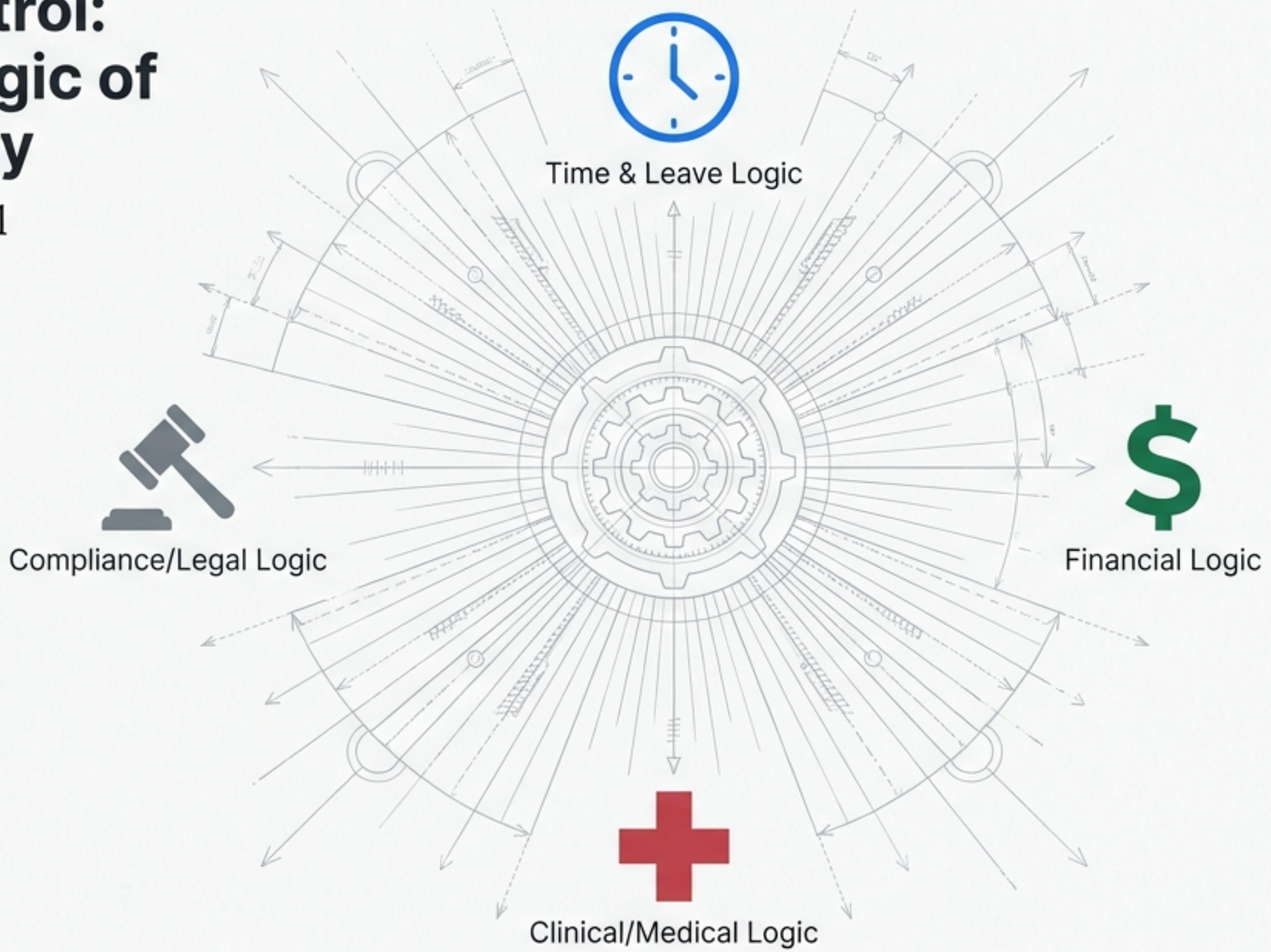
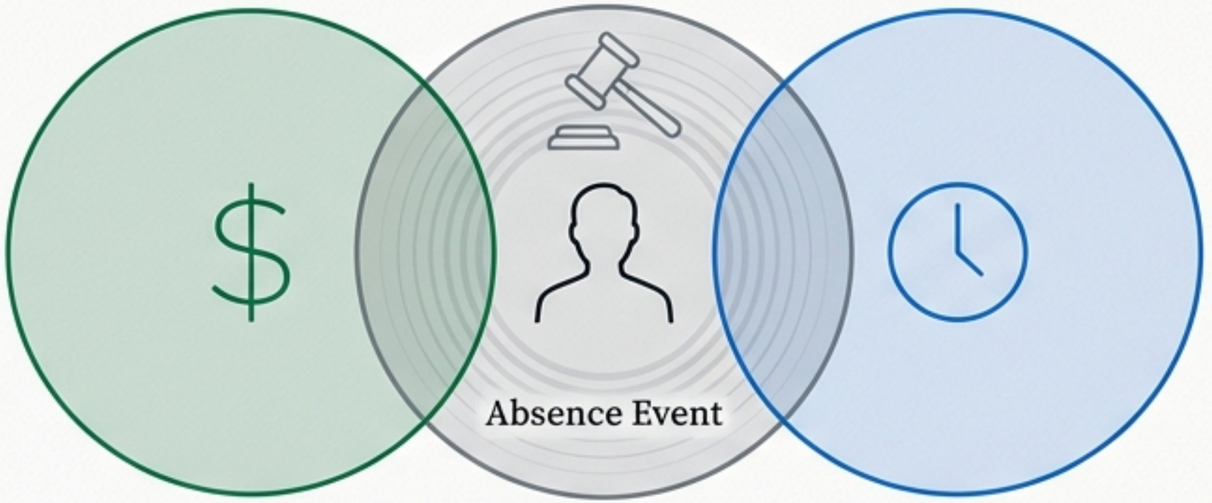
From Chaos to Control: Architecting the Logic of Absence & Disability

A System Blueprint for Technical Leaders and Architects



The Three Pillars of Coverage: A Balancing Act of Money, Time, and Law

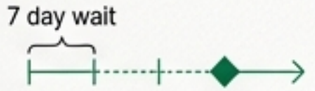


Pillar 1: Short-Term Disability (STD)

Purpose: Income Replacement (Temporary)

Duration: 3-6 Months

Key Logic:
The "Elimination Period" (e.g., 7-day wait). Benefit Start Date = Date of Disability + 7 days.



Typical Claims:
Surgery, Pregnancy, Acute Illness.

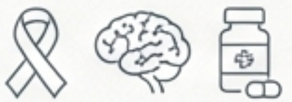
Pillar 2: Long-Term Disability (LTD)

Purpose: Income Replacement (Catastrophic)

Duration: Years, up to retirement age.

Key Logic:
The critical "Definition of Disability" change from "Own Occupation" to "Any Occupation" at the 24-month mark.

Typical Claims:
Cancer, MS, Chronic Conditions.

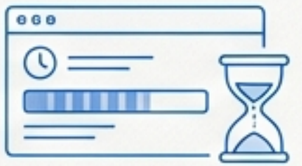


Pillar 3: Absence & Leave Management

Purpose: Job Protection

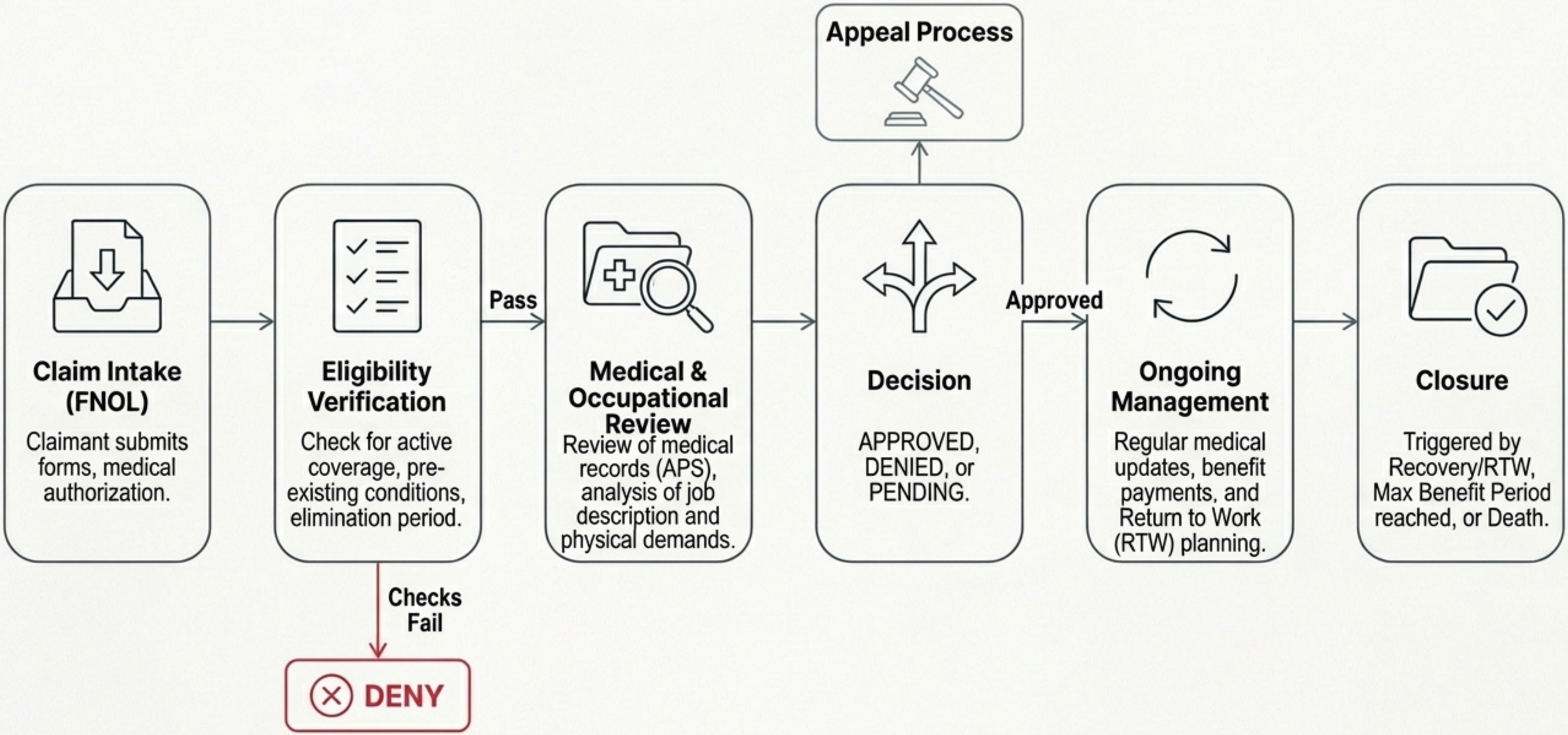
Duration: Varies (e.g., FMLA is 12 weeks).

Key Logic:
Tracking usage against a "bank" of time, often down to the hour.



Typical Claims:
FMLA, State PFML, Intermittent Leave.

The Claim Lifecycle: A State Machine from Notice to Resolution



Architect's Challenge #1: The Financial Waterfall and Overpayment Clawbacks

The Scenario

- A common and costly problem: retroactive Social Security (SSDI) awards.
- **Months 1-18:** Carrier pays claimant $3,000/mo. Total paid: $54,000.
- **Month 19:** SSDI retroactively approves claimant for $1,000/mo back to Month 1.
- **The Result:** The carrier *should* have only paid $2,000/mo.
- **The Overpayment:** $1,000 x 18 months = $18,000. The claimant is legally obligated to repay this from their SSDI lump sum.

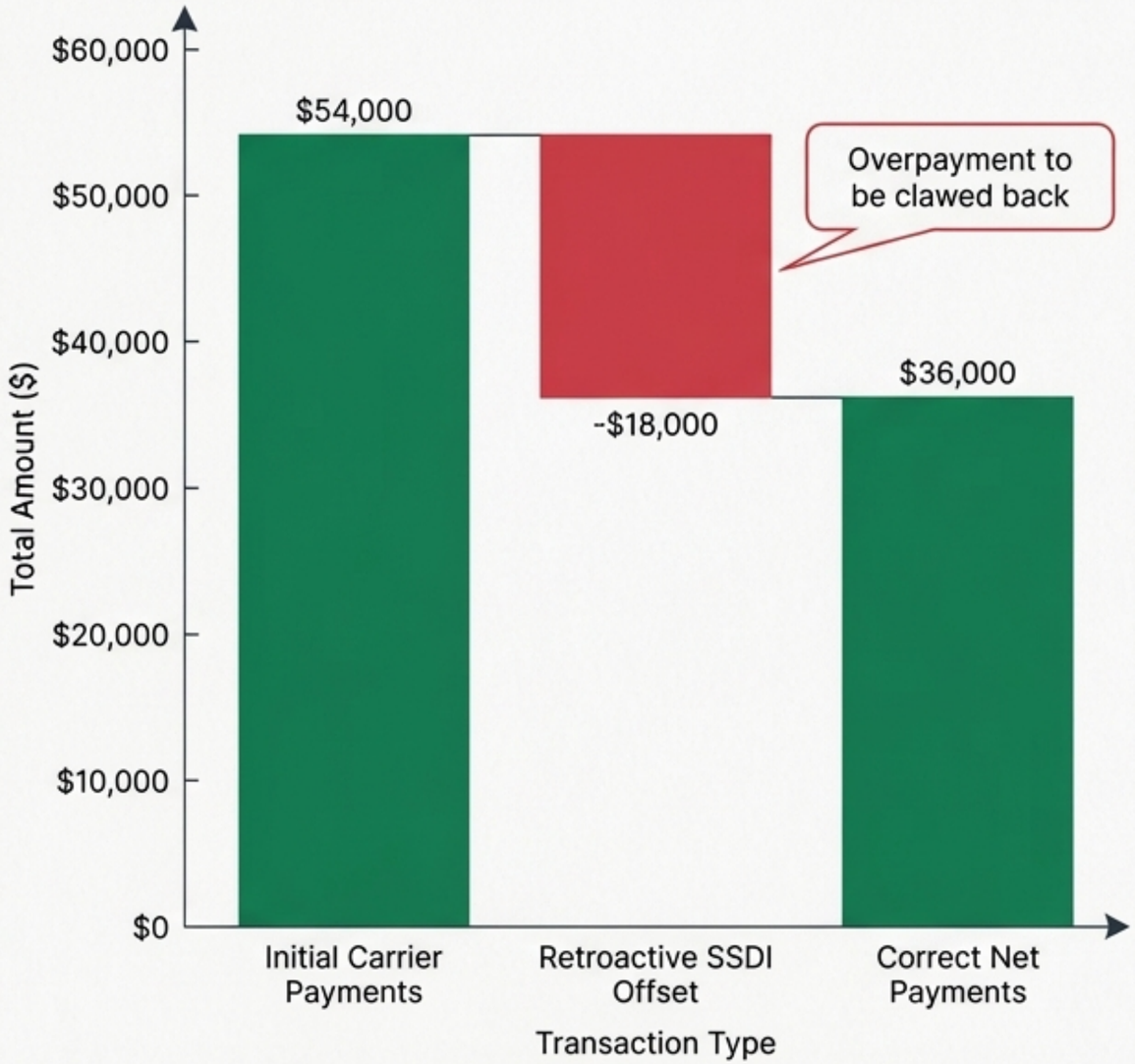
The Architectural Requirement

- The system needs a Ledger entity that supports negative balances.
- It must manage Recovery Plans (e.g., reduce future checks by 10%).

The Algorithm Corner

- Even with offsets, a minimum benefit is often guaranteed.

```
Payment = MAX( (Gross_Benefit - Offsets), MIN_BENEFIT_VALUE )
```



Architect’s Challenge #2: The Time-Tracking Paradox of “Rolling Backward” Leave

The Problem

The law allows employers to choose one of four methods for counting a “12-week” leave bank. The software must support all four, but one is notoriously difficult.

1. **Calendar Year**: Resets Jan 1. (Simple)
2. **Fixed Year**: Resets on a specific date (e.g., July 1). (Simple)
3. **Rolling Forward**: Bank resets 12 months from the first day of leave. (Moderate)
4. **Rolling Backward (The Gold Standard)**: “Look back 12 months from *today*.” (Complex)

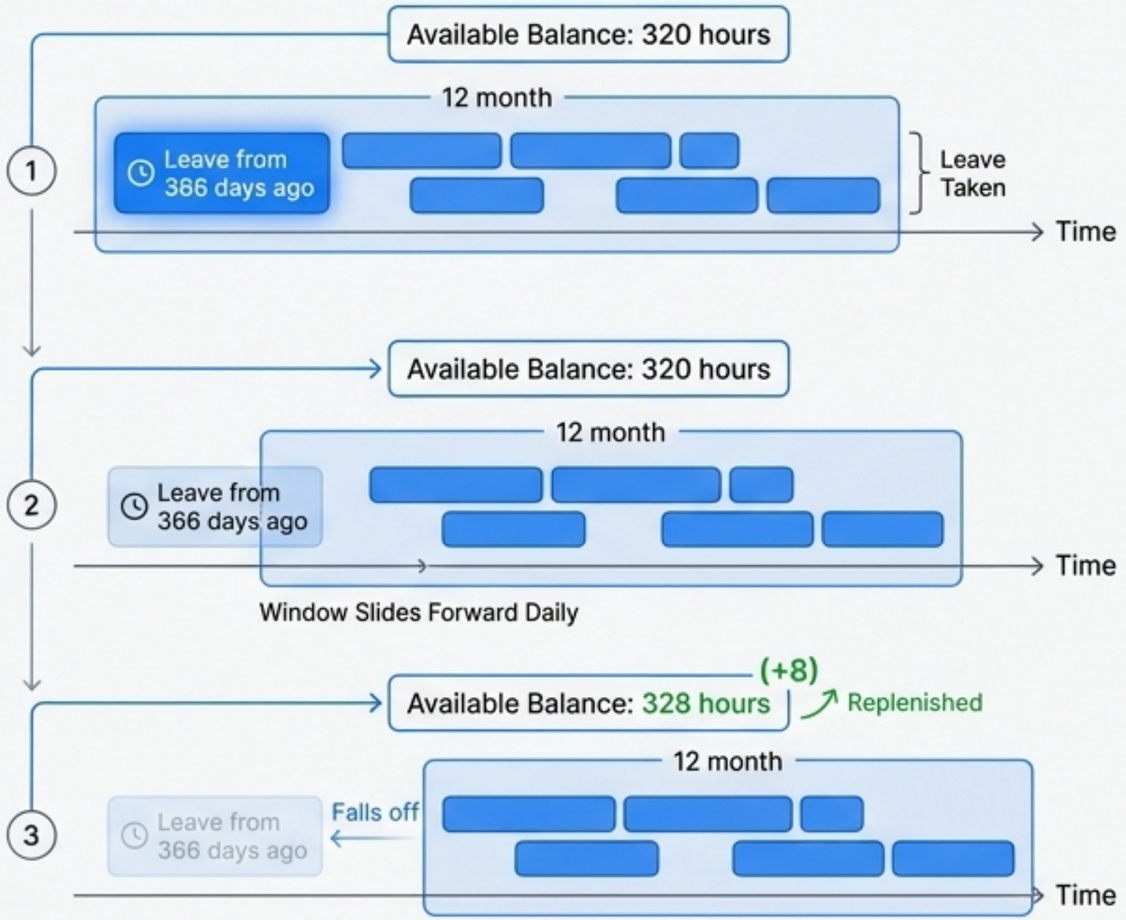
The “Rolling Backward” Math

Every single day, the system must perform a calculation for every employee: `Today - 365 days`. Any leave taken naturally “falls off” the back end as time moves forward, replenishing the available balance.

The Architectural Requirement

This demands a daily, scheduled recalculation of the available leave balance for every active employee, rather than a simple one-time decrement.

Visualizing the Rolling Window



Architect’s Challenge #3: The UI/UX Nightmare of "Stacked" Concurrent Leaves

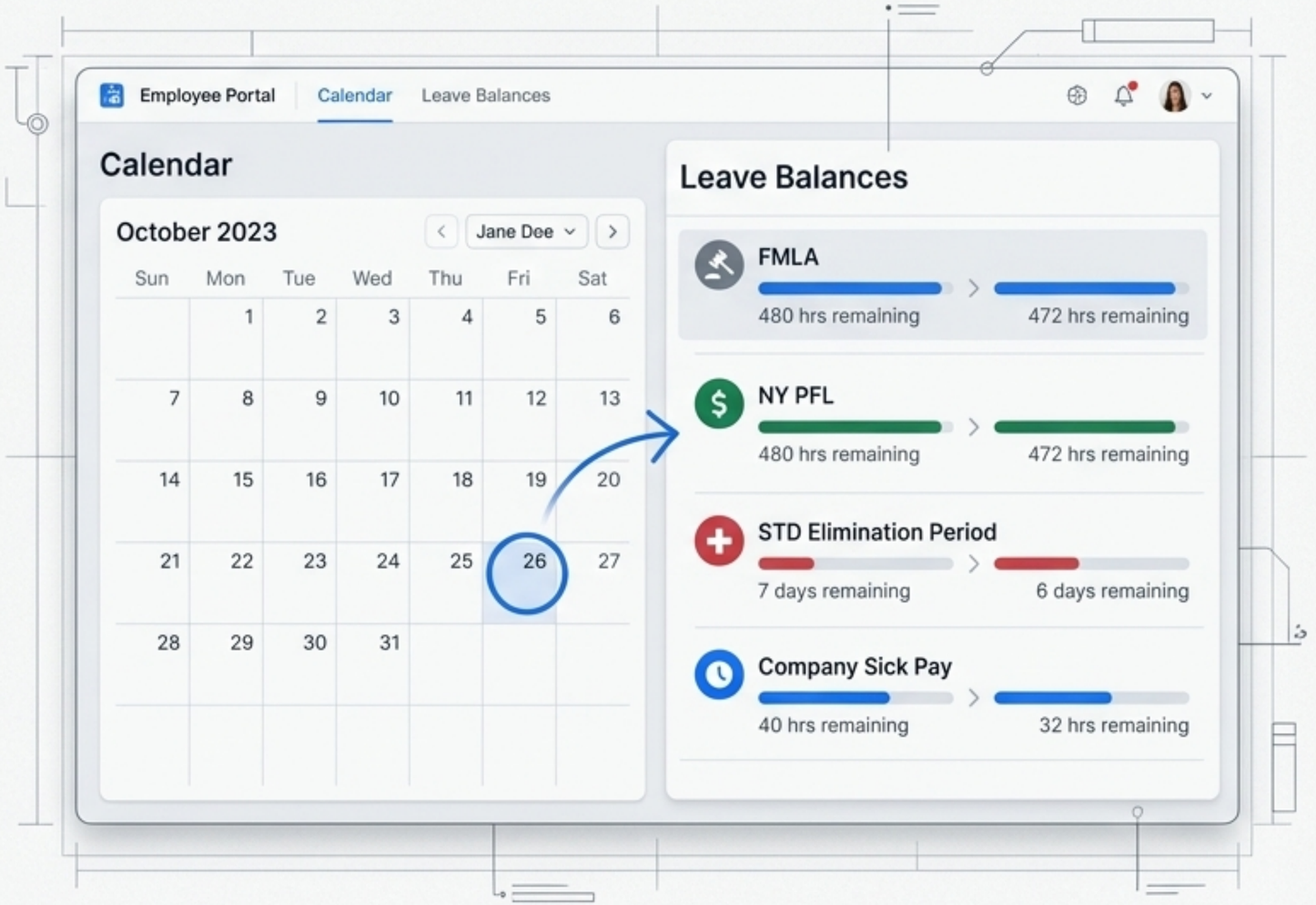
The Scenario

A single day of absence can draw from multiple policies and laws simultaneously. Consider a pregnant employee in New York taking a day off for prenatal care.

The System’s Action

One input (“I am out today”) must fork and update four separate counters/ledgers.

- **FMLA (Federal)**
  Decrements the 12-week job protection bank.
- **NY Paid Family Leave**
  Decrements the state-mandated paid leave bank.
- **STD Policy**
  Decrements the 7-day “Elimination Period” before payments start.
- **Company Sick Pay**
  Decrements the accrued sick hours and triggers payroll.



Architect's Challenge #4: The Shifting Goalposts of 'Disability'

The definition of disability in Long-Term Disability (LTD) policies changes over time, making it progressively harder to remain qualified. The system must flag this critical transition.

Phase 1: "Own Occupation" (First 24 months)

"Cannot perform the material duties of **YOUR OWN** occupation."

Cannot perform construction work.

Can perform office work.

Result: DISABLED

Phase 2: "Any Occupation" (After 24 months)

"Cannot perform **ANY** occupation for which you are reasonably qualified by education, training, or experience."

Still cannot perform construction work.

Considered capable of office work.

Result: NO LONGER DISABLED

First 24 Months

24 Months

After 24 Months

The Architectural Requirement

The system needs a scheduled trigger to flag claims for review at the 24-month mark, alerting the case manager to apply the stricter 'Any Occupation' standard.

Deep Logic: Three Hidden Traps That Break Systems

Trap #1: The Pre-Existing Condition Ambush

The Rule:
The standard '3/12 Rule." A claim filed in the first 12 months of coverage is checked against the 3 months *prior* to coverage.

The Logic: If treatment for the same condition occurred in the look-back period, the claim is denied.

The Trap: A simple prescription refill counts as treatment. If a pharmacy script refill date falls in the 3-month look-back window, it triggers the exclusion.

Trap #2: The Mental Health Lifetime Cap

The Rule:
Most LTD policies cap benefits for Mental/Nervous and 'Self-Reported' conditions at 24 months *for the claimant's lifetime.*

The Architectural Need:
A 'limitation bucket' counter that persists across claims. If a claimant uses 12 months for depression in 2020, they only have 12 months left for an anxiety claim in 2024.

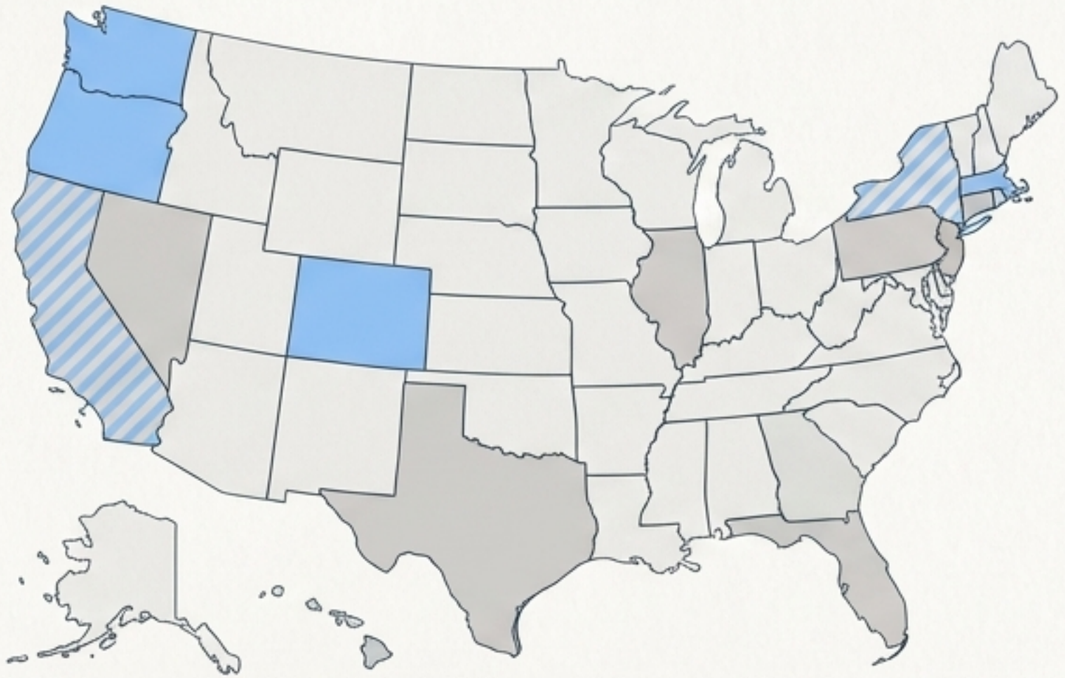
Trap #3: The Partial Disability Inflation Problem

The Rule:
Partial disability uses a 'Proportionate Loss' formula:
(% Loss) = (Pre_Disability_Earnings - Current_Earnings) / Pre_Disability_Earnings.

The Trap:
For a 10-year claim, 'Pre-Disability Earnings' from 2014 are meaningless today. The logic must apply a CPI (Inflation) increase to the earnings basis *specifically for this calculation*.

The Compliance Matrix: Architecting for a Patchwork of 50 Legislatures

The US has no single set of rules. Federal, State, and even Municipal laws stack on top of each other. Hardcoding is not an option.



Type A: Statutory Disability
Type B: Paid Family & Medical Leave (PFML)
Type C: Paid Sick Leave (Accrual)

Classification of State Leaves

Your data model must handle three distinct types.

Type A: Statutory Disability (The "Old Guard")

- **States:** CA, NY, NJ, HI, RI.
- **Function:** Mandates coverage for an employee's own medical condition.
- **Architectural Note:** These are often ***offsets*** to private STD.
  `STD Benefit = (Salary * 60%) - (NY DBL State Benefit)`.

Type B: Paid Family & Medical Leave (PFML - The "New Wave")

- **States:** WA, MA, CT, OR, CO, etc.
- **Function:** Covers Medical, Family Care, and Newborn Bonding.
- **Architectural Note:** Employers can "opt-out" of state funds with a private plan. Your software must act as a State Proxy.

Type C: Paid Sick Leave (Accrual-Based)

- **Jurisdiction:** Often City-level (NYC, SF) plus State-level.
- **Function:** "Earn 1 hour for every 30 hours worked."
- **Architectural Note:** Requires live integration with Payroll/Time & Attendance systems to read the real-time accrued balance.

State-Specific ‘Gotchas’: Why Hardcoding Logic Is Doomed to Fail

California: The “Pregnancy Stack”

The Laws: Pregnancy Disability Leave (PDL) + California Family Rights Act (CFRA).

The Logic: Most states run these concurrently. California runs them *sequentially*. The system must not decrement the 12-week CFRA (bonding) bank while the user is on PDL (disability), allowing for a much longer total leave.

New York: The “Two-Bucket Problem”

The Laws: Disability Benefits Law (DBL) + Paid Family Leave (PFL).

The Logic: You cannot take DBL and PFL on the same day. More importantly, there is a combined cap. Your system must enforce: `IF (DBL_Weeks_Used + PFL_Weeks_Used > 26) THEN DENY`.

Oregon: The Expansive Definition of “Family”

The Laws: Oregon Paid Family Leave.

The Logic: Most states define family narrowly (Spouse, Child, Parent). Oregon includes anyone with a ‘close association equivalent to a family relationship' (Affinity). Your 'Relationship' dropdown menu cannot be hardcoded; it must be dynamic.

The Core Concept: Job Protection vs. Paid Benefit

The Logic: FMLA (job protection) and PFML (paid benefit) are separate. An employee at a 15-person startup in MA might get paid (MA PFML) but have no job protection (FMLA requires 50+ employees). The system must generate letters with appropriate disclaimers.

The Architectural Mandate: A Rules Engine, Not Brittle Code

This approach is unmaintainable and guarantees failure.

The Anti-Pattern

```
if (state == 'CA') {
  ...
}
```

This approach is unmaintainable and guarantees failure.

The Solution: A Configuration-Driven Rules Engine

Create a "Plan Design" object for every state law and policy, stored in a database or database or JSON.

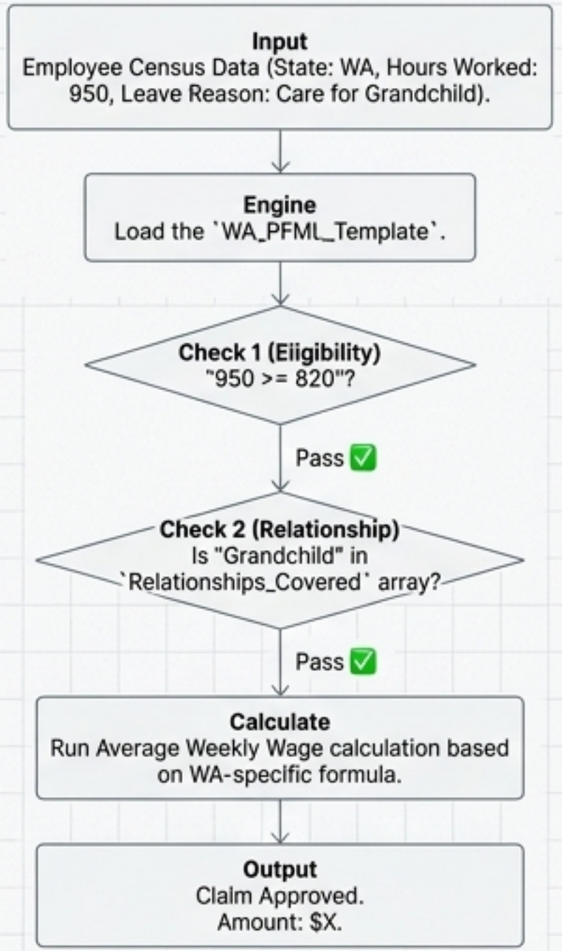
The Solution: A Configuration-Driven Rules Engine

Create a 'Plan Design' object for every state law and policy, stored in a database or JSON.

WA_PFML_Template.json

```
{
  "Eligibility_Hours": 820,
  "Eligibility_Tenure": "12 months employment",
  "Benefit_Cap": "1456 per week",
  "Bank_Duration": "12 weeks",
  "Bank_Method": "Rolling Forward",
  "Waiting_Period": "7 days unpaid",
  "Relationships_Covered": ["Parent", "Child",
   "Spouse", "Grandchild", "Affinity"]
}
```

The Calculation Pipeline



The Blueprint: Foundational Data Models

A robust and flexible schema is essential to support the complex logic. Here are three critical table structures.

Policy_Config Table (Configuration)

Needs to be JSON-heavy or NoSQL to handle variation.

Policy_ID
Effective_Date
Definition_Of_Disability: “24m_OwnOcc”
Elimination_Period_Days: 7
Benefit_Percentage: 0.60
Max_Monthly_Benefit: 10000
Pre_Ex_Rule: "3/12"
Lookback_Method: “Rolling_Backward”

Claim_Ledger Table (Financials)

Tracks every financial transaction for a claim.

Transaction_ID
Claim_ID
Date_Start
Date_End
Benefit_Type: (STD/LTD)
Gross_Amount
Offset_SSDI
Offset_WorkComp
Taxable_Amount
Net_Check_Amount

Absence_Bank Table (Time Tracking)

Tracks available time for leave entitlements.

Bank_ID
Employee_ID
Entitlement_Type: “FMLA”
Total_Hours_Entitlement: 480
Hours_Used: 45.5
Balance_As_Of_Today: 434.5

The Unifying Principle: Build an Event Sourcing System

Do not just store the 'current status' of a claim. The state of a claim is a function of a sequence of historical events. You must store this history.

Why is this essential?

Retroactive Recalculations

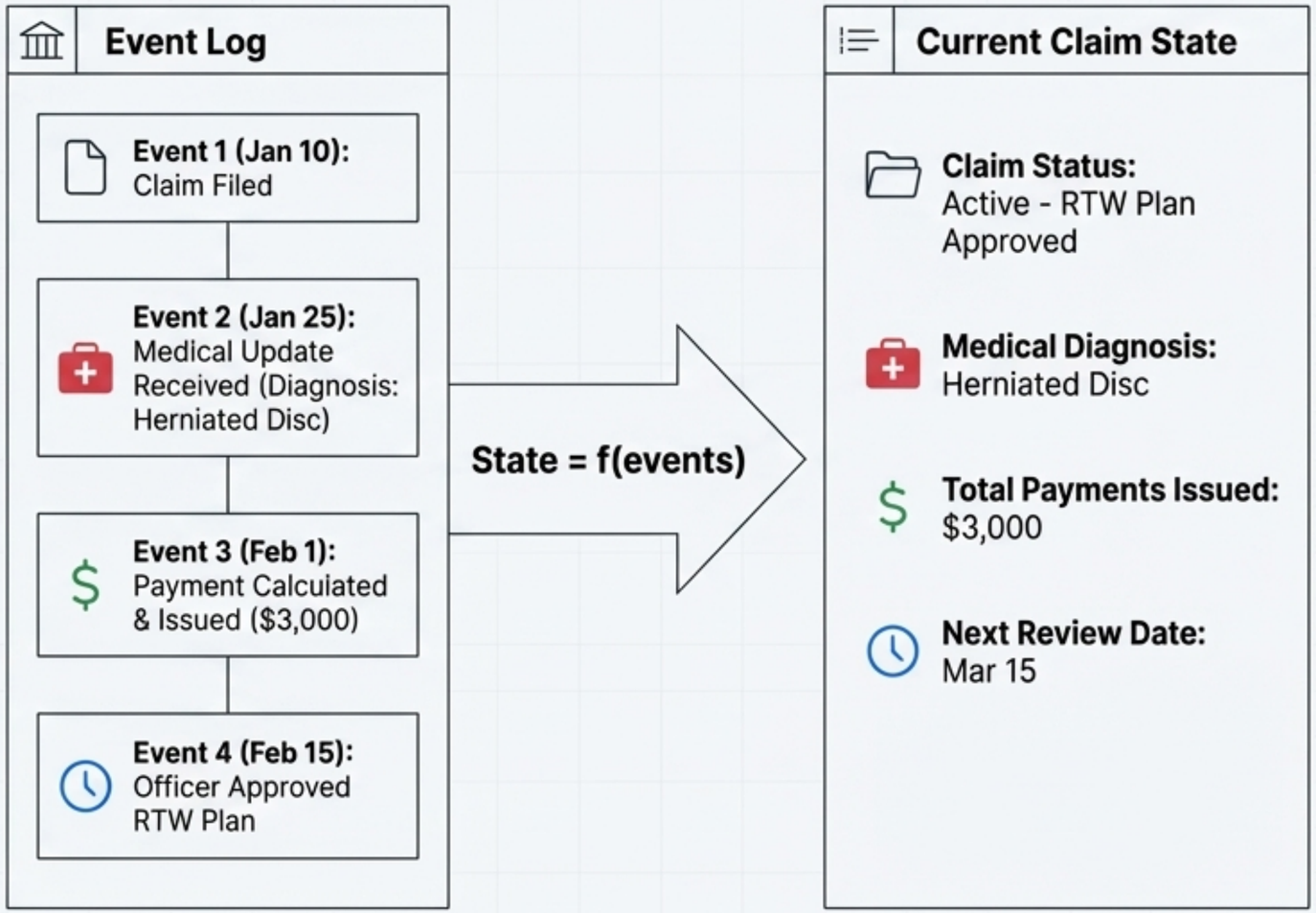
When an SSDI award is approved retroactively, you must rewind history and replay events with the new information to correctly calculate overpayments.

Audits & Compliance

Auditors need to see *why* a decision was made at a specific point in time, based on the information available *then*. An event log provides an immutable audit trail.

Debugging & Analytics

Replaying events makes it possible to debug complex state interactions and analyze the evolution of claims over time.



The Architect's Checklist for Success

1. 

Isolate Logic from Code

Use a Rules Engine for state/policy variations. Avoid hardcoded `if/else` logic.

2. 

Model Time and Money with Precision

Your Ledger must handle retroactive changes and negative balances. Your time tracking must support support daily 'rolling backward' calculations.

2. 

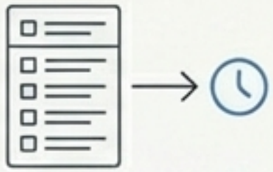
Embrace Concurrency

Design for a world where one absence event triggers multiple, independent processes (STD, FMLA, PFL, Payroll) 'rolling backward' calculations.

3. 

Embrace Concurrency

Design for a world where one absence event triggers multiple, independent processes (STD, FMLA, PFL, Payroll) simultaneously.

4. 

Persist Every Event, Not Just the Current State

An event-sourced architecture is not optional; it's required to handle retroactive adjustments and audits correctly.

5. 

Separate Job Protection from Paid Benefits

Understand that FMLA and STD/PFML are distinct concepts. The system's UI and communications must reflect this separation.

The Goal: A System as the Single Source of Truth

You are not just building a claims processing tool. You are architecting the “brain” that navigates a high-stakes world of finance, regulation, and human circumstances. A successful system brings clarity to chaos, ensures compliance, and ultimately provides a stable, predictable foundation for managing some of the most challenging moments in an employee’s life.