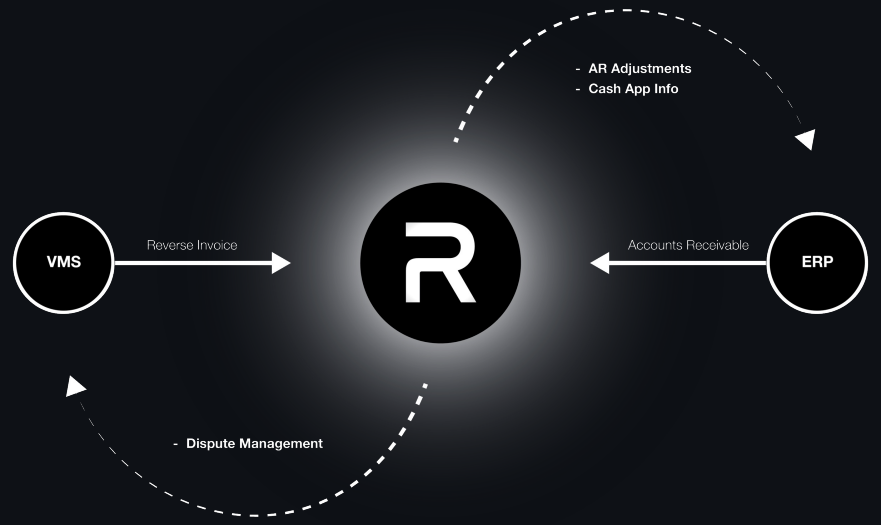
RECONCILED

Designing an Interoperability Platform for Accounts Receivable Reconciliation

How we built a shared model to let systems that were never designed to work together describe the same reality the same way.



Kyle Pontier | Co-Founder + CEO

Most people saw a workflow problem. We saw a data interoperability problem.

Everyone else threw spreadsheets and people at it. We asked a different question:

How do you create a shared understanding across systems that were never designed to work together?

Every platform used different schemas, naming, and business rules, and represented the same financial event differently. Before any reconciliation could happen, we needed a translation layer.

One agency, dozens of systems, no shared definition of the truth

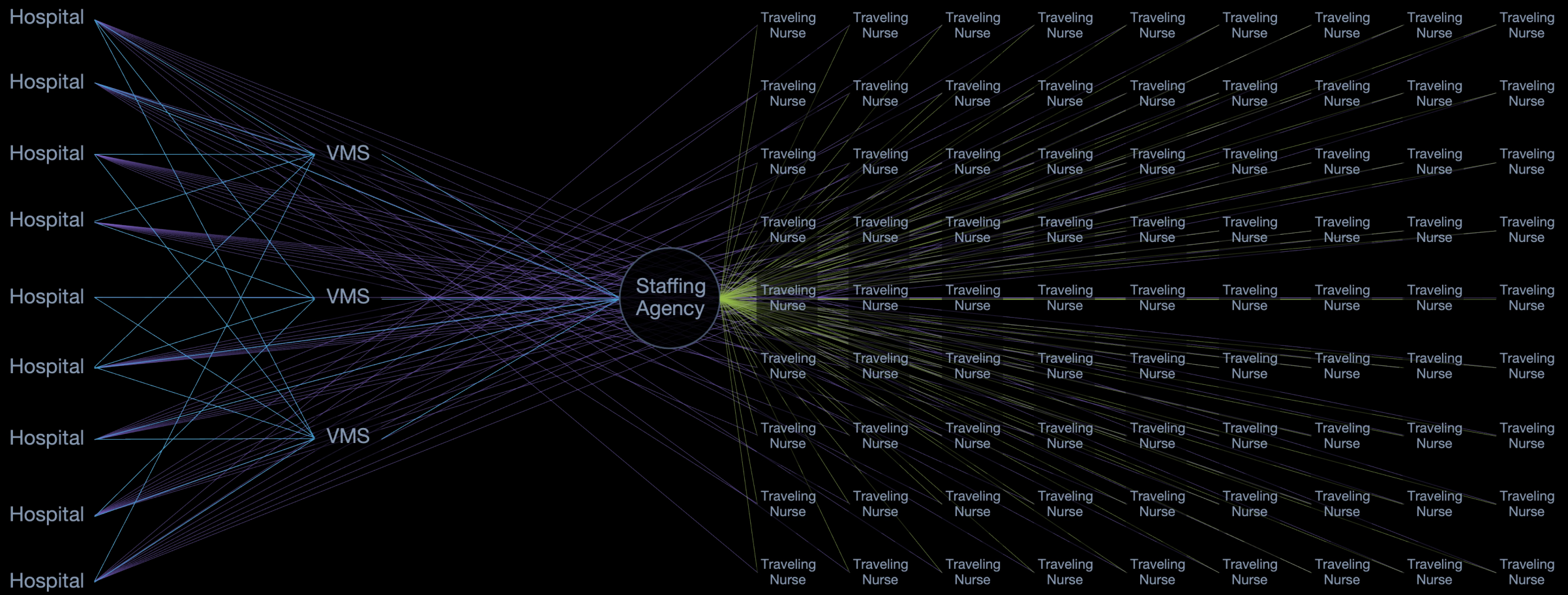
A single staffing agency works across dozens of employers, multiple Vendor Management Systems, separate internal timekeeping, and independent ERP and accounting platforms. We served agencies across healthcare, IT, and light industrial, the segments where this fragmentation runs deepest.

Each system describes the same underlying work differently.

THE ACTUAL CHALLENGE

The hard part wasn't collecting the data. It was determining whether multiple representations of reality referred to the same underlying event.

The Reconciliation Nightmare



One agency's web of hospitals, VMS platforms, and clinicians. Illustrated with healthcare staffing, the densest case.

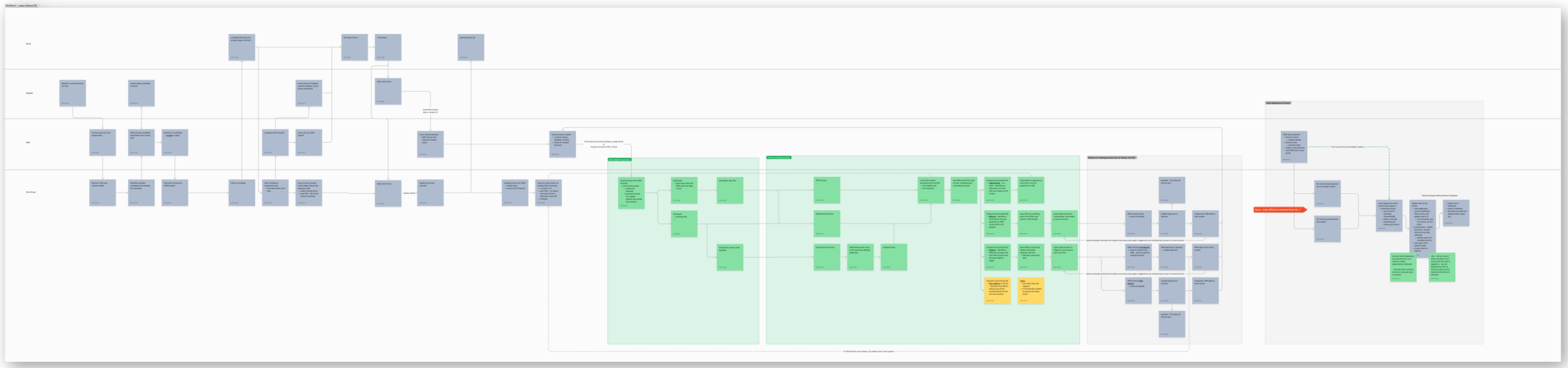
We mapped information flow, not screens

WHAT MOVED THROUGH THE SYSTEM

Employers · VMS platforms · Staffing agencies · Workers · Timecards · Invoices · Payments · Remittances

WHAT WE ACTUALLY STUDIED

Information flow · Ownership · Dependencies · Decision points · Failure modes



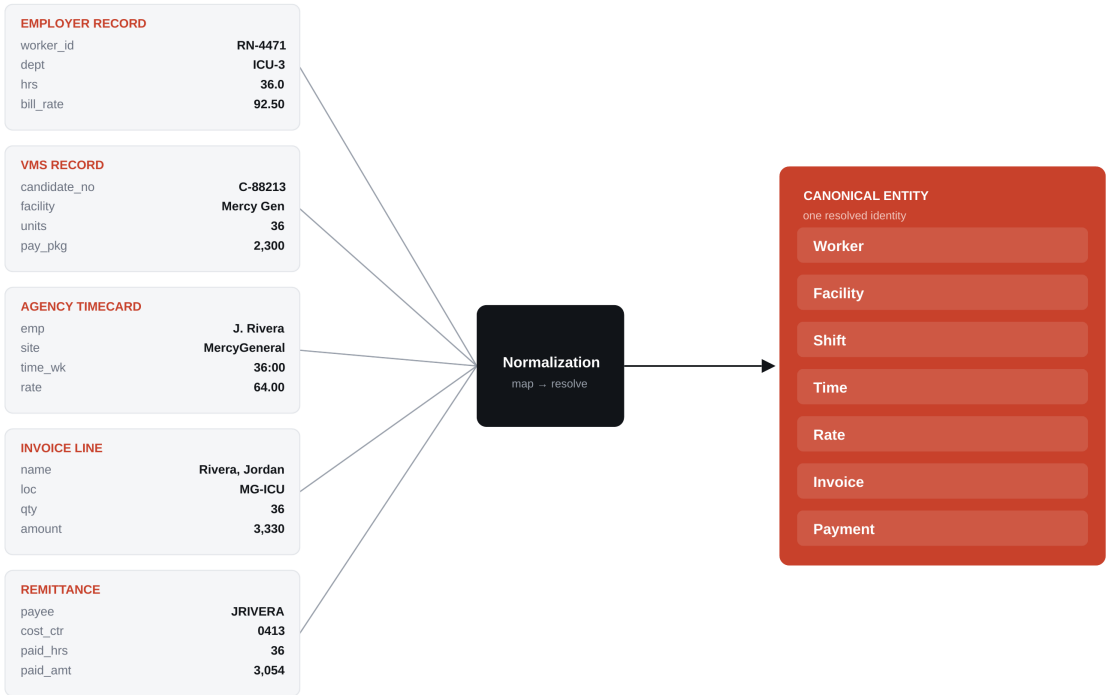
A fragment of the existing-state map. The finding: discrepancies were created upstream but discovered downstream, producing friction and delayed cash.

One identity, however many systems described it

The core design decision was not a screen. It was a normalized model every source could map into. Internally we called it the “Rosetta Stone” for reverse invoicing: a single canonical entity that resolved many representations to one truth.

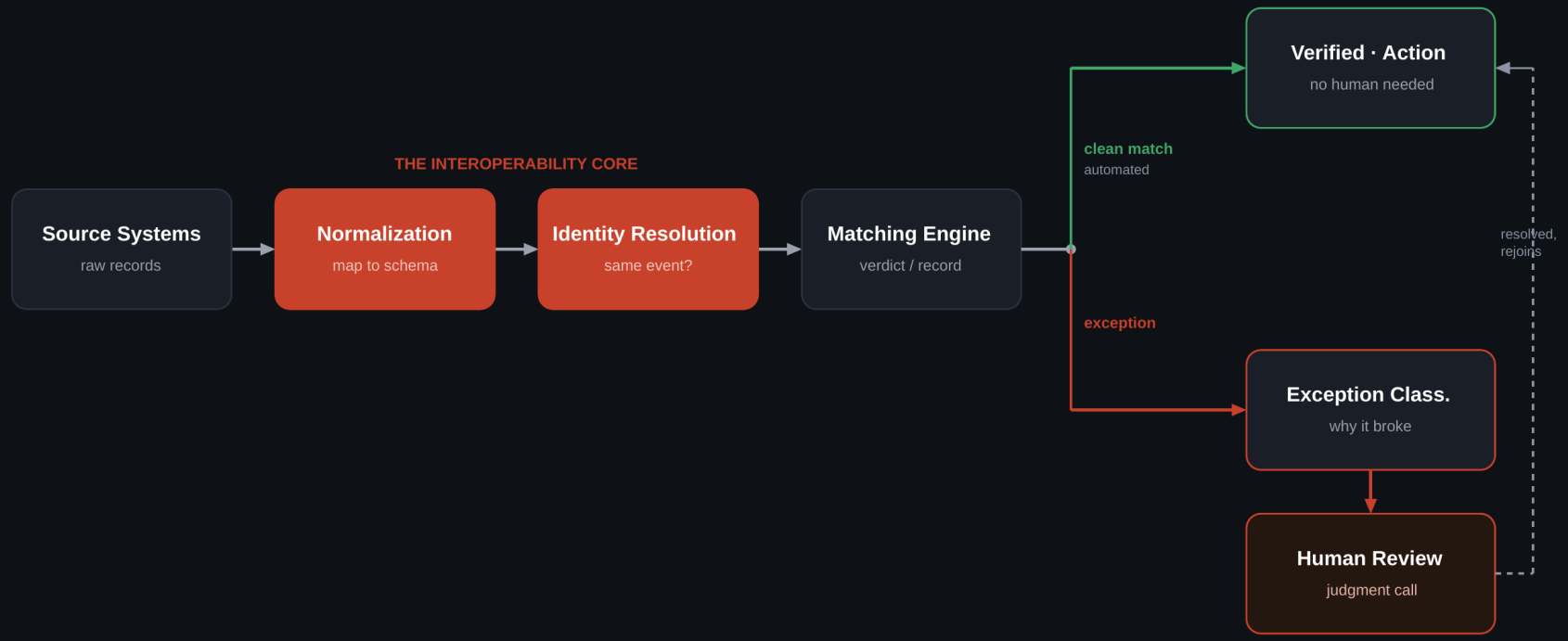
CANONICAL ENTITIES

Worker · Facility · Shift · Time · Rate · Invoice · Payment



FIVE SYSTEMS, FIVE VOCABULARIES, ONE REAL SHIFT

How a record becomes a verified, actionable truth



Clean matches resolve automatically. Only true exceptions reach a human, where the cost of a wrong call is highest.

Resolution was two decisions, not one

Once records entered the canonical model, the engine had to decide whether two records described the same event. Identifying a discrepancy was only the first decision. The harder one was what to do about it.

So we split resolution into two axes. First, classify the issue. Then determine the action. We encoded that logic into a decision matrix, so the right response to each kind of issue was explicit and repeatable, not re-litigated case by case.

ISSUE TYPE Exact Match · Rate · Hours · Information

ACTION Accept · Dispute · Unmatch · Missing

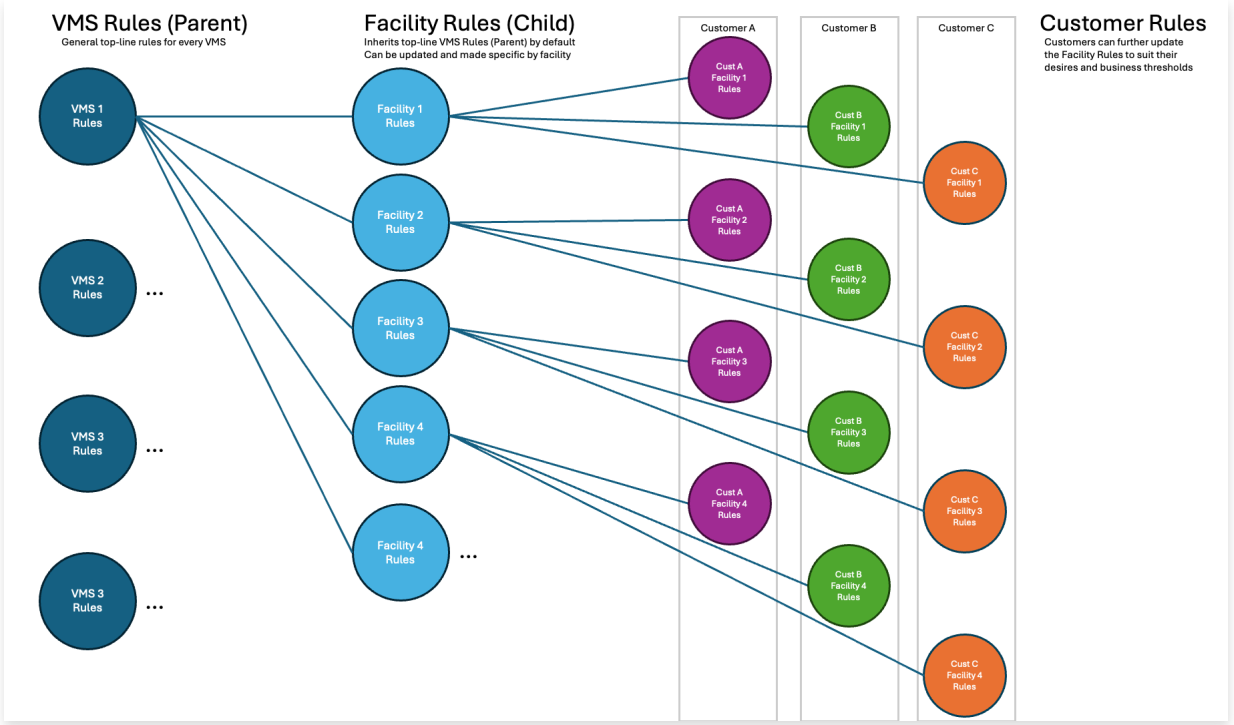
Our working decision matrix: issue type × action, operating logic in each cell.

The hard part was never displaying the discrepancy.
It was encoding the judgment for what to do about it.

Judgment that scaled without being rewritten

Encoding the judgment once was not enough. Every VMS, facility, and customer had its own thresholds for what counted as a discrepancy and what to do about it. Hard-coding each combination would not scale.

So we built the rules as an inheritance hierarchy. VMS-level rules set the defaults. Facility rules inherited them and could override. Customer rules refined them further. A change at the top propagated everywhere it should, while local exceptions stayed local.



VMS (parent) → Facility (child, inherits + overrides) → Customer (further refinement).

WHY IT MATTERED

Inheritance and override turned thousands of one-off rule decisions into one coherent, maintainable structure.

The system knew what was wrong. The user still had to trust it.

An automated verdict is worthless if the person accountable for every dollar won't act on it. Trust was the difference between using the system and overriding it.

So every flagged discrepancy had to answer four questions at a glance: why it was flagged, which rule fired, what it was worth, and what to do next.

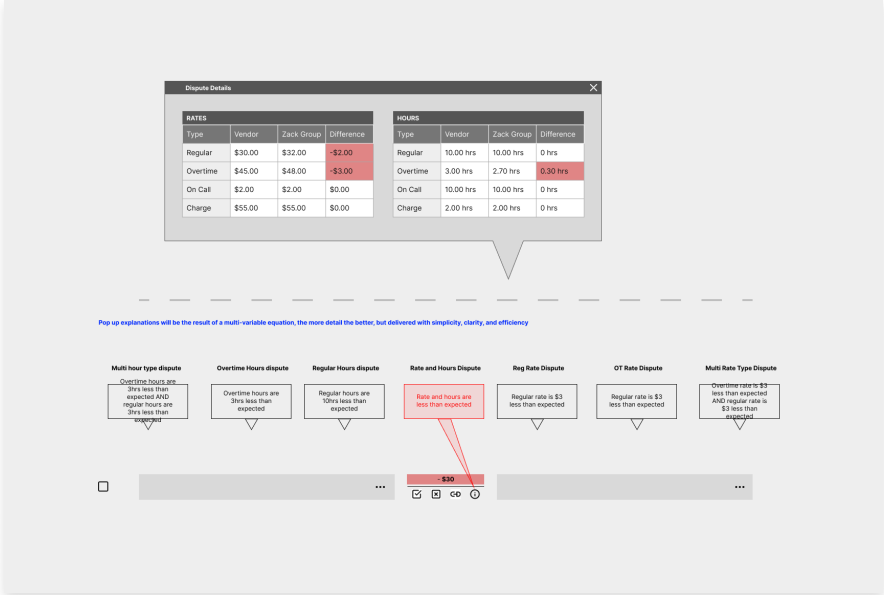
Judgment lived where it belonged: clean matches resolved themselves, and only true exceptions reached a person.

Transparent: why it happened is visible

Actionable: the next step is obvious

Defensible: it holds up to audit

Think of it like an inbox: the matches that were right filed themselves; what reached a person was only what genuinely needed one.



A flagged dispute, broken down: rates and hours, vendor vs. ours, with the difference and the action surfaced.

We earned the matching by hand before we automated it

1 · Deterministic first

Every new source of truth was hand-encoded into the Rosetta Stone matrix with explicit rules. No guessing. We built the ground truth ourselves.

2 · Patterns emerge

After dozens of encoded variations, structure appeared. New files were “like” ones we'd already solved, enough signal to train against.

3 · Then a classifier

Only then did we add an AI pipeline: new files run through the same funnel, get classified, and slot into the matrix, with humans owning judgment.

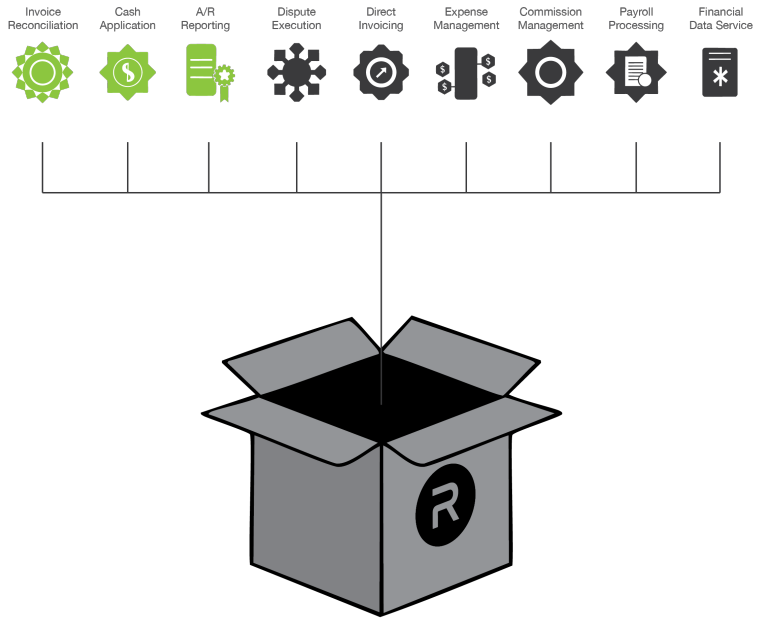
Why it mattered: finance demands accuracy, so we built for ~95% confidence and kept a human at the point of ambiguity. The same discipline scales to domains where the bar is five-nines and the cost of a wrong match is safety, not just dollars.

We weren't building a reconciliation tool. We were building a platform.

What began as reverse-invoice reconciliation became workflow automation, then a system of record, then a platform. Because every capability sat on the same shared model, each new module inherited the interoperability layer instead of rebuilding it.

THE BACK-OFFICE IN A BOX

One connected system, not a drawer of disconnected tools. The interoperability layer was the foundation every future module would stand on.



Green modules shipped; grey on the roadmap; all sharing one data foundation.

A shared model first; the metrics followed

SYSTEM OUTCOMES

- Unified multiple competing sources of truth
- Created a canonical, reusable information model
- Reduced ambiguity and increased trust in the data
- Established a scalable platform foundation

BUSINESS METRICS

90%+
less effort

~90%
write-off reduction

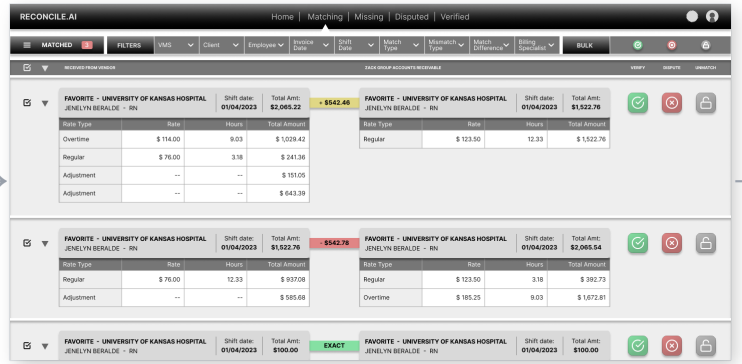
Faster
cash recovery

FROM PROOF TO PRODUCT

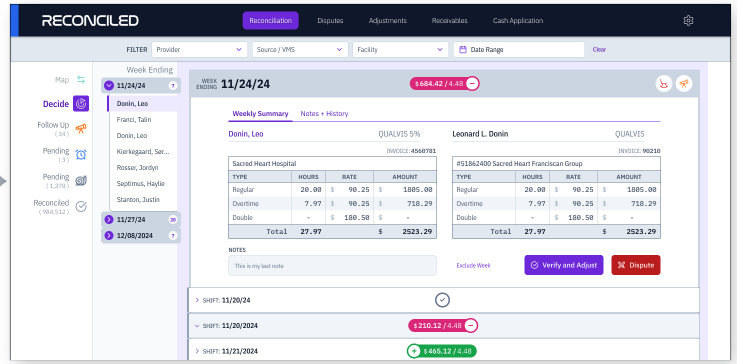
We proved the model in rough form, then refined it into production. The thinking came first; the polish followed.

1. Raw model validation

2. Working matching UI



3. Refined production UI



Complex systems rarely fail for lack of information.

They fail because information cannot move, cannot be trusted, or cannot be understood across the boundaries between teams, tools, and organizations.

The hardest problem at Reconciled was never collecting data or building screens. It was deciding when two systems were describing the same thing, and making that decision legible and trustworthy to the people who had to act on it.

That problem is not specific to staffing or finance. It is the shape of most complex, multi-system work.