ChatGPT

Design for Fractional Mortgage Ownership Tracking with Formance Ledger

Overview of Requirements

You need to track fractional ownership of private mortgages (up to 10 investors per mortgage, with ownership percentages precise to two decimal places) and handle interest-only payment distributions to those investors. Ownership stakes can change hands mid-period, so interest for a given month may need to be split between the original owner (for the days they held the stake) and the new owner (for the remainder of the period). Payments are collected monthly via Rotessa (which provides webhooks/API for confirmation of incoming funds), and disbursals to investors are done off-platform (e.g. via wire or e-transfer) with manual confirmation. The system should use Formance Ledger to record all ownership changes and money movements for audit and real-time balance tracking. The key challenges are:

- **Fractional ownership tracking:** Representing and updating a "cap table" of owners and their percentages for each mortgage, ensuring the total equals 100% at all times.
- **Mid-period ownership transfers:** Determining how to prorate the monthly interest payment between sellers and buyers when a transfer occurs mid-cycle.
- **Interest payment distribution:** Splitting each interest payment among the current (and former) owners appropriately and recording those transactions.
- **Off-platform payout confirmation:** Integrating a process to confirm when investors have actually been paid, and reflecting that in the ledger or system state.

We will use a Socratic approach to break down each aspect and design a solution that leverages Formance Ledger's capabilities as much as possible for accuracy and auditability.

Using Formance Ledger for Fractional Ownership

How do we model fractional mortgage shares in the ledger? Formance Ledger allows custom assets and multi-account transactions, which we can exploit to represent ownership stakes. A recommended approach is to treat each mortgage's ownership as a **fungible asset** within the ledger. For example, when a new mortgage `M123` is created, define a custom asset (e.g. symbol `M123SHARE`) representing 100% ownership of that mortgage. We can issue a fixed supply of this asset (say 10,000 units to represent 100.00%) and allocate those units to accounts for each owner according to their percentage. Formance supports arbitrary assets (not just currencies), so you can create assets like `M123SHARE` on the fly [1]. Using an asset to denote ownership has several benefits: it provides an auditable history of transfers, ensures that ownership sums to 100% (since units are conserved in transfers), and makes it easy to query who holds how many units at any time (effectively the cap table).

Ledger accounts setup: We will create ledger accounts for each investor and each mortgage. For instance:

- **Investor accounts:** Each investor (user) has an account (e.g. `investor:USER123`) for holding funds (cash balances) and can also hold mortgage share units.
- **Mortgage accounts:** Each mortgage might have a collection account for payments (e.g. `mortgage:M123:collect` to temporarily hold incoming interest before distribution). Additionally, an account can be used as the **issuance/treasury** for the mortgage share asset (e.g. `mortgage:M123:shares`). Initially, this account holds the total supply of `M123SHARE` units, which will be distributed to owners.

Initial ownership issuance: When a mortgage is created (funded), you would credit the initial owner with 100% of the `M123SHARE` asset. For example, if FairLend initially owns 100%, credit FairLend's account with 10,000 units of `M123SHARE` and debit the `mortgage:M123:shares` treasury account by 10,000 (reflecting that those shares have been issued to FairLend). This transaction establishes the initial cap table entry on the ledger. (In double-entry terms, the treasury account could start at 0 and go to -10000 units, representing an issuance liability, or you pre-populate it with 10000 and transfer out – either way, the ledger will record the transfer as the origin of shares.)

Ownership transfers: When an investor sells a fractional stake to another, record this as a **ledger transaction transferring the share asset** from the seller's account to the buyer's account. Because Formance Ledger supports **atomic multi-posting transactions**, you can swap assets between two parties in one operation if needed (for instance, if the sale involves a payment) [2]. However, if payments for the stake sale happen off-platform, you might simply record the ownership change as a transfer of the `M123SHARE` units (and optionally record the cash movement externally or as a separate metadata entry). The key is that after the transfer, the ledger's balances for `M123SHARE` units reflect the new cap table (e.g. seller's share units decrease, buyer's increase). Because these transfers are atomic, the total units in circulation remain 10,000 and you never exceed 100% ownership at any time – the ledger naturally enforces the invariant that ownership percentages sum to 100%. In other words, overlapping or double-allocated ownership cannot occur due to the transactional integrity of the ledger (the transfer either happens in whole or not at all).

By using the ledger in this way, you gain an **immutable, audited log** of every ownership change [3]. At any point, you can query the ledger for who holds how many `M123SHARE` units (i.e. each investor's current percentage) and even reconstruct historical ownership by reviewing past transactions. This satisfies the requirement that "Formance ledger should absolutely reflect every ownership change" – it becomes the source of truth for fractional ownership.

Handling Mid-Period Ownership Changes and Interest Accrual

The challenge: If an ownership transfer happens in the middle of an interest payment period, the monthly interest needs to be prorated between the former owner and the new owner based on how many days each held the stake. For example, if Owner A sells a 20% stake to Owner B halfway through the month, Owner A should receive roughly 50% of that stake's monthly interest (for the days they owned it) and Owner B receives the remaining 50% for that month. The difficulty is that by the time the payment is made (end of month), Owner A no longer appears as a current owner (having sold their stake). We need to ensure Owner A still gets their portion of the interest.

Option 1: Calculate prorated shares at distribution time (simpler approach). One way to handle this is to **compute the split externally when the payment is received**, using the historical ownership data. The ledger will give you the timeline of ownership: for each ownership transfer, you know the date it happened and the percentage transferred. The system (outside the ledger) can track the last payment date and the sale date to calculate how many days each owner held their stake in the period. Then, when an interest payment comes in, the backend can determine "Owner A gets X amount, Owner B gets Y amount" based on the fraction of the period each owned the stake. After computing these amounts, you would create **distribution transactions in the ledger** to credit those amounts to the respective owners (even if one of them is no longer a current owner).

In practice, you might maintain a record of ownership periods in your database for each mortgage (e.g. a list of intervals or a history of cap table changes with dates). Using that, you can split the interest payment accordingly. This logic can live in a Convex function or server component: on receiving a Rotessa payment webhook (indicating the monthly payment cleared), fetch the mortgage's ownership history for that month, compute each investor's share of the payment based on their percentage *and* the portion of the period they held it. Then proceed to ledger distribution (described in the next section).

This approach offloads the date arithmetic and proration logic to your code, which is flexible and easier to adjust. The Formance Ledger is then used to record the final calculated payouts to each party, ensuring a clear audit trail of who got paid what.

Option 2: Track accruals in the ledger (advanced approach). If you want to push more of the burden onto Formance and reduce external calculations, you could model **interest accrual transactions within the ledger** whenever ownership changes occur. For example, at the moment of a sale transfer, you could immediately allocate the interest accrued so far in the period to the seller in the ledger. This would involve posting a transaction that represents the accrued interest from the period start to the sale date: credit the seller's account with their accrued interest amount and debit a "mortgage interest payable" account. Essentially, you're recording a liability to the seller at sale time. The buyer would then only accrue interest from that point onward. In effect, the ledger would maintain running balances of "accrued interest owed" to each party as the period progresses. At the end of the month, when the payment is received, the ledger could then simply settle that liability.

While conceptually elegant, this method is more complex to implement. You'd need to calculate the prorated interest at the exact time of transfer (which means knowing how much interest per day, etc.), and possibly leverage ledger's support for time-based transactions or metadata. Formance Ledger does support **bi-temporality** (having effective dates on transactions) and real-time balance queries at any point in time [4], so in theory you could issue transactions with an effective date equal to the sale date. However, implementing this requires careful use of Formance's API and Numscript DSL to avoid inconsistencies.

For a first implementation, **it's recommended to use Option 1 (external proration calculation)** for clarity and simplicity. This keeps the ledger usage straightforward (recording final outcomes), and you can rely on your application logic to handle the date calculations. The ledger still provides real-time balances and ensures no double-counting of ownership, but you don't force it to calculate time-based splits internally. Option 2 could be a future enhancement if you need the ledger to become even more autonomous in tracking who is owed interest for what period.

(Aside: Because all payments are interest-only and the principal never amortizes, the monthly interest amount is predictable (e.g., `interest = principal * rate / 12`*). Prorating by days is a reasonable approach to split that interest. Make sure to clarify whether to use actual days in the month or a fixed 30-day month for proration, for consistency.)*

Interest Payment Collection and Distribution Flow

Now, let's design the flow for handling an interest payment from collection through distribution using Formance Ledger:

1. **Payment collection from borrower:** Each month, Rotessa will pull the interest payment from the borrower's account. When Rotessa confirms the payment (via webhook or API call), we know the money has been received by FairLend's bank. At this point, you create a transaction in the Formance Ledger to represent the incoming funds. For example, credit the mortgage's collection account with the payment amount. In double-entry terms, you might debit a generic "external/borrower" account or a temporary clearing account to balance the ledger entry (since the actual source is outside the platform). The ledger entry could be: *Source: external (or borrower), Destination:* `mortgage:M123:collect`*, Amount: $X*. This records that the mortgage's payment pot now has $X for distribution. (If you use multiple ledgers for segregation, this transaction goes into the ledger corresponding to financial transactions – likely a single main ledger since Formance can isolate by ledger name as well.)

2. **Determine distribution amounts:** Upon payment clearance, use the business logic discussed to compute how much each investor (including any who sold mid-period) should receive from this $X. Let's say Investor A and Investor B are the current owners with percentages, and Investor C was a previous owner during part of the month – you will compute amounts $A, $B, $C for each of them. The sum $A+$B+$C should equal the total $X (assuming no platform fees; if the platform takes a cut, that would be another component in the split).

3. **Ledger distribution transaction(s):** Use Formance Ledger to **split the funds to investors' accounts.** Thanks to Formance's atomic multi-destination support, this can be done in a single transaction script that specifies multiple destinations with percentages or amounts [5]. For example, a Numscript could look like:

```
send [CAD X] (
  source = @mortgage:M123:collect
  destination = {
    $A to @investor:InvestorA,
    $B to @investor:InvestorB,
    $C to @investor:InvestorC
  }
)
```

This would atomically move the respective amounts from the mortgage's collection account to each investor's account. The ledger now reflects that the mortgage's collected payment has been fully distributed out to the investors (the `collect` account balance would go down to zero). Each investor's ledger account is credited with the exact amount they are owed for that period. Formance's real-time balances allow us to know "who is owed what and where the money is" immediately after posting this [4]. Even if Investor C had sold their stake earlier, we still have their account on the ledger to credit (you wouldn't delete their account; it remains to receive any pending payouts).

Auditing: This transaction (or set of transactions) clearly documents the payout split. If you ever need to audit how a payment was divided, you can refer to this ledger entry which shows, for instance, that out of $X, Investor A got $A, B got $B, C got $C, on a given date.

1. **Pending vs. finalized status:** At this point in the flow, the ledger shows the investors have been credited. However, in reality the money is still in FairLend's bank account until it's actually sent out to each investor. We have two design choices here:
2. **Immediately reflect distribution in ledger (credit investors now):** This is the approach we just described. It treats the moment of allocation as the effective payout in the ledger. The advantage is the ledger now accurately shows a liability to each investor (their account balance increased) and no funds remaining in the mortgage pool. The disadvantage is that the actual bank transfers are still pending, so the ledger is slightly ahead of reality. However, since those funds are indeed owed to the investors at this point (the platform wouldn't use them for anything else), it's reasonable to consider them as essentially the investors' money already. The system can mark these ledger transactions with metadata like `status: "pending_payout"` to indicate that physical disbursement is in progress.
3. **Hold funds in a "pending" account until confirmation:** Alternatively, you could introduce a *Pending Payout* intermediary step. In that model, instead of crediting each investor's main account immediately, you transfer the funds to some holding accounts or mark them as pending. For example, you could have accounts like `investor:InvestorA:pending` to hold $A until it's confirmed paid, or simply keep it in the `mortgage:M123:collect` account and not split it until later. This adds complexity and delays reflecting the final allocation, so unless you expect frequent failures in the payout step, it's usually not worth it. It's often clearer to show the allocation when it's due, and handle exceptions if a payout fails.

Given the desire to put burden on Formance, **the first approach (immediate ledger distribution)** is preferred: record the distribution to investors as soon as the payment is available. This uses the ledger to track the obligation to each investor, leveraging its strength in real-time tracking. You can always explain that the ledger balance on an investor's account means "amount owed to them" if not yet withdrawn.

Off-Platform Disbursal and Confirmation

After the ledger shows amounts credited to investors, the actual disbursal happens off-platform (via manual wire transfers, e-transfers, or potentially a separate payment provider). Here's how to integrate that with the system:

- **Initiating payouts:** Once the distribution amounts are determined, your system can generate a payout instruction (this might be a CSV or a call to an external banking system) listing each investor and their amount. This is outside Formance's scope (since Formance isn't executing the wire), but you use the computed $A, $B, $C for that purpose. The business process might involve an operations team member taking those instructions and executing the transfers through a bank.

- **Confirmation webhook/API:** You will set up an API endpoint (or a small microservice) that can receive confirmation that the payouts have been completed. For instance, after the transfers are done, an operator could upload a CSV of completed payments or the system could receive a callback from a banking integration. The payload would include identifiers to match which mortgage payment or which distribution is being confirmed (e.g. a payment ID or mortgage ID and date, plus each investor's payout status).

- **Updating ledger or records on confirmation:** Upon confirmation, you want to mark those investor funds as disbursed. Since we credited the investor accounts already in the ledger, one way to indicate completion is to add metadata to the distribution transaction or to each investor's account balance. For example, you might call Formance's API to attach metadata `{"status": "paid", "paidDate": "<date>"}` to the transaction that credited Investor A $A. This doesn't change the balances (ledger entries are immutable aside from adding metadata), but it annotates that transaction as finalized. You could also record in your database that payment X was paid out on date Y for investor Z.

- **(Optional) Ledger transaction for cash outflow:** If you want the ledger to fully reflect money leaving the platform's books, you could also record a follow-up transaction that zeros out the investors' balances, representing the funds leaving the system to the investors' external bank accounts. For example, when Investor A's payout is confirmed, do a ledger transaction: *Source = @investor:InvestorA (debit $A), Destination = @payouts:external (credit $A)*. The `@payouts:external` account is a dummy account representing money that has been paid out of the platform (essentially the mirror of the external source we used when money came in). After this, Investor A's account balance in ledger goes back to 0 (since they no longer have money in the platform), and the external payouts account accumulates the total disbursed. This step ensures the ledger's internal accounts match reality (no lingering balances if you consider the platform doesn't actually hold those funds after payout). However, this approach doubles the number of transactions and might be overkill if all we need is to track that the payment occurred (since the primary concern is each investor got paid, which we know from confirmation).

In many cases, simply marking the original distribution as confirmed (via metadata or an external status flag) is sufficient. The ledger already did the heavy lifting of tracking who was owed how much. The confirmation step is mostly for the system's integrity and investor communication (e.g. to show in the UI that the payment status is now "Completed").

Handling delays or failures: If a payout is delayed a day or two (which you anticipated), the ledger and application can show the distribution as "pending". This could be done by a flag in your database or by the presence/absence of that confirmation metadata on the ledger transaction. For example, you might show in the investor's dashboard "Interest for November: $100 (Pending disbursal)" until the webhook comes in, then update it to "$100 (Paid on Dec 3)". Should a payout fail (say an e-transfer is rejected), you have a few options: - Reverse the ledger distribution (e.g. transfer $100 back from Investor's ledger account to the mortgage collect account or a "failed payouts" account) to reflect that it wasn't actually paid out, and then address the issue (perhaps retry or use a different method). Because the ledger is immutable, you wouldn't delete the original transaction, but you can post a new compensating transaction. This would be part of your exception handling flow. - Or simply leave the ledger as-is (investor still has $100 credited) and try the payout again later, since as far as the ledger is concerned they are still owed that money (the balance is sitting with them). This might actually be cleaner: the failure happened outside, so from the platform perspective the obligation is still there. You'd just ensure the status remains "pending" until successfully paid.

Using Formance Ledger for all these movements means **every financial event is recorded and auditable**: incoming payments, ownership changes, interest allocations, and even payout completions (if you choose to record them). The ledger's real-time and multi-ledger capabilities ensure you can query any account at any time to see balances. For example, you could query an investor's account balance to see if they have any unpaid distributions pending. Likewise, the mortgage's collection account balance being zero after distribution confirms that all collected funds have been allocated to owners.

Conclusion and Recommended Design

To summarize, the design is as follows: - **Fractional ownership represented via ledger assets:** Each mortgage's ownership is tracked by a custom asset (or ledger accounts with percentage balances), ensuring that transfers are atomic and the sum of ownership is always 100%. Formance Ledger supports custom assets and multi-party transactions to facilitate this [1] [5]. You will maintain a parallel record (in Convex DB's `mortgage_ownership` table) for convenient querying and enforcing invariants like 100%, but the ledger provides the authoritative history of changes. - **Interest collection and allocation:** When an interest-only payment is received, credit a mortgage-specific collection account in the ledger. Then use a **single multi-destination ledger transaction** (using Numscript or the API) to split that payment among all eligible recipients (current and former owners for that period). This leverages Formance's ability to do complex postings in one go, guaranteeing the totals are correct and the transaction is indivisible (all owners get paid in one atomic action). - **Proration for mid-period changes:** Initially, perform the proration calculation in your application logic. This means at distribution time, determine each investor's share of the interest based on how long they held their stake. This data can come from the ledger's recorded transfers or your `mortgage_ownership` history with timestamps. By doing this in code, you avoid having to manage partial accrual transactions inside the ledger. (In the future, if needed, you could introduce periodic accrual entries in the ledger for an even more automated approach.) - **Ledger as source of truth for balances owed:** By recording distributions in the ledger, you get real-time insight into obligations. At any point, **the ledger can tell you "what is owed to whom"** [4]. For example, an investor's account balance could represent the total funds that have been allocated to them but not yet withdrawn. This is incredibly useful for transparency and reconciliation. - **Payout confirmation workflow:** Since actual payouts happen off-platform, implement a callback or admin interface to confirm when payments are sent. Use this to update the status of those ledger transactions. If desired, also post a final ledger transaction to mark money leaving the platform (debit investor accounts, credit an external sink) to fully balance the books. This step ensures the ledger reflects that the platform no longer holds the money on behalf of the investor after payout. Even if you skip the explicit cash-out transaction, you will update your records so that the distribution is marked completed (and perhaps the investor's ledger balance being non-zero is understood as a transient state until payout, or cleared via a transaction upon confirmation).

By leaning on Formance Ledger for recording ownership and cash flows, you fulfill the goal of offloading as much as possible to the ledger. You gain the benefits of **atomic multi-party transfers, real-time balance tracking, and an immutable audit log** of all events [5] [3]. The application's responsibility is chiefly to orchestrate these ledger operations at the right times (e.g., on ownership transfer events, on payment received events, on payout confirmation events) and to handle the business-specific calculations (like proration and validation of 100% ownership).

This design provides a clear separation of concerns: - The **Formance Ledger** handles the correctness of financial transactions and state (no lost or double-counted funds, strict audit trail of who owns what and who got paid). - The **application logic (Convex backend)** handles conditional logic and integration (calculating how to split interest, interfacing with Rotessa and the manual payout process, and updating UI/ notifications).

By following this approach, you'll create a robust fractional ownership tracking system where every change and payout is transparently logged. Investors will receive the correct payouts even in complex transfer scenarios, and you (and auditors or regulators) can always reconcile the ledger to confirm that distributions were done correctly.

---

1 2 3 4 5 Formance - Ledger
https://www.formance.com/modules/ledger