \cdot 45{,}000}{2{,}025{,}000}\Big) \;=\; \min(450,\ 450) = 450 \text{ LPT}.$$

New reserves $(1010,\ 2{,}045{,}250)$; new $S = 45{,}450$; new $k = 2{,}065{,}702{,}500$; new $\sqrt{k} = 45{,}450$. Bob's pro-rata share is $450/45{,}450 = 0.9901\%$, exactly $\Delta s/S$.

Example 3.3 (Off-ratio subsequent mint)**.** *Carol attempts to deposit $\Delta x = 10$* CTN *but only $\Delta y = 10{,}000$* USDC *(the on-ratio amount would be 20,250). Then*

$$\Delta s = \min\!\big(\tfrac{10 \cdot 45{,}000}{1000},\; \tfrac{10{,}000 \cdot 45{,}000}{2{,}025{,}000}\big) = \min(450,\ 222.22\ldots) = 222.22\ldots \text{ LPT}.$$

Carol receives LP tokens reflecting her USDC *deposit; her* CTN *deposit is fully credited into the reserves but only the first $10{,}000/2025 = 4.938\ldots$* CTN*-equivalents are rewarded in LP tokens. The remaining* 5.06 CTN *are effectively donated to incumbent LPs. This is why every v2 front-end computes the correct on-ratio pair before broadcasting a mint.*

4 Burning LP tokens

Beginner note. Burning is the inverse of minting and is always a pro-rata withdrawal. Unlike v3, there is no range choice, no accrued-fee collection step, and no in/out-of-range question: fees were automatically added to the reserves, so you just receive your share of the reserves.

Let b be the number of LP tokens burned, S the current total supply, (x, y) the current reserves.

Proposition 4.1 (Pro-rata burn)**.** *Burning $b \le S$ LP tokens returns*

$$\Delta x \;=\; b \cdot \frac{x}{S}, \qquad \Delta y \;=\; b \cdot \frac{y}{S},$$

and updates $(x, y, S) \to (x{-}\Delta x,\ y{-}\Delta y,\ S{-}b)$. The post-burn invariant is $k' = (x{-}\Delta x)(y{-}\Delta y) = k{\cdot}(1{-}b/S)^2$, and $\sqrt{k'} = \sqrt{k} \cdot (1 - b/S)$.

Example 4.1 (Bob fully exits)**.** *Continuing Example 3.2 with $(x, y, S) = (1010,\ 2{,}045{,}250,\ 45{,}450)$ and Bob burning $b = 450$:*

$$\Delta x = 450 \cdot \tfrac{1010}{45{,}450} = 10, \qquad \Delta y = 450 \cdot \tfrac{2{,}045{,}250}{45{,}450} = 20{,}250.$$

Bob receives exactly what he deposited, because no swap has happened between his mint and burn (no fee accrual, no price drift). New state $(1000,\ 2{,}025{,}000,\ 45{,}000)$: the pool is back to the canonical configuration.

Example 4.2 (Burn after price drift and fees)**.** *Suppose after Bob's mint the price has drifted to $p = 2500$ and $\sqrt{k}$ has grown to 46,000 through accrued swap fees, so reserves are $(x, y) = (\sqrt{k/p}, \sqrt{kp}) = (\sqrt{2116 \cdot 10^6/2500}, \sqrt{2116 \cdot 10^6 \cdot 2500})$, i.e. $x = 920$ and $y = 2{,}300{,}000$. Total supply is still $S = 45{,}450$ because no new mint/burn has happened. Bob's pro-rata burn of* 450 LPT *now returns*

$$\Delta x = 450 \cdot \tfrac{920}{45{,}450} = 9.1089 \text{ CTN}, \qquad \Delta y = 450 \cdot \tfrac{2{,}300{,}000}{45{,}450} = 22{,}772.27 \text{ USDC}.$$

Bob's withdrawal value at $p = 2500$ is $9.1089 \cdot 2500 + 22{,}772.27 = 45{,}544.79$ USDC*. His original deposit valued at $p_0 = 2025$ was $10 \cdot 2025 + 20{,}250 = 40{,}500$* USDC*. The* USDC*-denominated position value has risen, but comparing against a passive hold (§6) is the correct accounting: fees must beat impermanent loss for the LP to have been better off.*

5 Position value $V(p)$ as a function of price

Beginner note. The dollar-value of an LP's share depends only on the current price p and on the constant k that was backing the position when they entered. The formula $V(p) = 2\sqrt{kp}$ is the workhorse of every impermanent-loss computation.

Let an LP hold a fraction $\alpha = s/S$ of the pool. At spot price p, the pool holds $x(p) = \sqrt{k/p}$ and $y(p) = \sqrt{kp}$, so the LP's position is worth

$$V(p) \;=\; \alpha\big(x(p)\,p + y(p)\big) \;=\; \alpha\big(\sqrt{kp} + \sqrt{kp}\big) \;=\; 2\alpha\sqrt{kp}.$$

Note that k here is the current invariant: if fees have grown $\sqrt{k}$ from L_0 to $L_1 > L_0$, then V at the same price p is larger by the factor L_1/L_0.

Proposition 5.1 (Reserve split at any price)**.** *On the canonical hyperbola $xy = k$, the dollar value is always* 50/50*:*

$$x(p) \cdot p = \sqrt{kp} = y(p),$$

so at every admissible state the USDC*-denominated values of the two sides of the pool are equal.*

Example 5.1 (Value at five prices across two decades)**.** *Take a 1% LP ($\alpha = 0.01$) in the canonical pool with $k = 2.025 \cdot 10^9$, so the pool-wide value is $V_{\text{pool}}(p) = 2\sqrt{k\,p}$ and the LP's share is $V(p) = 0.02\sqrt{2.025 \cdot 10^9 \cdot p}$. Writing $p = \alpha p_0$ with $p_0 = 2025$ and using $V(p) = V(p_0)\sqrt{\alpha} = 40{,}500\sqrt{\alpha}$,*

α	p (USDC/CTN)	$\sqrt{k\,p}$ (USDC)	$V_{\text{pool}}(p)$ (USDC)	$V(p)$, $\alpha = 0.01$ (USDC)
1/4	506.25	1,012,500	2,025,000	20,250.00
1/2	1012.50	1,431,782.11	2,863,564.21	28,635.64
1	2025.00	2,025,000	4,050,000	40,500.00
2	4050.00	2,863,564.21	5,727,128.43	57,271.28
4	8100.00	4,050,000	8,100,000	81,000.00

Verification at $\alpha = 1/2$*:* $\sqrt{k\,p} = \sqrt{2.025 \cdot 10^9 \cdot 1012.5} = \sqrt{2.050312\,5 \cdot 10^{12}} = 1{,}431{,}782.11$*, so* $V_{\text{pool}} = 2{,}863{,}564.21$ *and the* 1% *slice is* 28,635.64*. Verification at* $\alpha = 2$*:* $\sqrt{k \cdot 4050} = \sqrt{8.201\,250 \cdot 10^{12}} = 2{,}863{,}564.21$*, giving* $V_{\text{pool}} = 5{,}727{,}128.43$*. The extremal rows are exact:* $\sqrt{k \cdot 506.25} = \sqrt{1.025\,156\,25 \cdot 10^{12}} = 1{,}012{,}500$ *and* $\sqrt{k \cdot 8100} = \sqrt{1.640\,25 \cdot 10^{13}} = 4{,}050{,}000$*. The ratio* $V(2p)/V(p) = \sqrt{2} \approx 1.4142$ *is independent of starting price: LP value scales as* $\sqrt{p}$*, and every step up or down by a factor of* 4 *in* p *moves* V *by exactly a factor of* 2*.*

6 Impermanent loss

Beginner note. If you deposit at price p_0 and the price later moves to $p = \alpha \cdot p_0$, a passive hold of the same tokens would be worth $W(p) = x_0 p + y_0$; your LP position is worth $V(p) = 2\sqrt{kp}$. The ratio $V/W - 1$ is negative for any $\alpha \neq 1$ and is called impermanent loss. It is a deterministic function of α only.

Proposition 6.1 (Impermanent-loss formula)**.** *Let the LP deposit at* (x_0, y_0) *with spot price* $p_0 = y_0/x_0$ *and* $k_0 = x_0 y_0$*. At any later price* $p = \alpha\, p_0$ *with* $\alpha > 0$ *and no fees accrued* ($k = k_0$)*,*

$$W(p) \;=\; x_0 p + y_0 \;=\; y_0(1+\alpha), \qquad V(p) \;=\; 2\sqrt{k_0\, p} \;=\; 2y_0\sqrt{\alpha},$$

so the impermanent loss is

$$\mathrm{IL}(\alpha) \;:=\; \frac{V(p)}{W(p)} - 1 \;=\; \frac{2\sqrt{\alpha}}{1+\alpha} - 1 \;\leq\; 0, \quad \textit{with equality iff } \alpha = 1.$$

Proof. W is the value of the original bundle at the new price. V uses the position-value formula of §5. The ratio follows; the inequality $2\sqrt{\alpha} \leq 1 + \alpha$ is AM-GM, with equality iff $\alpha = 1$. □

Example 6.1 (A standard IL table)**.** *For an LP in the canonical pool with* $y_0 = 2{,}025{,}000$*:*

α	p (USDC/CTN)	$W(p)$	$V(p)$	$\mathrm{IL}(\alpha)$
0.25	506.25	2,531,250	2,025,000	−20.0000%
0.50	1012.50	3,037,500	2,863,564.21	−5.7191%
0.80	1620.00	3,645,000	3,622,485.08	−0.6177%
1.00	2025.00	4,050,000	4,050,000	0
1.25	2531.25	4,556,250	4,528,106.36	−0.6177%
2.00	4050.00	6,075,000	5,727,128.43	−5.7191%
4.00	8100.00	10,125,000	8,100,000	−20.0000%

The symmetry $\mathrm{IL}(\alpha) = \mathrm{IL}(1/\alpha)$ *is a consequence of the* $\sqrt{\alpha}$ *dependence: doubling and halving the price cause identical loss. Independent verification at* $\alpha = 1/2$*:* $V = 2y_0\sqrt{\alpha} = 2 \cdot 2{,}025{,}000/\sqrt{2} = 2{,}863{,}564.21$ *and* $W = y_0(1 + \alpha) = 3{,}037{,}500$*, so* $V/W - 1 = -0.057191 = -5.7191\%$*, matching the closed form* $2\sqrt{1/2}/(1 + 1/2) - 1 = 2\sqrt{2}/3 - 1$*.*

Remark. *IL is the* gross *drawdown vs. passive hold. Actual LP P&L in v2 also includes swap fees accrued via the growth of* $\sqrt{k}$*. In practice, an LP breaks even against passive hold at a given* α *when the fees captured between entry and exit multiply* V *by at least* $W(\alpha)/V_{\text{fee-free}}(\alpha) = (1+\alpha)/(2\sqrt{\alpha})$*.*

7 The four swap cases

Beginner note. Every v2 swap falls into one of four buckets: exact-in sell CTN, exact-in buy CTN, exact-out sell CTN, exact-out buy CTN. The pair contract supports all four through a single `swap` entry point by requiring the caller to push the input token first and asking the contract only for the desired output, with the invariant check at the end.

Let (x, y) be current reserves, $\gamma = 1 - \varphi = 0.997$ the fee retention, and write $\Delta x_{\text{in}}^{\text{eff}} = \gamma\,\Delta x_{\text{in}}$ for the fee-adjusted input. The four cases reduce to two closed forms.

Exact-in. Given the gross input, the output is

$$\Delta y_{\text{out}} = \frac{y \cdot \gamma \Delta x_{\text{in}}}{x + \gamma \Delta x_{\text{in}}}, \qquad \Delta x_{\text{out}} = \frac{x \cdot \gamma \Delta y_{\text{in}}}{y + \gamma \Delta y_{\text{in}}}.$$

Exact-out. Given the desired output, the required input is

$$\Delta x_{\text{in}} = \left\lceil \frac{x \cdot \Delta y_{\text{out}}}{\gamma\,(y - \Delta y_{\text{out}})} \right\rceil, \qquad \Delta y_{\text{in}} = \left\lceil \frac{y \cdot \Delta x_{\text{out}}}{\gamma\,(x - \Delta x_{\text{out}})} \right\rceil.$$

The ceiling is the single wei of rounding in favor of the pool that guarantees the post-trade invariant $x'y' \geq k$. The pool's actual enforcement is the inequality $(x + \Delta x_{\text{in}})(y - \Delta y_{\text{out}}) \cdot \gamma^{[\cdot]} \geq k \cdot 10^6$ after rescaling both sides by 1000×1000 to avoid fractional fee arithmetic; see Appendix A for the exact integer statement.

Example 7.1 (Case A: exact-in sell of 10 CTN)**.** *Trader sends* $\Delta x_{\text{in}} = 10$ CTN*. Fee retention makes the effective input* $\gamma \cdot 10 = 9.97$*. Then*

$$\Delta y_{\text{out}} = \frac{2{,}025{,}000 \cdot 9.97}{1000 + 9.97} = \frac{20{,}189{,}250}{1009.97} = 19{,}989.95 \text{ USDC}.$$

New reserves: $x' = 1010$*,* $y' = 2{,}005{,}010.05$*. New* $k = 1010 \cdot 2{,}005{,}010.05 = 2{,}025{,}060{,}150.5$*, strictly greater than* $2{,}025{,}000{,}000$*. The growth is exactly the fee portion:* $(0.03 \cdot 10) \cdot (2{,}005{,}010.05/1010) \cdot 1010 \approx 60{,}150.3$*, matching the* $\Delta k = 60{,}150.5$ *(small residual from exact fraction). Average price paid:* $\bar{p} = 19{,}989.95/10 = 1998.995$ USDC/CTN*, about* 1.29% *below* p_0*.*

Example 7.2 (Case B: exact-in buy with 20,250 USDC)**.** *Trader sends* $\Delta y_{\text{in}} = 20{,}250$ USDC *to buy* CTN*. Effective input* $\gamma \cdot 20{,}250 = 20{,}189.25$*. Then*

$$\Delta x_{\text{out}} = \frac{1000 \cdot 20{,}189.25}{2{,}025{,}000 + 20{,}189.25} = \frac{20{,}189{,}250}{2{,}045{,}189.25} = 9.87158 \text{ CTN}.$$

New reserves: $x' = 990.12842$*,* $y' = 2{,}045{,}250$*. Using the identity* $k' = k \cdot (y + \Delta y_{\text{in}})/(y + \gamma \Delta y_{\text{in}}) = 2{,}025{,}000{,}000 \cdot 2{,}045{,}250/2{,}045{,}189.25$*, the new invariant is* $k' = 2{,}025{,}060{,}150.89$*, strictly greater than the original by* $60{,}150.89$*. Average price paid:* $\bar{p} = 20{,}250/9.87158 = 2051.35$ USDC/CTN*, about* 1.30% *above* p_0 *(not the same* 1.29% *as case A because the trade is directionally reversed and the curve is not symmetric in* USDC *vs.* CTN *units).*

Example 7.3 (Case C: exact-out buy of 20,000 USDC)**.** *Trader wants* $\Delta y_{\text{out}} = 20{,}000$ USDC *and must pay the minimum* CTN*:*

$$\Delta x_{\text{in}} = \left\lceil \tfrac{1000 \cdot 20{,}000}{0.997 \cdot (2{,}025{,}000 - 20{,}000)} \right\rceil = \left\lceil \tfrac{20{,}000{,}000}{1{,}998{,}985} \right\rceil = \lceil 10.00507\,52\ldots \rceil \;\approx\; 10.005\,076 \text{ CTN}$$

(the ceiling is taken in wei, lifting the final wei by 1*; at* 18*-decimal precision the effect is below the rounding we display). The pool ends up at* $(x', y') = (1010.005\,076,\ 2{,}005{,}000)$ *with* $k' = x'y' = 1010.005\,076 \cdot 2{,}005{,}000 = 2{,}025{,}060{,}177.4$*, an increase of* 177.4 *over* k_0*. Average price paid* $\bar{p} = 20{,}000/10.005\,076 = 1998.985$ USDC/CTN*.*

Example 7.4 (Case D: exact-out sell of 10 CTN)**.** *Trader wants* $\Delta x_{\text{out}} = 10$ CTN *and must pay the minimum* USDC*:*

$$\Delta y_{\text{in}} = \left\lceil \frac{2{,}025{,}000 \cdot 10}{0.997 \cdot (1000-10)} \right\rceil = \left\lceil \frac{20{,}250{,}000}{987.030} \right\rceil = \lceil 20{,}516.0937\ldots \rceil \;\approx\; 20{,}516.094 \text{ USDC}.$$

New reserves (990, 2,045,516.094)*;* $k' = 990 \cdot 2{,}045{,}516.094 = 2{,}025{,}060{,}933.1$*, an increase of* $60{,}933.1$ *over* k_0*. Average price* $\bar{p} = 20{,}516.094/10 = 2051.61$ USDC/CTN*.*

Remark. *Cases A and C are inverses in the* CTN*-in direction at slightly different post-states because the fee is charged on the gross input in A and on the implied input in C. Cases B and D are the* USDC*-in analogues. In every case* $\Delta k > 0$*, and the increment is (to leading order)* $\varphi \cdot (\textit{input}) \cdot (\textit{opposite reserve})/(\textit{own reserve})$*.*

8 Price impact and slippage

Beginner note. For a constant-product pool, the bigger your trade, the worse the execution. This section derives the exact price-impact formula and reads off what it says at small trade sizes (linear in trade size) and large ones (saturating at the full inventory).

Define the *marginal price* before the trade as $p = y/x$ and the *average execution price* as $\bar{p} = |\Delta y|/|\Delta x|$. Their ratio is the price impact.

Proposition 8.1 (Exact-in sell price impact)**.** *For an exact-in sell of* Δx_{in} *CTN with fee* φ*,*

$$\frac{\bar{p}}{p} \;=\; \frac{x}{x + \gamma \Delta x_{\text{in}}} \;=\; 1 - \frac{\gamma \Delta x_{\text{in}}}{x + \gamma \Delta x_{\text{in}}}.$$

Proof. $\bar{p}/p = (\Delta y_{\text{out}}/\Delta x_{\text{in}})/(y/x) = (x \cdot \gamma)/(x + \gamma \Delta x_{\text{in}})$, using the exact-in formula. □

Example 8.1 (Impact at small, medium, and large trade sizes)**.** *At* $x = 1000$*,* $\gamma = 0.997$*, the ratio* $\bar{p}/p$ *is:*

Δx_{in} *(CTN)*	$\bar{p}/p$	*slippage*	*fee share of slippage*
1	0.999004	0.0996%	30.09%
10	0.990128	0.9872%	30.39%
100	0.909339	9.0661%	3.309%
500	0.667334	33.2665%	0.9019%

Computed from $\bar{p}/p = x/(x + \gamma \Delta x_{\text{in}})$ *with* $x = 1000$ *and* $\gamma = 0.997$*; the "fee share of slippage" is* $\varphi/(1 - \bar{p}/p)$*, i.e. the fraction of observed slippage attributable to the swap fee rather than to curve curvature. Small trades are fee-dominated: about* 30% *of the slippage at* 1 *CTN is the* 0.30% *fee and the remainder is pure curvature. Large trades are curvature-dominated: a* 500 *CTN sell is half the inventory and the curve itself accounts for* 33.27% *of the move, with the fee share collapsing to under* 1%*.*

Proposition 8.2 (Closed form for price impact of a fractional trade)**.** *For an exact-in sell of size* $\Delta x_{\text{in}} = \theta\, x$ *(a fraction* $\theta > 0$ *of the reserve), the marginal-price impact and average-execution-price impact are*

$$\eta_{\text{marg}} \;=\; \frac{p'}{p} - 1 \;=\; \frac{1}{(1+\theta)^2} \cdot \frac{k'/k}{1} - 1, \qquad \eta_{\text{exec}} \;=\; \frac{\bar{p}}{p} - 1 \;=\; -\frac{\gamma\theta}{1+\gamma\theta}.$$

The marginal form specializes to $\eta_{\text{marg}} = -1 + (1 + \varphi\theta/(1+\gamma\theta))/(1+\theta)^2$ *and satisfies* $\eta_{\text{marg}} \le \eta_{\text{exec}} < 0$*: the post-trade marginal price is strictly worse than the average price paid, because the last wei of the trade traverses more curve than the first wei.*

Example 8.2 (Slippage-tolerance revert scenario)**.** *A user calls* `swapExactTokensForTokens` *with* $\Delta x_{\text{in}} = 10$ CTN *and a user-supplied* `amountOutMin` *computed against a stale quote of* $p_{\text{quote}} = 2025$ *with a* 0.5% *tolerance:*

amountOutMin $= 10 \cdot 2025 \cdot (1 - 0.005) = 20{,}148.75$ USDC. *In the interval between quote and inclusion, an intervening trade pushes the pool to* $(x, y) = (1050,\ 1{,}928{,}571.43)$ *(same* k*), so our user's swap now yields*

$$\Delta y_{\text{out}} = \frac{1{,}928{,}571.43 \cdot 9.97}{1050 + 9.97} = 18{,}138.09 \text{ USDC} < \textit{amountOutMin} = 20{,}148.75 \text{ USDC}.$$

The Router's guard $\Delta y_{\text{out}} \geq$ *amountOutMin fails and the transaction reverts with the error string* INSUFFICIENT_OUTPUT_AMOUNT*. This is the user-side slippage check doing its job: the user intended a* 2025*-priced trade with* 0.5% *tolerance, but the pool is now at roughly* 1836*, outside tolerance. A well-behaved Router front-end recomputes* amountOutMin *against the most recent pool state immediately before submission.*

Why this matters in production. Price impact is the mechanism by which every v2 pool self-prices its own scarcity. Production integrations must (i) always pair an `amountIn` with a tolerance-adjusted `amountOutMin`, (ii) refresh the pool state between quote and submission whenever the interval exceeds a block, and (iii) size trades so that η_{exec} stays below the counterparty's risk budget. The closed form above is cheap to evaluate in a front-end and should be preferred over numerical swap simulation when the only goal is a sizing gate.

9 The swap fee φ and its bookkeeping

Beginner note. The flat 0.30% fee in v2 is applied by multiplying the input by $1 - \varphi = 0.997$ before the swap is executed. The withheld $0.003 \cdot \Delta x_{\text{in}}$ stays in the reserves forever (modulo the optional protocol skim). There is no separate fee vault.

Proposition 9.1 (Fee goes straight to reserves)**.** *After an exact-in sell with gross input* Δx_{in}*,*

$$x' \;=\; x + \Delta x_{\text{in}}, \qquad y' \;=\; y - \Delta y_{\text{out}}, \qquad k' \;=\; k \cdot \Big(1 + \frac{\varphi \Delta x_{\text{in}}}{x + \gamma \Delta x_{\text{in}}}\Big).$$

Proof. Substitute $\Delta y_{\text{out}} = y\gamma\Delta x_{\text{in}}/(x + \gamma\Delta x_{\text{in}})$ into $x'y'$ and simplify. □

Example 9.1 (Δk and $\sqrt{k}$ growth on a 10-CTN sell)**.** *Using Example 7.1:* $k' = 2{,}025{,}060{,}150.5$*, a relative growth of* $\Delta k/k = 2.9704 \cdot 10^{-5}$*. So* $\sqrt{k'}/\sqrt{k} - 1 \approx 1.4852 \cdot 10^{-5}$*, i.e. L grew from* 45,000 *to* 45,000.668*. Every LP's position value has risen by a factor of* 1.00001485 *as a direct consequence of this single trade.*

Example 9.2 (Fee take per unit volume)**.** *Over a day, say* $V = 10M$ *USDC of volume flows through the pool in both directions in small, low-impact trades, with average effective price* $\approx p_0 = 2025$*. The cumulative fee captured is* $\approx \varphi \cdot V = 30{,}000$ *USDC-equivalent. At the initial* $k = 2.025 \cdot 10^9$*, this raises* $\sqrt{k}$ *by a factor* $\sqrt{1 + 30{,}000/(y_0/\textit{effective scaling})}$*; for small, symmetric flow the scaling reduces to* $\Delta\sqrt{k}/\sqrt{k} \approx \varphi V/(2L\bar{p}) = 30{,}000/(2 \cdot 45{,}000 \cdot 2025) \approx 1.65 \cdot 10^{-4}$*, i.e. a* 0.0165% *boost to every LP's position for that day's flow.*

10 $\sqrt{k}$ as the canonical fee-growth summary

Beginner note. There is no separate fee accumulator in v2. The single scalar $\sqrt{k}$ monotonically encodes all accrued fees. Comparing $\sqrt{k}$ at two times gives the LP-token-equivalent fee growth between them, which is exactly what the optional protocol fee uses.

Proposition 10.1 (Monotonicity)**.** *For any admissible swap,* $k' \geq k$ *with equality iff* $\varphi = 0$*. Over any sequence of swaps (without mints or burns),* $\sqrt{k}$ *is non-decreasing.*

Proposition 10.2 (Fee growth as an LP multiplier)**.** *Suppose at time* t_1 *the pool has* $(\sqrt{k_1},\ S_1)$*, and at time* $t_2 > t_1$*, with no mints or burns in between, it has* $(\sqrt{k_2},\ S_2 = S_1)$*. Then an LP's pro-rata position value at fixed price has grown by exactly* $\sqrt{k_2}/\sqrt{k_1}$*.*

Proof. Between mint/burn events, S is constant and $V(p) = 2(s/S)\sqrt{k\,p}$; ratio at fixed p is $\sqrt{k_2}/\sqrt{k_1}$. □

Example 10.1 (From 45,000 to 45,500)**.** *Suppose over a week of trading the pool grows from* $\sqrt{k} = 45{,}000$ *to* $\sqrt{k} = 45{,}500$ *at constant* $S = 45{,}000$*. An LP's position has appreciated by* $1.0111\ldots$*, i.e.* 1.11% *of pool value in LP-token-equivalent terms. If the price has also moved, the total P&L is* $V_2/V_1 = (\sqrt{k_2}/\sqrt{k_1}) \cdot \sqrt{p_2/p_1}$.

Why this matters in production. In v3, fee growth is tracked per-range in an outside/inside accumulator pair, and an LP must actively collect or compound. In v2, $\sqrt{k}$ is the *only* fee signal, is automatically compounded into every LP token, and requires no per-position state. An indexer that wants to compute LP realized yields should tabulate $\sqrt{k}$ at block boundaries and reconstruct LP returns as products of consecutive ratios; there is no need to scan every swap event individually.

11 The block-level TWAP oracle

Beginner note. v2's oracle is a simple arithmetic-mean time-weighted average price computed from two cumulative price accumulators. Consumers read the accumulators at two block heights, subtract, and divide by the elapsed time. The pair contract does not itself compute the average; the consumer does.

The pair contract maintains two cumulative accumulators updated at the first touch of a new block:

$$c_0^t \;=\; c_0^{t-1} + \mathrm{UQ112.112}\big(y/x\big) \cdot (\tau_t - \tau_{t-1}), \qquad c_1^t \;=\; c_1^{t-1} + \mathrm{UQ112.112}\big(x/y\big) \cdot (\tau_t - \tau_{t-1}).$$

The price used is the *marginal price at the start of the block*, not during it: the accumulator records the pre-trade price multiplied by the block duration. This prevents within-block manipulation of the TWAP: an attacker who moves the price mid-block still contributes the pre-block price to the accumulator.

Proposition 11.1 (Arithmetic-mean TWAP)**.** *A consumer that reads* $(c_0^{t_{\text{start}}}, \tau_{\text{start}})$ *and* $(c_0^{t_{\text{end}}}, \tau_{\text{end}})$ *computes*

$$\mathrm{TWAP}_0[\tau_{\text{start}}, \tau_{\text{end}}] \;=\; \frac{c_0^{t_{\text{end}}} - c_0^{t_{\text{start}}}}{\tau_{\text{end}} - \tau_{\text{start}}}.$$

This equals the arithmetic mean of the marginal price $p = y/x$ *weighted by block duration.*

Example 11.1 (Three-segment TWAP window)**.** *Suppose a consumer wants the TWAP over a window of length* $\Delta\tau = 900$ *seconds, during which the pool experienced three price regimes at the block boundaries:*

Segment duration (s)	*pre-block p*	*contribution to accumulator*
300	2000	600,000
400	2025	810,000
200	2100	420,000
900		1,830,000

TWAP = 1,830,000/900 = 2033.33 USDC/CTN*. Compare to the instantaneous p at the end of the window (*2100*) and at the start (*2000*): the TWAP is a duration-weighted average, not an endpoint average.*

Example 11.2 (Why marginal, not realized, price)**.** *If within one block the pool moves from* $p = 2000$ *to* $p = 2100$ *via a single large trade, the accumulator records the pre-block price* 2000 *times the full block duration. The post-trade price* 2100 *enters the accumulator only on the first touch of the* next *block. This one-block lag is the v2 oracle's primary security mechanism: within a single block, the manipulator pays slippage and fees but cannot retroactively alter the accumulator.*

Why this matters in production. A TWAP consumer that fails to distinguish marginal from realized price is vulnerable to sandwich-style manipulation in the one-block window at the edge of its sampling interval. The v2 oracle's design eliminates *intra-block* manipulation entirely but does nothing about multi-block sustained attacks, so window length and capital cost must be traded off at integration time. A 30-minute window on a 12-second block chain is generally considered the minimum for lending-adjacent use cases.

12 Oracle internals and fixed-point encoding

Beginner note. The accumulators are stored as 224-bit unsigned integers in UQ112.112 format. The block timestamp is 32 bits and wraps every ~ 136 years; the arithmetic is designed so that the wraparound is transparent to a consumer that subtracts two accumulator readings.

Proposition 12.1 (Wraparound transparency)**.** *Let $\tau_a, \tau_b \in \{0, \ldots, 2^{32}-1\}$ with τ_b possibly less than τ_a due to wrap. Then the unsigned subtraction $(\tau_b - \tau_a) \bmod 2^{32}$ equals the true elapsed time provided that elapsed $< 2^{32}$. The same holds for accumulator differences modulo 2^{224}.*

Example 12.1 (Wrap at $\tau = 2^{32} - 100$)**.** *A consumer reads $\tau_{\text{start}} = 2^{32} - 200 = 4{,}294{,}967{,}096$ and $\tau_{\text{end}} = 100$. The raw difference is negative, but the modular difference $(100 - (2^{32} - 200)) \bmod 2^{32} = 300$ is correct. The* UQ112.112 *accumulator differs by the same modular subtraction; since the accumulator's maximum increment per block is bounded (see below),* 300 *seconds of TWAP never overflows a* 224*-bit modular difference.*

Example 12.2 (Wrap-safe TWAP over a $\tau = 2^{32}$ boundary)**.** *Suppose the canonical pool crosses $\tau = 2^{32}$ at block height $H^\star$ with $\sqrt{k}$ unchanged, and the consumer samples at $\tau_{\text{start}} = 2^{32} - 120$ (in block $H^\star - 10$ say, at price $p = 2020$) and at $\tau_{\text{end}} = 180$ (in block $H^\star + 15$, at price $p = 2028$). The pair's accumulator stores values modulo 2^{256}, and the consumer reads $c_0^{t_{\text{start}}}$ and $c_0^{t_{\text{end}}}$ directly. Suppose the pre-start accumulator is $c_0^{t_{\text{start}}} = C$ (a* 256*-bit opaque number whose absolute value is irrelevant). Over the* 300*-second window the pool contributes pre-block prices totaling, in* UQ112.112 *units, $\tilde{p}_{\text{avg}} \cdot 300$, so*

$$c_0^{t_{\text{end}}} \;=\; \big(C + 300 \cdot \tilde{p}_{\text{avg}}\big) \bmod 2^{256},$$

and the consumer's wrap-safe subtraction $(c_0^{t_{\text{end}}} - c_0^{t_{\text{start}}}) \bmod 2^{256} = 300 \cdot \tilde{p}_{\text{avg}}$ recovers the true weighted accumulator regardless of whether the τ wrap occurred inside the window. Dividing by $(\tau_{\text{end}} - \tau_{\text{start}}) \bmod 2^{32} = 300$ gives the plain UQ112.112 *TWAP, and a final right-shift by* 112 *converts to a plain integer price ≈ 2024* USDC/CTN*. The single-block discretization error is at most one block's price deviation, which at $\sim$ 12*-second blocks and normal volatility is well under* 0.1%*.*

Why this matters in production. Cumulative-price math that is not wrap-safe can silently produce negative or wildly large TWAPs at $\tau \approx 2^{32}$, i.e. February 2106. No production pool has yet encountered this, but any long-lived integration should use unsigned modular subtraction for both the time delta and the accumulator delta to guarantee correctness across the boundary.

Proposition 12.2 (Accumulator overflow bound)**.** *In any window of elapsed time $\Delta\tau < 2^{32}$, the accumulator difference satisfies*

$$c_0^{t_{\text{end}}} - c_0^{t_{\text{start}}} \;\le\; \Delta\tau \cdot p_{\max} \cdot 2^{112},$$

where $p_{\max}$ is the maximum marginal price in the window. For $p_{\max} \le 2^{112}$ the product fits in 224 *bits.*

Implementation checklist for oracle consumers. Production integrations should enforce three invariants when using v2 TWAP: (i) snapshot the cumulatives and timestamp from the same read-path (no mixed-block reads), (ii) reject windows with $\Delta\tau = 0$ or stale timestamps, and (iii) compute both time and cumulative deltas with unsigned modular subtraction. These three checks eliminate nearly all practical integration bugs around wraparound and stale-oracle reads.

Example 12.3 (UQ112.112 encoding of $p = 2025$)**.** $2025 \cdot 2^{112} = 2025 \cdot 5.1923 \cdot 10^{33} \approx 1.0515 \cdot 10^{37}$*, representable in* 224 *bits. In one day, the maximum contribution of a block with $p = 2025$ is $2025 \cdot 2^{112} \cdot 86{,}400 \approx 9.08 \cdot 10^{41}$, still representable in the* 224*-bit accumulator.*

13 The protocol fee (optional 1/6 capture)

Beginner note. v2 has a protocol-level fee toggle that, when enabled, mints new LP tokens at each liquidity event representing 1/6 of the fees accrued since the last event. This is implemented by comparing $\sqrt{k}$ at the event against a snapshot $\sqrt{k}_{\text{last}}$.

Proposition 13.1 (Protocol-fee mint)**.** *At a liquidity event (mint or burn) with current state* $(\sqrt{k}, S)$ *and stored snapshot* $\sqrt{k}_{\text{last}}$*, the contract mints to the fee recipient:*

$$\Delta s_{\text{prot}} \;=\; S \cdot \frac{\sqrt{k} - \sqrt{k}_{\text{last}}}{5\sqrt{k} + \sqrt{k}_{\text{last}}}, \qquad \textit{then set } \sqrt{k}_{\text{last}} \leftarrow \sqrt{k}.$$

Proposition 13.2 (1/6 limit)**.** *In the limit of small fee growth* $\varepsilon = (\sqrt{k} - \sqrt{k}_{\text{last}})/\sqrt{k}_{\text{last}} \to 0$*, the minted protocol share divided by the total fee growth satisfies*

$$\lim_{\varepsilon \to 0} \frac{\Delta s_{\text{prot}}/S}{\varepsilon} \;=\; \frac{1}{6}.$$

Proof. Write $\sqrt{k} = \sqrt{k}_{\text{last}}(1+\varepsilon)$. Then $\Delta s_{\text{prot}}/S = \varepsilon/(5(1+\varepsilon)+1) = \varepsilon/(6+5\varepsilon)$, and as $\varepsilon \to 0$ this is $\varepsilon/6 + O(\varepsilon^2)$. □

Example 13.1 (Protocol-fee reconciliation on a small growth)**.** *Suppose* $\sqrt{k}_{\text{last}} = 45{,}000$ *and* $\sqrt{k} = 45{,}450$ *(a 1% growth). The protocol mint on a total supply* $S = 45{,}000$ *is*

$$\Delta s_{\text{prot}} = 45{,}000 \cdot \frac{450}{5 \cdot 45{,}450 + 45{,}000} = 45{,}000 \cdot \frac{450}{272{,}250} = 74.38 \text{ LPT}.$$

This is 0.1653% *of* S*, and the total LP-value growth is* 1%*; the protocol captured* $0.1653/1 = 16.53\% \approx 1/6$ *of the fee income, as expected.*

Example 13.2 (Why 1/6)**.** *The protocol-fee switch is nominally a* 1/6 *of swap fees, but the formula achieves this only asymptotically because* $\sqrt{k}$ *is a concave function of* k*: a* 1% *growth in* k *is* $\approx 0.5\%$ *in* $\sqrt{k}$*, so mint-equivalent issuance relative to total fee income approaches* 1/6 *rather than the face-value* 1/5 *that a naive formula with denominator* $5\sqrt{k}$ *would produce. The correction term* $+\sqrt{k}_{\text{last}}$ *in the denominator is precisely what enforces the* 1/6 *limit.*

Example 13.3 (Two-event reconciliation in LP-denominated units)**.** *This is an end-to-end reconciliation of the* 1/6 *share in LP-token-denominated units across a full swap-volume-then-mint cycle. The pool starts at the canonical state* $(x, y, S) = (1000,\ 2{,}025{,}000,\ 45{,}000)$ *with* $\sqrt{k}_{\text{last}} = 45{,}000$ *and the protocol fee* on*.*

Event (i): swap volume accrues fees into reserves. Assume a sequence of small symmetric swaps raises $\sqrt{k}$ *from* 45,000 *to* 45,090 *(a* 0.20000% *growth in* $\sqrt{k}$*, equivalently* $\Delta k/k \approx 0.40040\%$*). No mint or burn has occurred, so* $\sqrt{k}_{\text{last}}$ *is still* 45,000 *and* S *is unchanged at* 45,000*.*

Event (ii): Eve triggers a mint on top of the fee-grown state. Eve deposits on-ratio liquidity. At the first line of the mint handler, before any new LP tokens are minted to Eve, the protocol-fee branch fires:

$$\Delta s_{\text{prot}} \;=\; S \cdot \frac{\sqrt{k} - \sqrt{k}_{\text{last}}}{5\sqrt{k} + \sqrt{k}_{\text{last}}} \;=\; 45{,}000 \cdot \frac{90}{5 \cdot 45{,}090 + 45{,}000} \;=\; 45{,}000 \cdot \frac{90}{270{,}450} \;=\; 14.9751 \text{ LPT}.$$

The total supply increments to $S' = 45{,}014.9751$ *and* $\sqrt{k}_{\text{last}}$ *is updated to* 45,090*.*

Reconciliation. In LP-denominated terms, the total fee-driven value growth on the pre-mint *supply is the factor* $\sqrt{k}/\sqrt{k}_{\text{last}} - 1 = 45{,}090/45{,}000 - 1 = 0.002000$*, i.e.* 0.2000%*. Expressed in pre-mint units this is* $45{,}000 \cdot 0.002000 = 90.000$ *LP-equivalent units. The protocol captures* 14.9751 *of those* 90.000 *units, a share of* $14.9751/90.000 = 0.16639 = 16.639\%$*, matching the closed-form prediction* $1/(6+5\varepsilon) = 1/(6+5\cdot 0.002) = 1/6.010 = 0.16639$*. At leading order in* ε *this is exactly* 1/6*; the* 0.017 *percentage-point shortfall from* $1/6 = 16.667\%$ *is the* $O(\varepsilon)$ *correction that vanishes as swap volume between events becomes infinitesimal.*

Effective LP yield reduction. Existing LPs own $S/S' = 45{,}000/45{,}014.9751 = 99.9667\%$ *of the new supply, so their realized per-token growth is*

$$\frac{\sqrt{k}/\sqrt{k}_{\text{last}}}{S'/S} - 1 \;=\; \frac{1.002000}{1.000333} - 1 \;=\; 0.001667 \;=\; 0.1667\%.$$

That is 5/6 *of the* 0.2000% *gross fee growth, consistent with a* 1/6 *haircut taken by the protocol.*

Example 13.4 (Large-growth reconciliation: the $O(\varepsilon)$ correction)**.** *Repeating Example 13.3 with a much larger gap between events, let* $\sqrt{k}_{\text{last}} = 45{,}000$ *and suppose swap volume pushes* $\sqrt{k}$ *all the way to* 49,500 *before the next liquidity event (a* 10% *jump in* $\sqrt{k}$*). Then*

$$\Delta s_{\text{prot}} = 45{,}000 \cdot \frac{4500}{5 \cdot 49{,}500 + 45{,}000} = 45{,}000 \cdot \frac{4500}{292{,}500} = 692.3077 \text{ LPT},$$

a 1.5385% *dilution. The pre-mint gross growth in LP-equivalent units is* $45{,}000 \cdot 0.10 = 4500$*, so the captured share is* $692.3077/4500 = 15.385\% = 2/13$*, below the asymptotic* $1/6 \approx 16.667\%$*. The closed form matches:* $1/(6 + 5 \cdot 0.1) = 1/6.5 = 2/13$*. The protocol is intentionally under-compensated when LPs let fees compound for a long time between mint events, because the quadratic term in* $\varepsilon/(6 + 5\varepsilon)$ *favors LPs. In practice* $\sqrt{k}_{\text{last}}$ *is refreshed every liquidity event, so* ε *is small and the* $1/6$ *share is close to exact.*

Why this matters in production. Protocol-fee accounting in v2 is bundled into the mint/burn path, which means (i) the $\sqrt{k}_{\text{last}}$ snapshot becomes stale if no liquidity event occurs for a long time, and (ii) the protocol's realized take is a function of the cadence of LP mint/burn events, not only of swap volume. A DAO treasury that wants predictable protocol-fee revenue should therefore seed a small periodic mint/burn loop (or rely on organic LP churn) to refresh the snapshot. Turning the switch on also mints *new* LP tokens to the fee recipient rather than transferring reserves, which means the protocol-fee module composes trivially with every secondary use of the LPT token.

14 Flash swaps

Beginner note. The pair contract allows optimistic output: a caller can request Δy_{out} first, receive a callback during which they do arbitrary computation, and only then be required to deposit enough input to keep k non-decreasing. If they return both Δx_{in} and Δy_{out} (net-zero), they pay only the φ-fee on the larger leg.

Proposition 14.1 (Flash swap accounting)**.** *If a flash-swap caller borrows* Δx_{out} *and* Δy_{out}*, then within the callback must deposit* $(\Delta x_{\text{in}}, \Delta y_{\text{in}})$ *such that*

$$(x - \Delta x_{\text{out}} + \Delta x_{\text{in}})(y - \Delta y_{\text{out}} + \Delta y_{\text{in}}) \;\geq\; k,$$

with the fee retention γ *applied to the* non-zero *components of input amount. A common special case is arbitrage repayment in-kind:* $\Delta x_{\text{out}} = 0$*,* $\Delta y_{\text{out}} > 0$*,* $\Delta y_{\text{in}} = \Delta y_{\text{out}}$ *(exact repayment), requiring* $\Delta y_{\text{in}} \geq \Delta y_{\text{out}} + \varphi/\gamma$ *of additional token.*

Example 14.1 (Flash loan of 100,000 USDC, repaid in kind)**.** *Caller requests* $\Delta y_{\text{out}} = 100{,}000$ *USDC and repays entirely in USDC during the callback. The invariant check after the callback is* $(x)(y - 100{,}000 + \Delta y_{\text{in}}) \geq k\cdot$ *rescaling; solving for the minimum* Δy_{in} *gives*

$$\Delta y_{\text{in}} \;\geq\; 100{,}000 \cdot \frac{1}{\gamma} \;=\; 100{,}300.9 \text{ USDC},$$

i.e. the 0.30% *fee applies to the flash loan amount, exactly as to a "real" swap input. This example is what makes v2 flash loans extraordinarily cheap relative to dedicated flash-loan protocols that charge* 0.09% *or more.*

Example 14.2 (Cross-token arbitrage with explicit feasibility check)**.** *Suppose the canonical pool is* $(x_1, y_1) =$ (1000, 2,025,000) *at spot price* 2025*, and a secondary* CTN/USDC *venue off-chain quotes* CTN *at* 2050 USDC *on a marketable bid of at least* 10 CTN *depth. An arbitrageur borrows* $\Delta x_{\text{out}} = 10$ CTN *from the v2 pool via a flash swap, sells the* 10 CTN *for* 20,500 USDC *on the secondary venue during the callback, and returns* USDC *to the v2 pool.*

Constraint. At the callback return, the v2 pair checks the adjusted invariant

$$(x_1 - \Delta x_{\text{out}})(y_1 + \gamma\, \Delta y_{\text{in}}) \;\geq\; k_1 \;=\; 2{,}025{,}000{,}000,$$

where the fee retention $\gamma = 0.997$ *is applied to the* incoming *token because that is the side the pool treats as a swap input. Solving for the minimum required* Δy_{in}*:*

$$\Delta y_{\text{in}} \;\geq\; \frac{1}{\gamma}\Big(\frac{k_1}{x_1 - \Delta x_{\text{out}}} - y_1\Big) \;=\; \frac{1}{0.997}\Big(\frac{2{,}025{,}000{,}000}{990} - 2{,}025{,}000\Big) \;=\; \frac{20{,}454.545\ldots}{0.997} \;=\; 20{,}515.09 \text{ USDC}.$$

This matches Example 7.4's exact-out sell required input (up to the wei-ceiling rounding), as it must: a flash borrow of 10 CTN *with* USDC *repayment is operationally indistinguishable from an exact-out sell of* 10 CTN.

Profit accounting. The arbitrageur collected 20,500 USDC *on the secondary venue and must deposit at least* 20,515.09 USDC *into the pool—a shortfall of* 15.09 USDC*. The opportunity is* not *feasible: the secondary venue's* 2050 *price beats the v2 spot by only* 25 USDC *per* CTN*, but the v2 swap cost (curvature +* 0.30% *fee) over* 10 CTN *is* 51.6 USDC *per* CTN *in gross terms, and the arbitrageur's flash-swap attempt reverts for insufficient* USDC *input. The correct size is found by the break-even condition on* Δx*:*

$$2050 \cdot \Delta x \;=\; \frac{1}{\gamma}\Big(\frac{k_1}{x_1 - \Delta x} - y_1\Big) \quad\Longrightarrow\quad \Delta x \approx 5.944 \text{ CTN},$$

at which size the USDC *proceeds on the secondary venue exactly pay the pair's adjusted-invariant requirement and the arbitrageur's profit collapses to zero. A profitable flash-arb in the* 2050 *scenario therefore requires either a deeper secondary-venue bid above* 2050 *or a higher secondary price; as a practical rule of thumb the secondary price must exceed the v2 exact-out-sell average price for the attempted size.*

Feasibility invariant. The single check that determines whether the arbitrage closes is the post-callback inequality

$$k^{\text{adj}}_{\text{post}} \;=\; (x_1 - \Delta x_{\text{out}})(y_1 + \gamma \Delta y_{\text{in}}) \;\geq\; k_1.$$

A front-end simulator that evaluates this in the transaction's pre-commit phase can short-circuit any infeasible arbitrage before consuming Newton.

Why this matters in production. Flash swaps are the single most-used composability primitive in v2. Every CEX-DEX arbitrage bot, liquidation keeper, and collateral-swap Router built on v2 uses the flash-swap callback because the alternative—holding inventory across blocks—exposes the caller to MEV sandwiching and adverse selection. Production integrations should keep callback code stateless where possible, verify the adjusted-invariant inequality explicitly in the simulator before submission, and never rely on a guessed fee amount: always compute $\gamma \Delta y_{\text{in}}$ from the pair's pre-callback balance.

15 Routing across multiple pools

Beginner note. A user who wants to go from CTN to USDC could either hit the direct CTN/USDC pool or route via CTN/ETH/USDC through two pools, paying two fees. The right choice depends on the depths and spot prices of the pools.

Proposition 15.1 (Compounded fee on a two-hop route)**.** *A trade routed through two v2 pools with the same fee* φ *pays a cumulative fee multiplier of* $\gamma^2 = 0.994009$*, i.e.* 0.5991% *total, on the gross input.*

Example 15.1 (Direct vs. two-hop CTN → USDC)**.** *Direct: at* $x = 1000$*,* $y = 2{,}025{,}000$*, a* 10 *CTN sell yields* 19,989.95 *USDC (Example 7.1).*

Two-hop: suppose a CTN/ETH *pool has reserves* (100, 50) *at price* 0.5 ETH/CTN*, and an* ETH/USDC *pool has reserves* (1000, 4,000,000) *at price* 4000 USDC/ETH*. The implied cross-rate is* $0.5 \cdot 4000 = 2000$ USDC/CTN*, slightly below the direct pool. First hop:* 10 *CTN in, effective* 9.97*, out* $50 \cdot 9.97/(100 + 9.97) = 4.5330$ *ETH. Second hop:* 4.5330 *ETH in, effective* 4.5194*, out* $4{,}000{,}000 \cdot 4.5194/(1000 + 4.5194) = 18{,}006.4$ *USDC. The two-hop route delivers* \$18,006 *vs. the direct's* \$19,990*—the thin intermediate pool's price impact more than compensates for the fee doubling. A smart router would pick direct.*

Example 15.2 (Cross-over depth for routing preference)**.** *The intermediate* CTN/ETH *pool in the previous example is "thin" at depth* $L_1 = \sqrt{100 \cdot 50} = 70.71$*. If instead the* CTN/ETH *pool had depth* $L_1' = \lambda \cdot L_1$ *for some* $\lambda > 1$ *(with reserves* $(100\lambda, 50\lambda)$ *at the same price), the first-hop price impact shrinks as* $1/\lambda$ *and the two-hop output approaches* $\gamma^2 \cdot 20{,}250 = \$20{,}128.51$ *as* $\lambda \to \infty$*. The cross-over where two-hop beats direct is found numerically: for the direct pool* (1000, 2,025,000) *delivering* \$19,989.95 *on a* 10-CTN *sell, the two-hop route with proportional scaling* λ *delivers*

$$\Delta y_{\text{out}}^{2\text{-}hop}(\lambda) \;=\; \frac{4{,}000{,}000 \cdot \gamma \cdot \Delta\text{ETH}(\lambda)}{1000 + \gamma\Delta\text{ETH}(\lambda)}, \quad \Delta\text{ETH}(\lambda) = \frac{50\lambda \cdot 9.97}{100\lambda + 9.97}.$$

Setting this equal to 19,989.95 *and solving gives* $\lambda \approx 1.58$*, i.e. the intermediate* CTN/ETH *pool must be about* 58% *deeper than the baseline before routing becomes competitive with direct for this trade size. A Router that enumerates only the direct route is leaving nothing on the table in this pool landscape, but a Router that enumerates multi-hop paths must make this comparison on every trade because the cross-over depth is trade-size-dependent.*

16 The LP lifecycle end-to-end

Beginner note. This section stitches together mint, swap accumulation, burn, fees, and IL into a single numerical story for a hypothetical LP over a period in which the price moves and then partially returns. It also highlights the single structural property that most distinguishes v2 from v3: the LP token is a fungible CRC-20, not a non-fungible position.

Fungibility and identical per-token claims (the v2→v3 pedagogical contrast). Every LPT in a given pool represents an identical pro-rata share of (x, y) and of the entire history of accrued fees captured in $\sqrt{k}$. Concretely: if Alice minted at t_1 with $\sqrt{k}_1 = 45{,}000$ and received 450 LPT, and Bob mints at a later time t_2 with $\sqrt{k}_2 = 46{,}200$ on the same pool and receives Δs_{Bob} computed by the subsequent-mint formula, then at every moment $t \geq t_2$ the two LPs' positions satisfy

$$\frac{V_{\text{Alice}}(t)}{s_{\text{Alice}}} \;=\; \frac{V_{\text{Bob}}(t)}{s_{\text{Bob}}},$$

because both LPT holders are claims on the same scalar pair $(x(t), y(t))$ and the same scalar $k(t)$. This is not a theorem that requires any per-position state: it is a direct consequence of S and the reserves being the only pool-level quantities the pair contract reads. Concretely, Bob's LPT balance at t_2 is priced exactly as Alice's LPT balance is priced at t_2; he simply paid a higher k-denominated price per LPT at entry. The v3 analogue replaces the fungible token with a per-position NFT carrying its own range $[p_a, p_b]$ and its own fee-growth snapshots, so two v3 LPs minted into the same pool generally have different per-position claims at every moment and two NFTs are not substitutable. In v2, by contrast, a pair of LPT tokens from the same pool are perfectly substitutable and can be freely transferred, composed into secondary protocols, or used as collateral with no bespoke accounting.

Transfer semantics. Because LPT is an CRC-20, `transfer` is a single storage update to `balanceOf` and does not touch the pair's $(x, y, \sqrt{k})$ state at all; fee accrual continues as before, and the transferee inherits the sender's proportional share without any snapshot or catch-up call. In v3 a position transfer is an NFT transfer that carries along a range and a `feeGrowthInsideLast` snapshot, which must be reconciled against the current f_g; in v2 there is nothing to reconcile.

Example 16.1 (A 30-day LP trajectory)**.** *Alice mints in the canonical pool with* 1% *share: she deposits* 10 *CTN and* 20,250 *USDC at* $p_0 = 2025$*, receiving* $\Delta s = 450$ *LP tokens out of new total* $S = 45{,}450$ *(consistent with Example 3.2).*

Day 1–15: price drifts up to $p = 2916$ *(factor* $\alpha = 1.44$*, so* $\sqrt{\alpha} = 1.2$*). In the absence of fees* k *is unchanged; reserves rebalance to* $x = \sqrt{k/p} = 835.82$*,* $y = \sqrt{kp} = 2{,}437{,}561.3$*. Pool-wide swap volume over the* 15 *days deposits* $\varphi \cdot V$ *fees into reserves; assume* $\sqrt{k}$ *has grown from* 45,450 *to* 45,800*.*

Day 16–30: price retraces to $p = 2025$ and finishes at $p = 2250$ (factor $\alpha = 10/9$). Further fee growth takes $\sqrt{k}$ to 46,200.

At exit, Alice burns $b = 450$ LP tokens out of $S = 45{,}450$ (no mints in the interim). Her reserves out are

$$\Delta x = 450 \cdot \frac{x}{S} = 450 \cdot \frac{\sqrt{k/p}}{45{,}450} = 450 \cdot \frac{46{,}200/\sqrt{2250}}{45{,}450} = 9.6436 \text{ CTN},$$

$$\Delta y = 450 \cdot \frac{y}{S} = 450 \cdot \frac{46{,}200 \cdot \sqrt{2250}}{45{,}450} = 21{,}700.7 \text{ USDC}.$$

Valued at $p = 2250$: $\$9.6436 \cdot 2250 + 21{,}700.7 = \$43{,}398.8$. Passive-hold value of her (10, 20,250) at $p = 2250$: $\$22{,}500 + 20{,}250 = \$42{,}750$. Net LP P&L: $+\$648.8$, a 1.52% outperformance of passive hold driven by fees overcoming the $\alpha = 10/9$ IL.

17 Worked end-to-end example with on-chain receipts

Beginner note. This section repeats a tight end-to-end cycle—mint, swap, swap, burn—with the level of precision you would see in a block explorer, so that a reader can audit every reserve update and LP balance by hand.

Example 17.1 (Explicit receipts for a mint-swap-swap-burn cycle)**.** *Initial state is*

$$(x, y, S, \sqrt{k}_{\text{last}}) = (1000,\ 2{,}025{,}000,\ 45{,}000,\ 45{,}000),$$

with protocol fee off.

Event 1: Bob mints. Deposit $(\Delta x, \Delta y) = (10,\ 20{,}250)$. Then

$$\Delta s = \min(450, 450) = 450.$$

New state is (1010, 2,045,250, 45,450, 45,000) with $\sqrt{k} = 45{,}450$. If protocol fee were on, minted protocol LP would be

$$\Delta s_{\text{prot}} = 45{,}000 \cdot \frac{450}{5 \cdot 45{,}450 + 45{,}000} = 74.38 \text{ LPT}$$

(see Example 13.1); here it is off, so no protocol mint.

Event 2: Charlie sells 10 CTN. Gross input is 10, effective input is $10\gamma = 9.97$. Output:

$$\Delta y_{\text{out}} = \frac{2{,}045{,}250 \cdot 9.97}{1010 + 9.97} = 19{,}991.904 \text{ USDC}.$$

New reserves:

$$(x, y) = (1010 + 10,\ 2{,}045{,}250 - \Delta y_{\text{out}}) = (1020,\ 2{,}025{,}258.096).$$

Hence

$$k = 1020 \cdot 2{,}025{,}258.096 = 2{,}065{,}763{,}257.743, \qquad \sqrt{k} = 45{,}450.668.$$

Compared to post-mint $\sqrt{k} = 45{,}450$, the increase of 0.668 is fee capture.

Event 3: Dan buys with 40,000 USDC. Effective input is 39,880. Output:

$$\Delta x_{\text{out}} = \frac{1020 \cdot 39{,}880}{2{,}025{,}258.096 + 39{,}880} = 19.6973 \text{ CTN}.$$

New reserves:

$$(x, y) = (1020 - \Delta x_{\text{out}},\ 2{,}025{,}258.096 + 40{,}000) = (1000.3027,\ 2{,}065{,}258.096).$$

Then

$$k = 2{,}065{,}883{,}294.069, \qquad \sqrt{k} = 45{,}451.989.$$

Event 4: Bob burns 450 *LPT. Reserves per LPT are*

$$\left(\frac{x}{S}, \frac{y}{S}\right) = \left(\frac{1000.3027}{45{,}450}, \frac{2{,}065{,}258.096}{45{,}450}\right) = (0.0220089,\ 45.44022).$$

Bob withdraws

$$\Delta x = 450 \cdot \frac{x}{S} = 9.90399, \qquad \Delta y = 450 \cdot \frac{y}{S} = 20{,}448.100.$$

At final spot $p = 2{,}065{,}258.096/1000.3027 = 2064.633$*, Bob's exit value is* $9.90399 \cdot 2064.633 + 20{,}448.100 =$ \$40,896.20*. The same notional deposit marked at final spot is* $10 \cdot 2064.633 + 20{,}250 =$ \$40,896.33*. Bob underperforms passive hold by about* \$0.13 *(*$\sim 0.0003\%$*), i.e. almost exactly flat over this short window.*

Example 17.2 (Per-LP-token value trace over the cycle)**.** *Taking the same cycle as Example 17.1, the per-*LPT *value* $V_{\text{pool}}(p)/S = 2\sqrt{kp}/S$ *evolves as follows:*

Event	S *(*LPT*)*	$\sqrt{k}$	p *(*USDC*/*CTN*)*	V_{pool}/S *(*USDC*/*LPT*)*
Initial	45,000	45,000	2025	90.00000
After mint	45,450	45,450	2025	90.00000
After sell	45,450	45,450.668	1985.547	89.11028
After buy	45,450	45,451.989	2064.633	90.87944

*The per-*LPT *value at the initial state is* $V_{\text{pool}}/S = 2 \cdot 45{,}000 \cdot \sqrt{2025}/45{,}000 = 2\sqrt{2025} = 90.00000$*. After Bob's on-ratio mint it is unchanged because mint is value-preserving per* LPT*. The sell moves the price and the fee adds* 0.668 *to* $\sqrt{k}$*; computing* $2\sqrt{kp}/S = 2 \cdot \sqrt{45{,}450.668^2 \cdot 1985.547}/45{,}450$ *gives* 89.11028*. The buy moves the price back toward the original and adds another fee increment, ending at* 90.87944*. Per-*LPT *value has grown by* $0.87944/90 = 0.9771\%$ *across the cycle, entirely attributable to fee accrual and to the terminal price drift of* $+1.956\%$ *above* p_0*: with* $\alpha = 2064.633/2025 = 1.01958$*, the baseline* $\sqrt{\alpha} = 1.00974$ *predicts a* 0.974% *move, within* 0.003% *of the observed* 0.9771%*, the remainder coming from* $\sqrt{k}$ *growth.*

18 Theoretical layer and optimality properties

Beginner note. This section steps back from specific numbers and summarizes the short list of global properties that make the constant-product design the reference AMM: it is the unique CFMM with uniform depth on $(0, \infty)$, it is path-independent between the same endpoints (in the fee-free limit), it admits a closed-form LVR computation, and it supports clean arbitrage-tracking and microstructure accounting.

A. CFMMs as feasible-set geometry

A constant-function market maker (CFMM) defines an admissible region

$$\mathcal{F} \ = \ \{(x, y) \in \mathbb{R}^2_+ : f(x, y) \geq k\}$$

with trading function f and invariant k. A swap is a move from $(x, y) \in \partial\mathcal{F}$ to $(x', y') \in \partial\mathcal{F}$ (with equality replaced by $\geq$ under fees). For v2, $f(x, y) = xy$ and the boundary is the hyperbola $xy = k$. Two properties make this geometry especially clean: $\mathcal{F}$ is convex, and the boundary is smooth, so the no-arbitrage price $p = -dy/dx$ equals the gradient ratio $\partial_x f/\partial_y f = y/x$ [4, 6]. The fee cone $\gamma < 1$ simply thickens the admissible region into $\{(x', y') : x'y' \geq k\}$, which is why $\sqrt{k}$ can only grow: every fee-charging swap moves strictly into the interior of the feasible set, raising the new boundary's k.

Proposition 18.1 (Uniqueness of constant product among symmetric CFMMs)**.** *Among constant-function market makers* $f(x, y) = k$ *that satisfy (i) positive one-homogeneity* $f(\lambda x, \lambda y) = \lambda^\rho f(x, y)$ *for some* $\rho > 0$*, (ii) spot price* $p = y/x$*, and (iii)* 50/50 *value-split at every admissible state, the only solution is* $f(x, y) = xy$ *(up to a monotone transform).*

C. Arbitrage and price tracking

In a v2 pool with external reference price $p^\star$, the no-arbitrage condition is that any rational external agent moves the pool's spot price toward $p^\star$ until the profit of a further swap is exactly eaten by the 0.30% fee. The resulting band is approximately $p \in [p^\star\gamma^{1/2},\, p^\star\gamma^{-1/2}]$, a $\pm 0.15\%$ dead-zone around $p^\star$.

Proposition 18.2 (Fee-bounded arbitrage band)**.** *If an external venue quotes a marketable price $p^\star$ with infinite depth at zero cost, the v2 pool's spot price p is confined to the interval where an exact-in arbitrage of infinitesimal size earns zero profit:*

$$\frac{p}{p^\star} \;\in\; \left[\gamma,\, \gamma^{-1}\right] \;=\; [0.997,\, 1.003009\ldots],$$

i.e. $|p - p^\star|/p^\star \le 0.3009\%$ across any block boundary at which arbitrageurs have free entry [2, 3].

Example 18.1 (Numerical band at $p^\star = 2025$)**.** *For $p^\star = 2025$, the pool's spot price can drift to as low as $p = 2025 \cdot 0.997 = 2018.925$ before any exact-in sell is profitable to an arbitrageur, and as high as $p = 2025/0.997 = 2031.094\ldots$ before any exact-in buy is profitable. The mid-band of* 6.17 USDC *per* CTN *is the arbitrageur's effective collection for the round-trip of aligning the pool, and is the direct source of LP fee income under realistic external-venue quoting.*

D. LP payoff structure

Proposition 18.3 (Path independence)**.** *For a fee-free v2 pool, the realized output of a trade from state (x_1, y_1) to state (x_2, y_2) depends only on the endpoint states, not on the intermediate path (e.g. arbitrary partition into sub-trades).*

The LP's terminal payoff against a passive hold is the impermanent-loss curve of §6. When the pool earns fees between entry and exit, the LP's realized payoff is $V(p_1) \cdot (\sqrt{k_1}/\sqrt{k_0})$; the factor $\sqrt{k_1}/\sqrt{k_0}$ is the multiplier one would apply to an otherwise-passive V-position to reflect accrued fees. The break-even condition against a passive hold with terminal price factor α is $\sqrt{k_1}/\sqrt{k_0} \ge (1+\alpha)/(2\sqrt{\alpha})$, i.e. the cumulative fee multiplier must offset the IL factor.

E. Loss-versus-rebalancing (LVR)

Proposition 18.4 (LVR for constant product, constant price volatility)**.** *Under a geometric Brownian motion price with volatility σ, the instantaneous loss-versus-rebalancing rate for a v2 LP is* $\mathrm{LVR}(t) = \frac{1}{8}\sigma^2\, V(t)$*, i.e. the position loses $\sigma^2/8$ per unit time to arbitrageurs against an optimally rebalanced benchmark [7].*

Example 18.2 (LVR at $\sigma = 80\%$ annual)**.** *With $\sigma^2 = 0.64$ per year, LVR $= 0.08$ per year, i.e. 8% of position value annually as the "cost" of being an LP against a hypothetical perfectly-rebalancing counter-party. Swap fees must exceed 8% per year for the LP to beat LVR at this volatility. For the canonical pool, at \$10M daily volume and $\varphi = 0.30\%$, annual fees are \$11M; measured against TVL \$4.05M at $p_0 = 2025$ the fee yield is $\sim 273\%$ per year, far in excess of LVR at this volatility. Thinner pools with lower volume-to-TVL ratios, or higher-volatility asset pairs, can easily lose this race, which is the quantitative argument for the v3 generalization at high-σ pairs.*

F. Microstructure and MEV

Proposition 18.5 (One-block oracle lag)**.** *The v2 TWAP accumulator records the pre-block marginal price multiplied by the block duration, so any within-block price movement is invisible to oracle consumers until the* next *block. An attacker who wishes to manipulate the TWAP over a window of W blocks must sustain a non-equilibrium price for at least W consecutive blocks, each of which exposes the attacker to arbitrage risk and inventory drain.*

Example 18.3 (Manipulation cost lower bound)**.** *Consider an attacker who wants to push a $\Delta\tau = 30$-minute TWAP upward by 10%. In the canonical pool, maintaining a price of $p = 2227.5$ for 30 minutes against a $p^\star = 2025$ external market invites continuous arbitrage: each arbitrageur can buy* CTN *at the external*

venue at 2025 *and sell into the v2 pool at above* 2200*. The per-block arbitrage drain on the attacker's capital is bounded below by* $(\eta - \varphi) \cdot V_{\text{block}}$ *where* V_{block} *is the per-block arbitrage volume, and over* 30 *minutes of* ~ 150 *blocks this accumulates to tens of percent of the pool's TVL. The numerical lesson is that v2 TWAP is economically secure for any* $W \gg 1$ *block, but the single-block readings are manipulable [10, 8, 9].*

Why this matters in production. These theoretical anchors tell a consistent story: v2 is cheap, composable, and well-analyzed, but it is not free. LVR sets a volatility ceiling above which a pool of fixed φ will underperform, the arbitrage band sets a fee floor on LP income, and the TWAP lag sets a window floor on secure oracle consumption. A mature protocol that builds on v2 should respect each of these in its design: do not lend against v2 TWAP snapshots shorter than several blocks, do not expect LPs in high-volatility pairs to earn positive net returns at $\varphi = 0.30\%$, and do not expect internal pool prices to deviate from external reference prices by more than the $\pm 0.30\%$ band.

19 Edge cases and boundary behavior

Beginner note. Three boundary behaviors show up in v2 audits: a trade that would empty a reserve, a mint that attempts to bootstrap a pool with one-sided liquidity, and a sub-wei rounding at a single-wei trade. Each is small but worth the careful handling baked into the pair contract.

Example 19.1 (Emptying a reserve is impossible)**.** *A trader attempts* $\Delta x_{\text{out}} = 1000$ *CTN from the canonical pool. The exact-out formula gives* $\Delta y_{\text{in}} = \lceil 2{,}025{,}000 \cdot 1000/(0.997 \cdot 0) \rceil = \infty$*: no finite USDC payment can drain the pool, because the hyperbola never reaches the axis. The v2 pair reverts with an over/underflow before the trader ever sees a quote. This is the curve's asymptotic protection of LPs: reserves stay strictly positive under the constant-product law.*

Example 19.2 (First mint with $\Delta x \Delta y < 10^6$ wei^2)**.** *If Alice tries to bootstrap with* $\Delta x = 500$ *wei,* $\Delta y = 500$ *wei, then* $\sqrt{\Delta x \Delta y} = 500 < 1000 =$ `MINIMUM_LIQUIDITY`*, and the mint reverts. This forces the bootstrap to be at least* 10^6 *wei*^{2} *in geometric mean, which rules out adversarial pools seeded with* 1 *wei and* 10^{18} *wei used to inflate LP token prices.*

Example 19.3 (One-wei exact-out rounding)**.** *A trader requests* $\Delta x_{\text{out}} = 1$ *wei of CTN (i.e.* 10^{-18} *CTN). The required input is* $\Delta y_{\text{in}} = \lceil 2{,}025{,}000 \cdot 10^{18} \cdot 1/(0.997 \cdot (10^{21} - 1)) \rceil \approx \lceil 2031.09 \rceil = 2032$ *USDC-wei. The rounding adds under a USDC-wei of over-collection, which accumulates to LPs across many trades as the "phantom liquidity" described in Appendix B.*

Example 19.4 (Fee-on-transfer token interaction)**.** *A fee-on-transfer (FOT) token debits a fixed percentage* τ_{FOT} *from every* `transfer` *or* `transferFrom` *call, so that a caller who instructs "send* 100*" to the pair sees only* $100 \cdot (1 - \tau_{\text{FOT}})$ *arrive at the pair's balance. The v2 pair contract computes reserves via a* `_update` *call that reads* `balanceOf(pair)` *after the caller's* `safeTransfer`*; it does* not *trust the amount the caller claimed to send. Two failure modes follow.*

(i) Swap input under-credited. Suppose a trader attempts an exact-in swap of 1000 *FOT-token units (*$\tau_{\text{FOT}} = 1\%$*). The Router computes* Δy_{out} *assuming the pair will see* 1000 *units of input; the pair actually sees* 990*, so the post-swap invariant check* $(x + 990)(y - \Delta y_{\text{out}}) \geq k$ *fails by a factor related to the missing* 10 *units, and the transaction reverts with* `Pair: K`*. The canonical remedy is to call the Router's* `swapExactTokensForTokensSupportingFeeOnTransferTokens` *entry point, which reads the pair's balance* after *the transfer to determine the actual input, then computes* Δy_{out} *from that.*

(ii) Mint over-credited, burn under-credited. On a mint, the pair reads its own balance to infer $\Delta x, \Delta y$*, so an FOT token's transfer fee effectively donates* τ_{FOT} *of every LP's contribution to the pool without any offsetting LP-token grant; the initial LP is always short-changed. On a burn, the pair sends the proportional amount to the LP and the LP receives* $(1 - \tau_{\text{FOT}})$ *times the expected balance, an additional one-time haircut. Taken together these two one-way leaks turn an FOT-token pool into a slowly-draining vault for anyone who holds* LPT *across many mint-burn cycles.*

Operational guidance. The standard workaround, predating v2 by several years, is to (a) use `SafeCRC20`*'s* `safeTransfer/safeTransferFrom` *wrappers to catch non-standard return-value behavior and (b) on any pair*

whose token list includes an FOT token, route swaps only through the Router's FOT-aware entry points. LPs who hold such pairs should account for the systematic drag in their realized-yield calculations and should not expect $\sqrt{k}$ growth to be a pure fee signal. No retrofit of the pair contract itself was ever made in v2; this is purely a periphery concern.

Why this matters in production. Asymptotic reserve protection, minimum-liquidity burn, and one-wei rounding are three simple mechanisms that together ensure the invariant $x'y' \geq k$ holds at every block, regardless of input size or adversarial intent. A robust integration (i) never assumes a pool can be emptied, (ii) verifies that the first mint exceeds the minimum-liquidity threshold, and (iii) documents the one-wei-in-favor-of-pool convention for any downstream accounting.

Closing remark. Three drivers make the v2 design the reference constant-product AMM: the invariant $x \cdot y = k$ that collapses all pricing into one hyperbola, the fee-in-reserve convention that makes $\sqrt{k}$ monotone and obviates per-LP fee bookkeeping, and the fungibility of the CRC-20 LP token that renders positions composable without any range metadata. The worked examples above exhibit each of these mechanics end-to-end in numbers that reproduce exactly by hand.

A Proofs of key identities

Beginner note. These proofs justify formulas used earlier. If you only need operational understanding, you can skim this section and return later when you want mathematical confidence.

Proposition A.1 (Exact-in swap output)**.** *For an exact-in sell of Δx_{in} with fee retention $\gamma = 1 - \varphi$, the output is*

$$\Delta y_{\text{out}} = \frac{y \cdot \gamma \Delta x_{\text{in}}}{x + \gamma \Delta x_{\text{in}}},$$

and the new invariant satisfies $k' = k + y \cdot (1 - \gamma)\Delta x_{\text{in}} \geq k$.

Proof. The invariant after the trade is $(x + \Delta x_{\text{in}})(y - \Delta y_{\text{out}}) \geq k$, with equality on the fee-adjusted portion: $(x + \gamma \Delta x_{\text{in}})(y - \Delta y_{\text{out}}) = k$. Solve for Δy_{out} to obtain the stated formula. Then $k' = (x + \Delta x_{\text{in}})(y - \Delta y_{\text{out}}) = k + (1 - \gamma)\Delta x_{\text{in}}(y - \Delta y_{\text{out}})$, and substituting $y - \Delta y_{\text{out}} = k/(x + \gamma \Delta x_{\text{in}})$ gives the stated increment, non-negative since $\gamma \leq 1$. □

Proposition A.2 (Exact-out required input)**.** *For a requested $\Delta y_{\text{out}} < y$, the required input is*

$$\Delta x_{\text{in}} = \left\lceil \frac{x \cdot \Delta y_{\text{out}}}{\gamma (y - \Delta y_{\text{out}})} \right\rceil.$$

Proof. Set $(x + \gamma \Delta x_{\text{in}})(y - \Delta y_{\text{out}}) \geq k$ with equality up to the single wei of rounding; solve for Δx_{in} and take the ceiling in favor of the pool. □

Proposition A.3 (Position value formula)**.** *At spot price p in a pool with invariant k, the reserves are $x(p) = \sqrt{k/p}$, $y(p) = \sqrt{kp}$, and the total pool value is*

$$V_{\text{pool}}(p) = x(p)p + y(p) = 2\sqrt{k\,p}.$$

Proof. From $xy = k$ and $p = y/x$, $x = \sqrt{k/p}$ and $y = \sqrt{kp}$. Then $xp + y = \sqrt{kp} + \sqrt{kp} = 2\sqrt{kp}$. □

Proposition A.4 (Impermanent-loss closed form)**.** *For a deposit at p_0 and evaluation at $p = \alpha p_0$ with k unchanged,*

$$\text{IL}(\alpha) = \frac{2\sqrt{\alpha}}{1 + \alpha} - 1 \leq 0,$$

with equality iff $\alpha = 1$.

Proof. $V(p) = 2\sqrt{kp} = 2\sqrt{\alpha} \cdot \sqrt{kp_0}$ and $W(p) = x_0 p + y_0 = y_0(1+\alpha)$. Using $\sqrt{kp_0} = y_0$, $V/W = 2\sqrt{\alpha}/(1+\alpha)$. By AM-GM, $1 + \alpha \geq 2\sqrt{\alpha}$ with equality iff $\alpha = 1$. □

Proposition A.5 (Protocol-fee 1/6 limit)**.** *As $\varepsilon = (\sqrt{k} - \sqrt{k}_{\text{last}})/\sqrt{k}_{\text{last}} \to 0$, the protocol-fee mint formula satisfies $\Delta s_{\text{prot}}/(S\varepsilon) \to 1/6$.*

Proof. Write $\sqrt{k} = \sqrt{k}_{\text{last}}(1 + \varepsilon)$. Then

$$\frac{\Delta s_{\text{prot}}}{S} = \frac{\sqrt{k} - \sqrt{k}_{\text{last}}}{5\sqrt{k} + \sqrt{k}_{\text{last}}} = \frac{\sqrt{k}_{\text{last}}\varepsilon}{\sqrt{k}_{\text{last}}(5(1+\varepsilon)+1)} = \frac{\varepsilon}{6+5\varepsilon} \to \frac{\varepsilon}{6}.$$

□

B UQ112.112 fixed-point arithmetic

Beginner note. Smart contracts cannot store floating-point numbers safely, so they use fixed-point integers. This appendix explains how prices are encoded in the accumulators and why the 112/112 split interacts with the 112-bit reserve width.

On-chain, prices in the accumulators are stored in UQ112.112 fixed-point: an unsigned 224-bit integer $\tilde{p} = p \cdot 2^{112}$, with 112 integer bits and 112 fractional bits. The representation has:

- **Resolution.** $\Delta p = 2^{-112} \approx 1.93 \cdot 10^{-34}$, far below any meaningful price quantum.
- **Range.** $p \in [2^{-112},\, 2^{112} - 2^{-112}]$, i.e. p between $\sim 10^{-34}$ and $\sim 5.19 \cdot 10^{33}$. Every realistic asset pair fits.
- **Per-block increment.** One block with $p = P$ and duration $\Delta\tau$ adds $P \cdot 2^{112} \cdot \Delta\tau$ to the accumulator. With $P < 2^{112}$ and $\Delta\tau < 2^{32}$, the per-block increment is below 2^{256}; the accumulator itself is stored modulo 2^{256} and overflow is transparent.
- **Consumer arithmetic.** A consumer reads $(c_0^{t_1}, \tau_1)$ and $(c_0^{t_2}, \tau_2)$, computes $(c_0^{t_2} - c_0^{t_1}) \bmod 2^{256}$ and $(\tau_2 - \tau_1) \bmod 2^{32}$, and divides to obtain the TWAP in UQ112.112; a final right-shift by 112 bits yields the plain-integer price.

Example B.1 (Encoding $p = 2025$)**.** $\tilde{p} = 2025 \cdot 2^{112} = 2025 \cdot 5.1923 \cdot 10^{33} = 1.0515 \cdot 10^{37}$*, which fits in* 224 *bits. A* 1*-second block at this price adds* $1.0515 \cdot 10^{37}$ *to the accumulator. Over a year (*$\sim 3.15 \cdot 10^7$ *seconds) the accumulator grows by* $\sim 3.3 \cdot 10^{44}$*, still well below the* $2^{256} \approx 1.16 \cdot 10^{77}$ *wrap.*

Rounding policy. The pair contract consistently rounds exact-out inputs *up* by at most one wei, which guarantees the invariant $x'y' \geq k$ is preserved across every discrete step. Over many swaps this accumulates into vanishingly small "phantom liquidity" that belongs to the LPs as a whole but cannot be attributed to any single swap; it is commonly left uncollected and simply raises $\sqrt{k}$ in perpetuity.

References

[1] H. Adams, N. Zinsmeister, and D. Robinson, *Centurion v2 Core*, March 2020. Technical whitepaper. Available at: `https://app.centurion.org/whitepaper.pdf`.

[2] G. Angeris, H.-T. Kao, R. Chiang, C. Noyes, and T. Chitra, *An Analysis of Centurion Markets*, arXiv:1911.03380, 2019.

[3] G. Angeris and T. Chitra, *Improved Price Oracles: Constant Function Market Makers*, arXiv:2003.10001, 2020.

[4] G. Angeris, A. Agrawal, A. Evans, T. Chitra, and S. Boyd, *Constant Function Market Makers: Multi-Asset Trades via Convex Optimization*, arXiv:2107.12484, 2021.

[5] G. Angeris, A. Evans, and T. Chitra, *Replicating Market Makers*, arXiv:2103.14769, 2021.

[6] G. Angeris, T. Chitra, T. Diamandis, A. Evans, and K. Kulkarni, *The Geometry of Constant Function Market Makers*, arXiv:2308.08066, 2023.

[7] J. Milionis, C. C. Moallemi, T. Roughgarden, and A. L. Zhang, *Automated Market Making and Loss-Versus-Rebalancing*, arXiv:2208.06046, 2022 (revised 2024).

[8] L. Zhou, X. Xiong, J. Erway, B. Feng, C. Wang, and M. Wang, *High-Frequency Trading on Decentralized On-Chain Exchanges*, arXiv:2009.14021, 2020.

[9] L. Zhou, X. Xiong, J. Easton-Calabria, D. Perez, K. Qin, and A. Gervais, *SoK: Decentralized Finance (DeFi) Attacks*, arXiv:2106.09872, 2021.

[10] A. Adams, B. Y. Chan, S. Markovich, and X. Wan, *Don't Let MEV Slip: The Costs of Swapping on the Centurion Protocol*, arXiv:2309.13648, version updated 2024.

[11] Centurion Labs, *Centurion v2 Documentation: Pair, Router, and Oracle*, Available at: `https://docs.centurion.org/`.