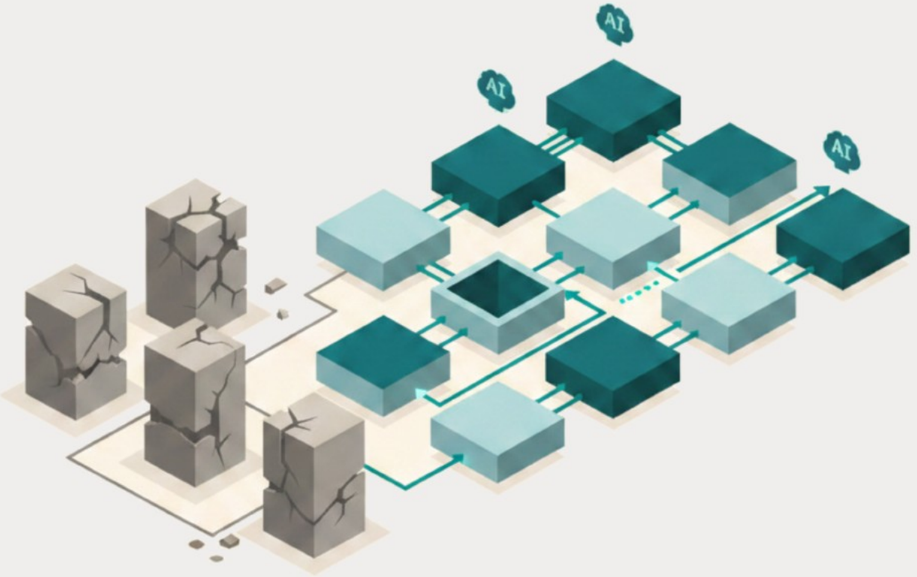
Accelerating legacy modernization with AI at PayFit

FlowCon meetup · Paris · April 2026

AI-Powered | Legacy Systems | Modernization



Context: We have a legacy system slowing us down and causing data sync / UX issues

Question: Can AI help us to modernize this faster so we can focus on improving the product and innovating faster?

Insights I'll share today:

- where AI helped us go 2 - 3x with instant ROI
- where it didn't help / where it caused friction
- what's possible with more investment in our AI modernization harness

PayFit staging

Rechercher ...

Les essentiels

- Tableau de bord
- Mes collaborateurs
- Demandes à valider 6
- Vérifier et clôturer la paie

Au quotidien

- Absences et temps de travail
- Éléments de rémunération
- Gestion de l'équipe
- Entretiens et objectifs
- Documents

Ma structure

- Données et exports
- Modes de paiement

← Retour à mes collaborateurs

Rechercher un autre collaborateur...

test test

test

Aperçu du bulletin

Contrat (1)

Salarié - CDI
Depuis le 01/09/2022
Action nécessaire

Organisation

Équipe
Ajouter une équipe

Manager
Ajouter un manager

Aperçu | Informations personnelles | Synthèse du mois | Congés & Absences | Éléments de rémunération | Contrat | Fin de contrat

Action nécessaire
La date de début de contrat de test test a été modifiée. Il est nécessaire de mettre à jour la reprise d'historique. Traiter

Informations

01/01/2001

test@test.com

test
93110, ROSNY SOUS BOIS
FR

Salaire contractuel

Voir les détails

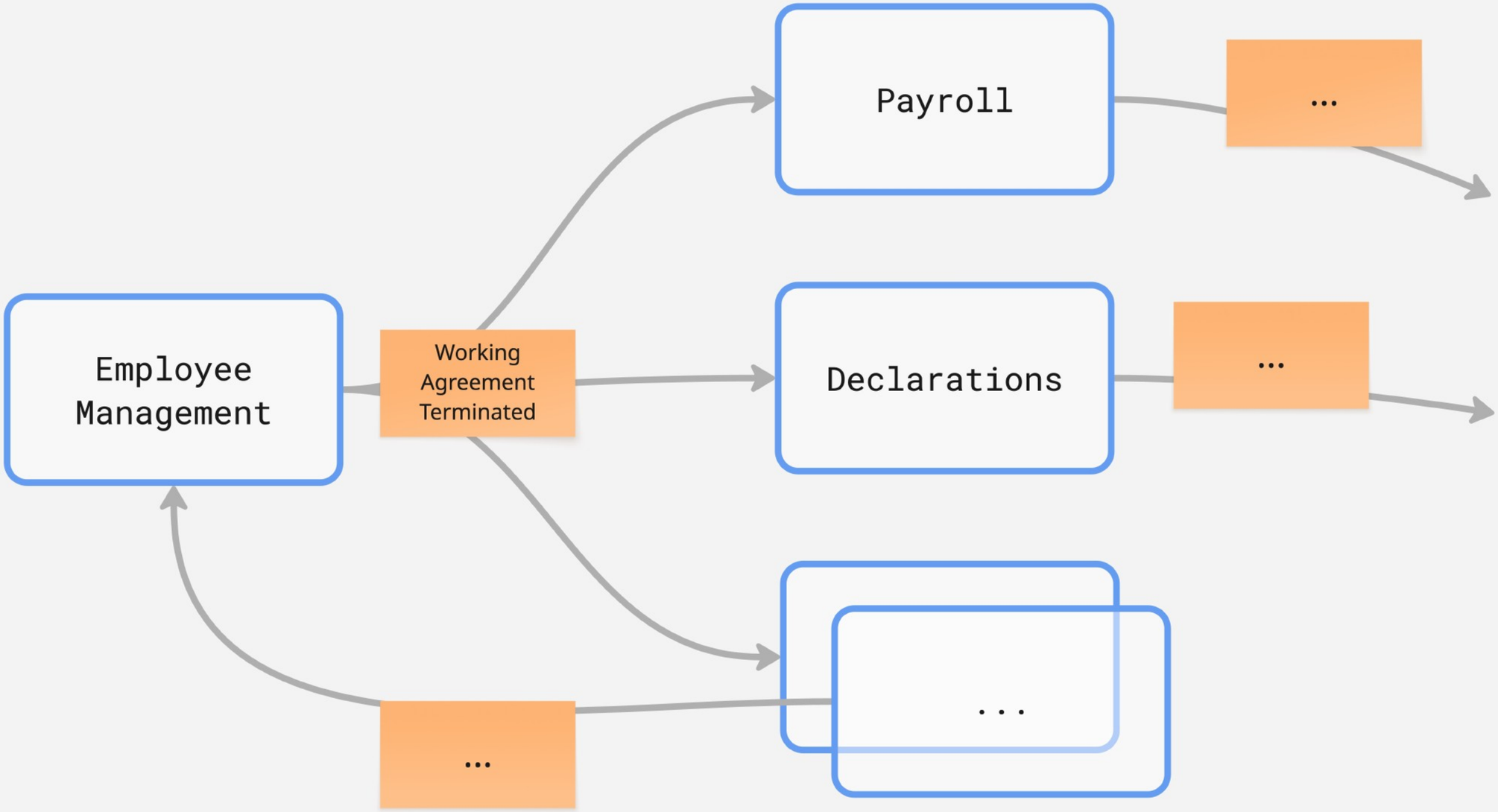
24 396 €
Annuel

2 033 €
Mensuel

Evénements

Anniversaire 01/01/20...

PayFit Architecture Vision: DDD & EDA



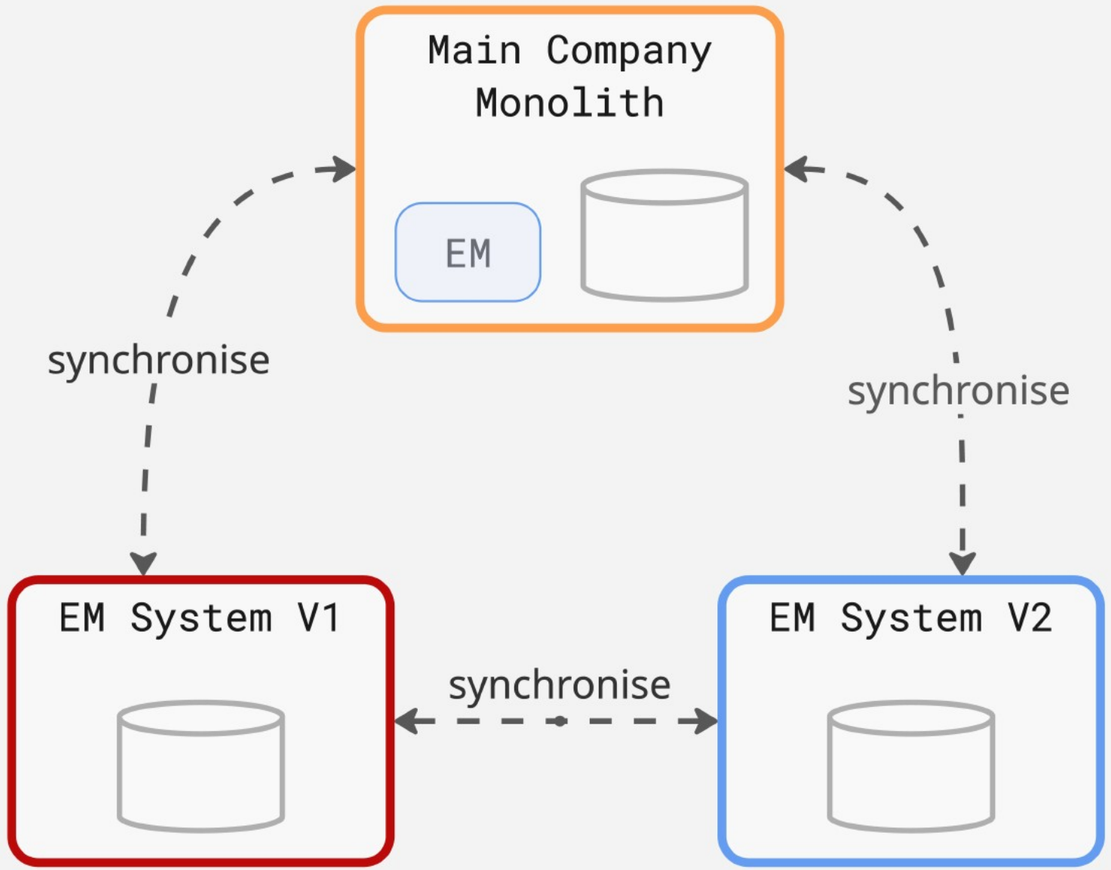
careers.payfit.com/

01

PARTIE 1

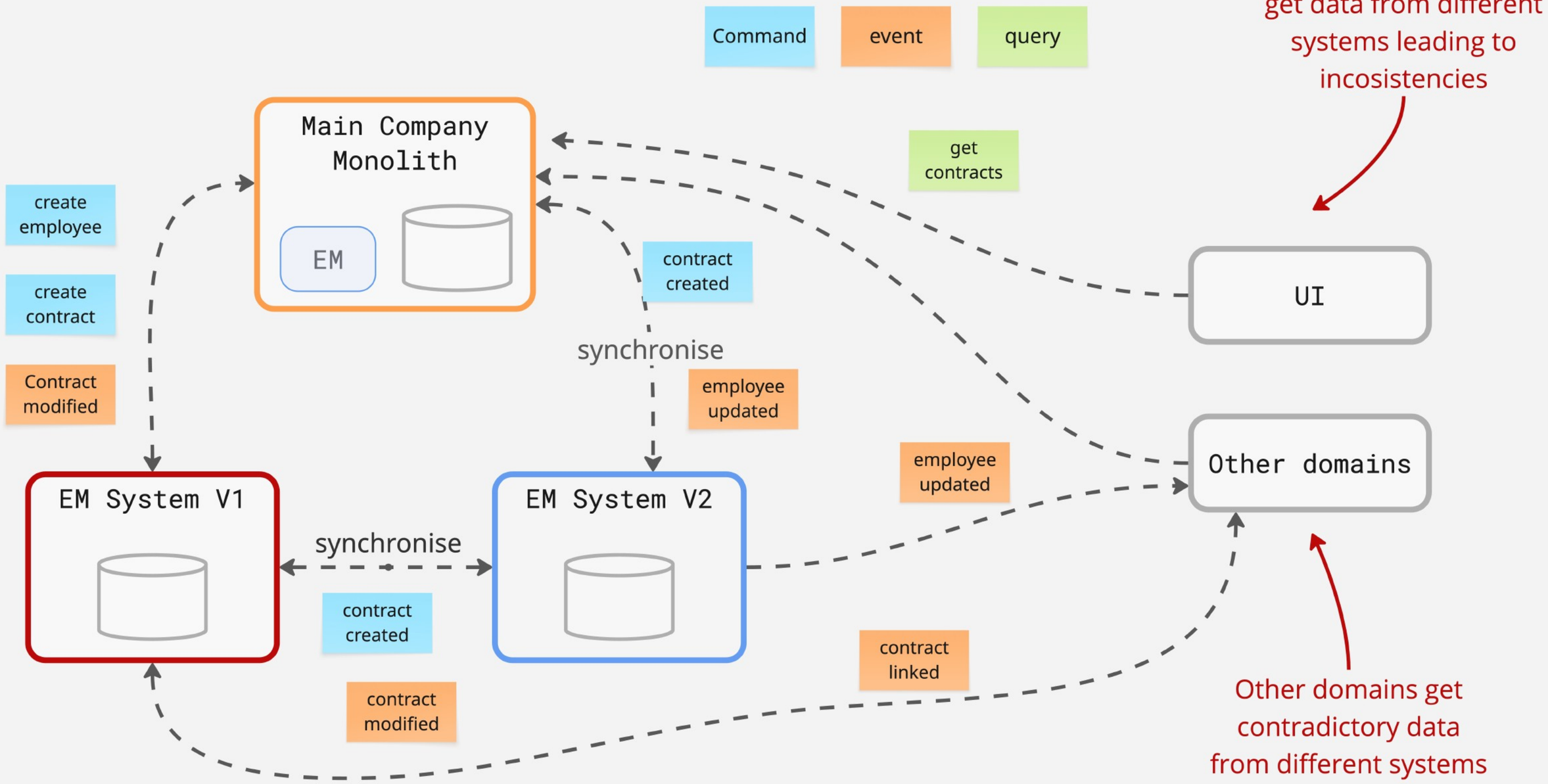
The triangle of doom

1 domain split across 3 systems (and DBs)



- Implementing new features takes longer than necessary
- Some features are impossible
- Synchronise issues affect the UX and take time to investigate and solve

A lot of use cases x3 countries

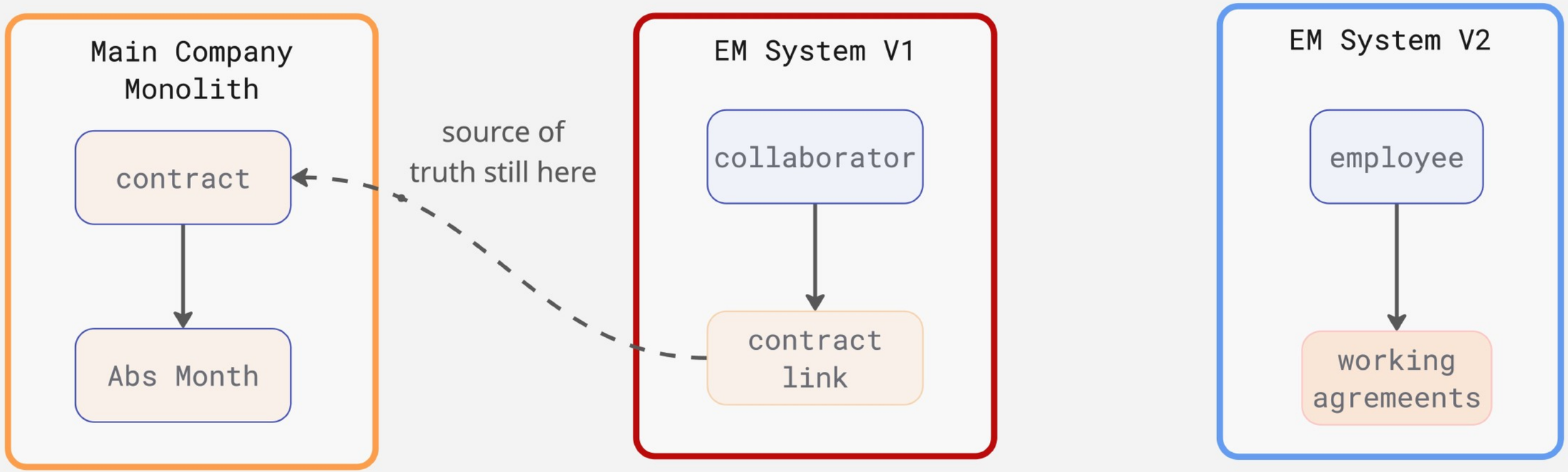


Translating domain models: Not just copying data

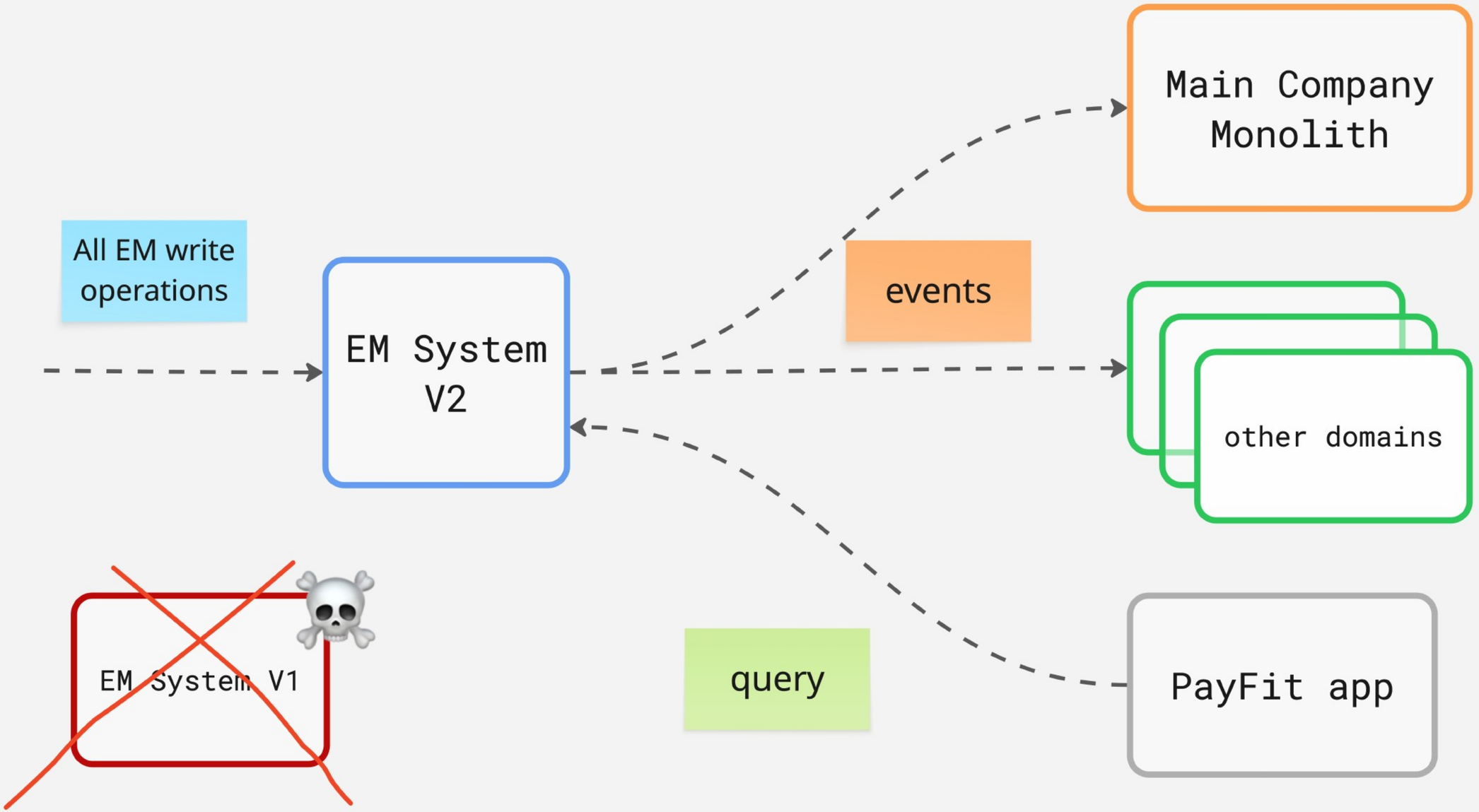
The original system that handled all use cases. A low code system

Originally built for non-payroll collaborators and link between collaborator and multiple contracts

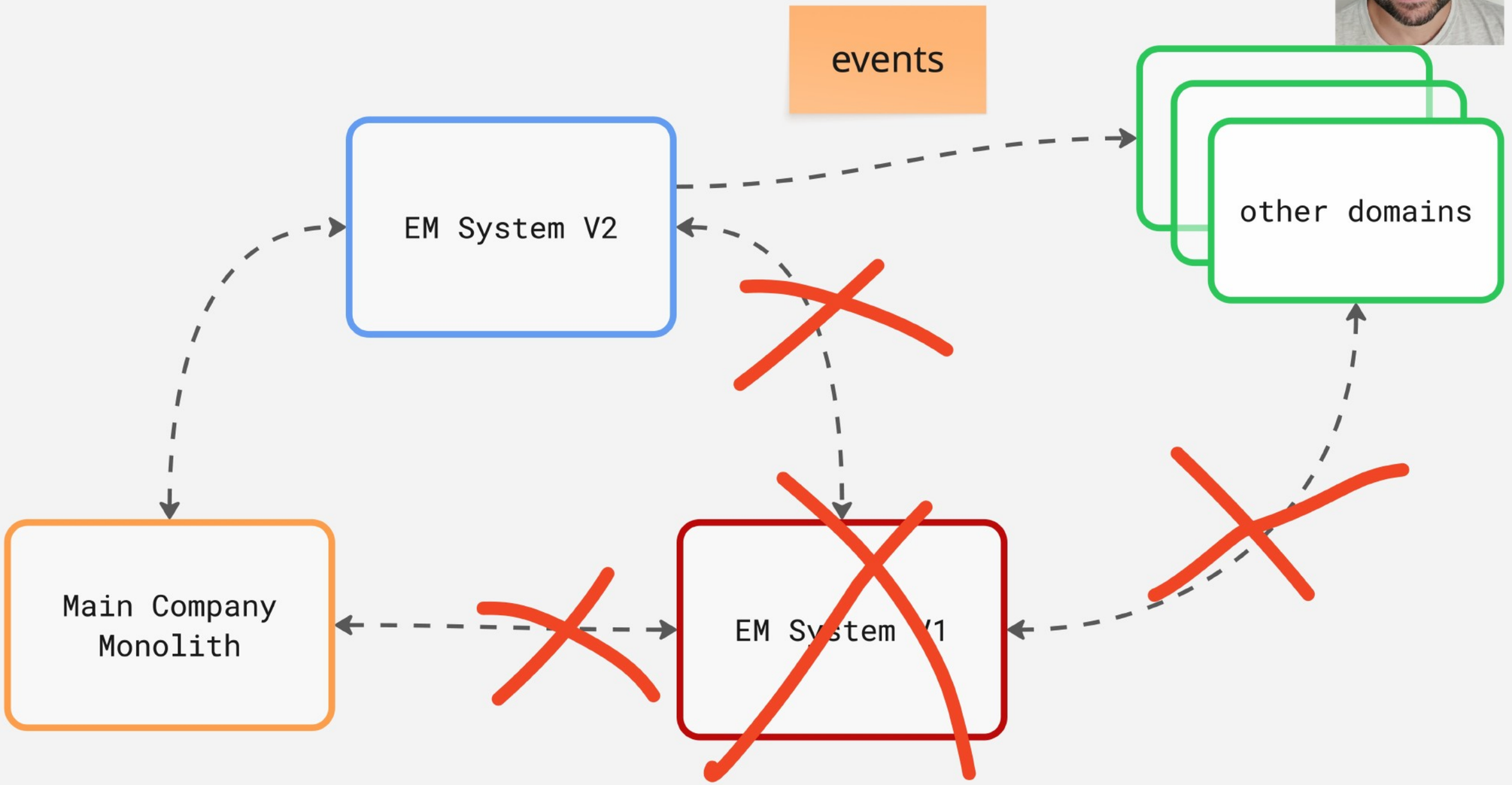
Originally built specifically to serve companies during their initial onboarding with PayFit



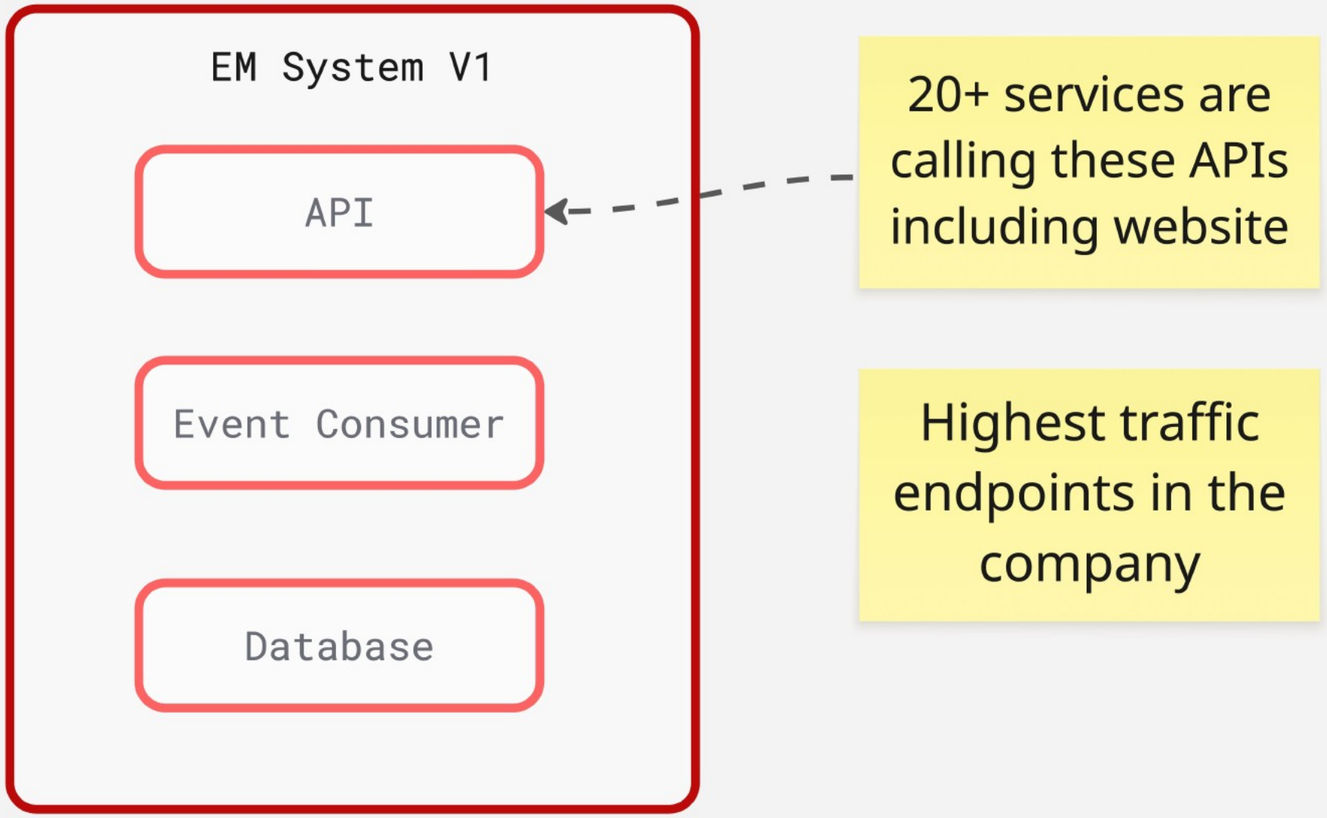
LTAV vision: consolidated domain



Charles and Nick's: V1 Retirement Plan



Charles and Nick's: V1 Retirement Plan



20+ services are calling these APIs including website

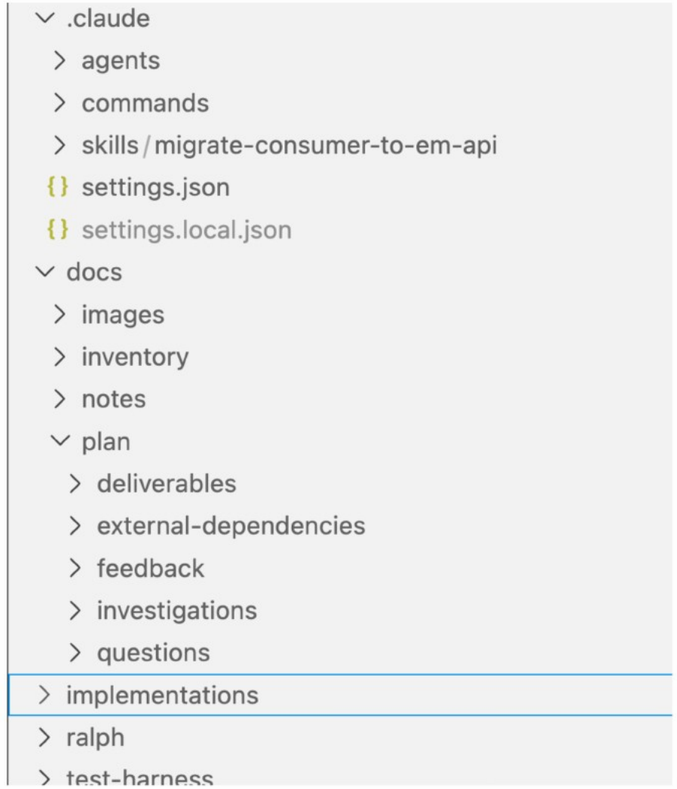
Highest traffic endpoints in the company

02

PARTIE 2

AI for Modernization Project Management

We created a git repository as our control center



- We didn't need all the functionality of JIRA
- We wanted to store all of the knowledge about our project in a place where AI could easily access it (we optimised for AI consumption)
- Full git history of all changes in the project

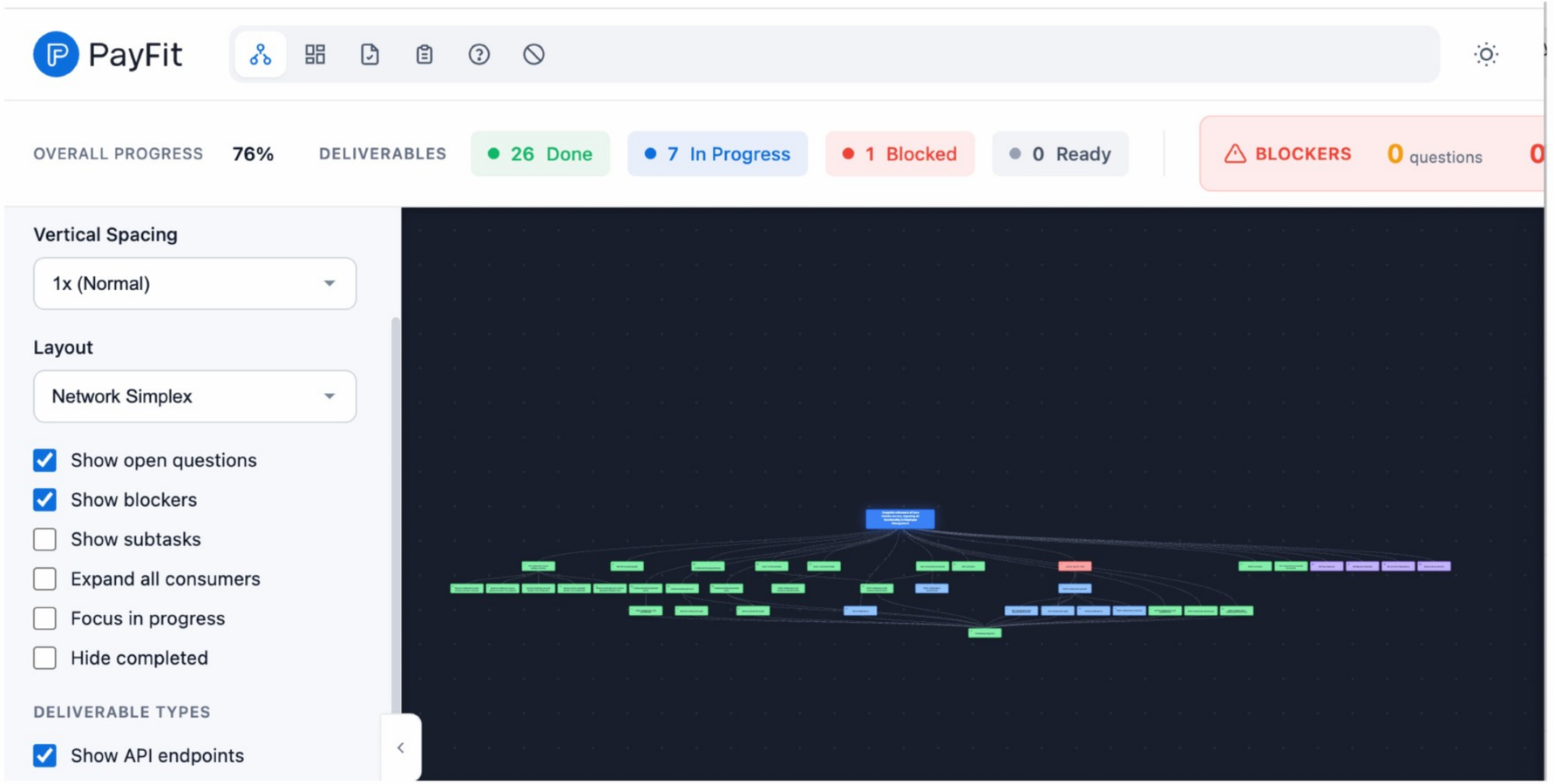
Full control over our workflow: AI commands and knowledge storage co-designed

```
/update-plan <natural language describing what changed>

1. Parse ALL updates from user's input

 The input may contain multiple paragraphs with several updates. Parse each one:
   - Scan the entire text for updates (approvals, PRs, completions, questions,
feedback, etc.)
   - For each update found, identify the target item (search by name, title, or ID)
   - Extract details (PR numbers, step IDs, resolution text, feedback content)
   - Build a list of all changes to propose
```

3. **Route to appropriate action**

Intent	Action
ADR approved	Edit deliverable: set `adr_accepted_date`
Work started	Edit deliverable: set `started_date`
Work completed	Edit deliverable: set `completed_date`
PR (no step specified)	Edit deliverable: add to deliverable's `prs[]`
PR for step	Edit deliverable: add to step's `prs[]`, set step `status: in_progress`
Step done	Edit deliverable: set step `status: done`
Add step/subtask	Edit deliverable: append to `migration_steps[]`
Multi-phase work detected	**Offer to create subtasks** - list phases found, ask user to confirm
Time-gated work ("after X weeks/days")	**Offer subtask** with time gate in description
Multiple PRs for different phases	**Ask if these are phases** needing separate subtasks
New question	Invoke `architecture-migration-skill:add-question`
Question resolved	Invoke `architecture-migration-skill:resolve-question`
New external dependency	Invoke `architecture-migration-skill:add-external-dependency`
Dependency done	Invoke `architecture-migration-skill:resolve-external-dependency`
Feedback received	Invoke `architecture-migration-skill:add-feedback`

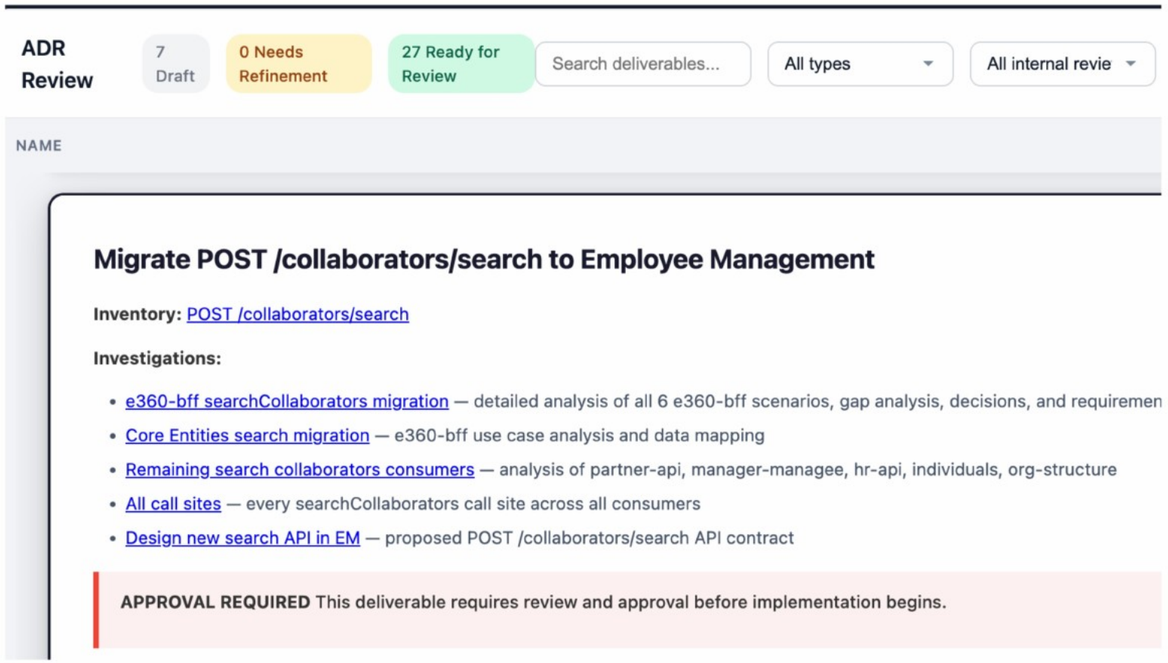
Built our own project management tool around our conventions (cost us nothing)

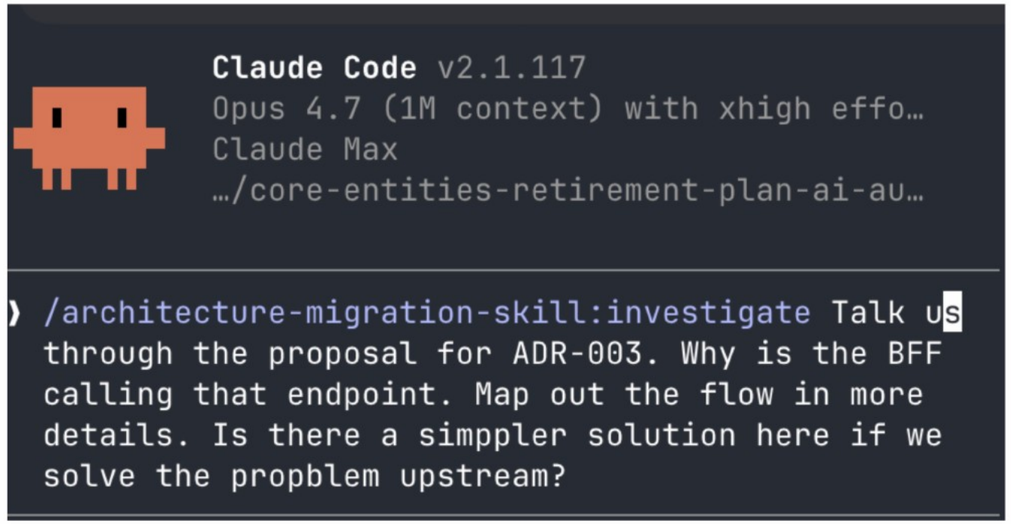


This tool cost us nothing in terms of time. AI can build tools like this quickly and in parallel with other work

ADR review with AI agent side-by-side

This was the optimal UX. We built a feature that helped us review every migration plan in the UI, and an AI that was fully aware of our project to help us dig in an explore each proposal.





we had both open side-by-side

You store everything in markdown and build project management tools on top of that?

YAML frontmatter provides structured data for our tools and AI agents

```yaml
---
id: migrate-get-collaborators
name: GET /collaborators
title: Deprecate GET /collaborators (passthrough to search)
type: endpoint
started_date: null
completed_date: null
dependencies:
  - migrate-ce-database-to-em
prs: []
category: B
risk: low
effort: S
target_service: employee-management
inventory_id: docs/inventory/core-entities/endpoints/get-collaborators.md
internal_review_status: ready_for_review
adr_accepted_date: "2026-01-28"
migration_steps:
  - id: step-1
    description: >-
      Migrate data integration team consumer — they may switch to querying
      the EM database directly (no API needed)
    started_date: "2026-03-27"
    completed_date: null
    prs: []
---
```

- Each of our project management artefacts contains YAML front matter
- Our tooling uses this data - it doesn't rely on AI agents reading markdown files and extracting values

Hallucination proof: Frontmatter has a schema which is validated on every change

```
<schema-validation-errors>
    SCHEMA VALIDATION ERRORS -
    1 files need fixing:
      docs/plan/deliverables/migrate-partner-api-bff-to-em.md:-
    root: must NOT have additional properties"consumers"
</schema-validation-errors>
```

03

PARTIE 3

Mapping out the current state inventory with AI

Mapping out the current state manually would take 1 person multiple **weeks**.

Can we do that **reliably** with AI in 2 **hours**?

Simple principles for accurate analysis results

1. Use AI only where deterministic approaches are not possible
2. Benefit from fresh context windows whenever possible
3. Force analysis precision and depth
4. AI & human verification steps
5. Throw away and re-rune

Use deterministic tools

OpenAPI specs as source of truth for API endpoints

EventCatalog as source of truth for events (including publisher and subscribers)

Nx as source of truth for codebase structure and package dependencies

Benefit from fresh context windows wherever possible

/discover-inventory

```
For each **source** repository:

1. List endpoint files: `docs/inventory/<source-repo>/endpoints/*.md`
2. Spawn subagent **api-analyzer** for each endpoint (can run in parallel) with:
 - inventory_path: path to the endpoint file
 - repo_path: from `reference-sources.md`

Wait for all analyzers to return `FINISHED`.

...
```

Agents will often optimise for efficiency or "best effort".

For discover tasks that's not good enough. Your plans are based on this information. It must be as accurate as possible

Force precision: define the algorithm not the end state

Agent: api-analyzer

```
### Step 2: Deep Code Walkthrough

Follow the complete code path from route to response:

1. Route definition → Controller method
2. Controller → Service layer
3. Service → Repository/Data access
4. Service → External API clients

Document each step with **exact file path and line number**.

> **PayFit default:** NestJS is the standard framework. Look for these patterns
first. Otherwise, find equivalent patterns for the framework in use.
```

Step 9: Generate Sequence Diagram

Create a mermaid sequence diagram showing the full request lifecycle.

```mermaid
sequenceDiagram
 participant Client
    participant Controller as ContractController
    participant Service as ContractService
    participant Validator as ValidationPipe
    participant Repo as ContractRepository
    participant External as CoreEntitiesClient

    Client->>+Controller: POST /contracts
    Controller->>+Validator: validate(dto)
    Validator-->>-Controller: validated
    Controller->>+
```

Force precision: checklists & review

Step 11: Quality Checklist

Before completing, verify:

- [ ] Handler file path and line number documented
- [ ] Context diagram has actual consumers and dependencies (or placeholders)
- [ ] Sequence diagram shows complete request flow
- [ ] Request body has full TypeScript interface (not just field names)
- [ ] Response has full TypeScript interface
- [ ] All database operations have fields and conditions
- [ ] All external calls have exact endpoint, request body, and response fields used
- [ ] Business rules extracted and numbered
- [ ] Error handling documented with HTTP codes

Accuracy verification

1. ran **/discover-inventory** multiple times, compare the results and assess anomalies

2. Use multiple models for both generation and validation

3. Manual code walkthrough: are diagrams accurate?

Agent: inventory-validator

Step 6: Validate Diagram Accuracy

If inventory has a mermaid diagram:

1. Parse diagram nodes (services, databases, consumers)
2. Verify each node corresponds to actual code artifact
3. Check edges match actual call relationships

If diagram doesn't match reality:
Create question: `q-validate-<id>-diagram-inaccurate

04

PARTIE 4

Shaping the target architecture with AI

/create-deliverables

Create migration deliverables from discovered endpoints and events.

...

3. Spawn migration-architect (5 at a time)

Max parallelism 5 to avoid GitHub rate limits.

Process inventory files in batches of 5.

For each item in the batch, spawn the **migration-architect** subagent with arguments:
- inventory_path: path to the inventory file

Run all 5 subagents in parallel. Wait for all to return `FINISHED` before starting the next batch.

...

Domain fit: Where should code be migrated to?

Agent: migration-architect

```
### Step 2: Score Domain Fit Options

**Populates:** `## 4. Options Considered > ### Domain Fit` table

Assess based on:

- **Semantic alignment:** Does endpoint name match target domain language? Do related responsibilities
belong to same domain?
- **Data alignment:** Does endpoint access same entities as target? Does target already own this data?

For each option, provide confidence + specific rationale:

- target (assumed service)
- other:<service-name>
- split
- deprecate

...
```

Endpoint: Where should consumers be migrated to?

Agent: migration-architect

```
### Step 3: Score Consumer Destination Options

**Populates:** `## 4. Options Considered > ### Consumer Destination` table

**3.1 Gather evidence**

From inventory:

- Endpoint contract (fields, types, response shape)
- Consumer list with usage patterns
- Traffic volume

From target service:

- Existing endpoints
- Their contracts
- Their consumers
```

Agent: migration-architect

3.2 Compare contracts

- Count field differences
- Identify semantic mismatches (same name, different meaning)
- Identify semantic equivalences (different name, same data)
- Note shape differences

3.3 Analyze consumer needs

Per consumer:

- What fields do they use?
- What access pattern?
- What SLAs?

3.4 Score options with evidence

Option	When Applicable
Existing endpoint in target	Target already has equivalent
New endpoint in target	No equivalent exists
Modify existing endpoint	Equivalent exists but needs changes
Split into multiple	Endpoint does too many things
Deprecate	Zero traffic, no consumers

Which API endpoint in target to use?

Is there an existing endpoint or are changes required?

Does this endpoint need to be split (model mismatch)

/challenge-dependency

Step 2: Extract Used Fields

Analyze the handler/client code:
- Which fields from the payload/response are destructured/accessed?
- Mark each field as: USED / UNUSED / CONDITIONAL

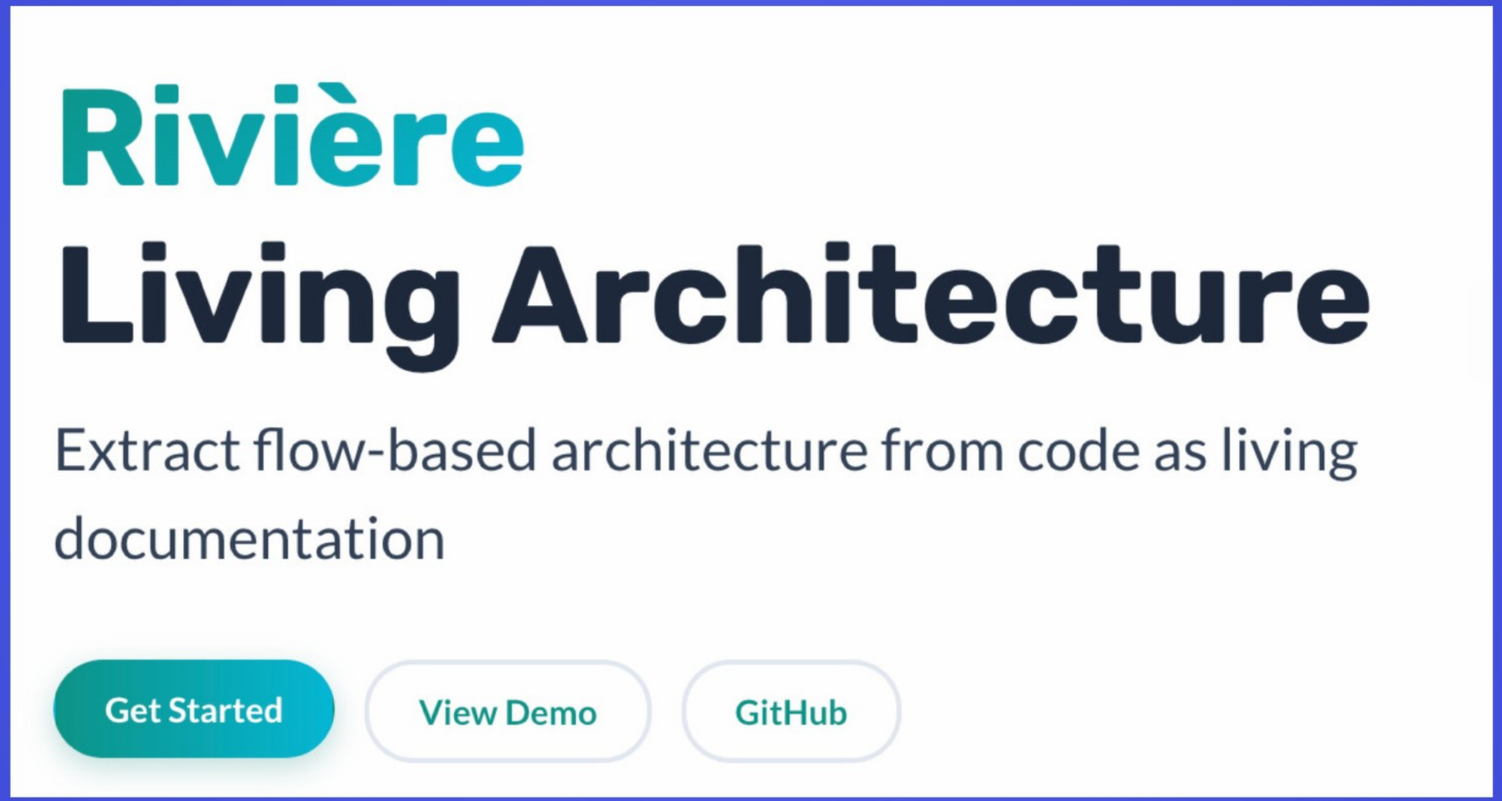
Create inventory with code references (file:line).

Step 3: Trace Each Field's Journey

For each USED field, follow the data flow until you reach a terminal point:
- DATABASE_WRITE: table and columns affected
- EXTERNAL_API_CALL: service, endpoint, data sent
- EVENT_PUBLISHED: event type, fields included
- API_RESPONSE: endpoint that returns this data

Go as deep as necessary. Do not stop at arbitrary depth limits.

That part can be deterministic as well



living-architecture.dev/

05

PARTIE 5

Executing the migration plan with AI

/create-implementation-plan

Before doing anything else, use TaskCreate to create these tasks. You MUST complete all tasks in order.

```
TaskCreate: "Git setup: pull latest changes"
TaskCreate: "Identify and load deliverable"
TaskCreate: "Extract repo guidelines from AGENTS.md/CLAUDE.md"
TaskCreate: "Explore code for implementation details"
TaskCreate: "Draft plan steps with user"
TaskCreate: "Conduct pre-mortem analysis"
TaskCreate: "Run plan-validator subagent"
TaskCreate: "Run 3 challenger subagents in parallel"
TaskCreate: "Review challenger findings with user"
TaskCreate: "Define completion criteria"
TaskCreate: "Write plan.md and tracking.md"
TaskCreate: "Get user approval for plan"
TaskCreate: "Git commit"
```

we are working in 25+ code repositories.

Some differences in tech stack, architecture, patterns, preferences etc

The more you standardize, the easier modernization will be

Human review and sign off for every plan.

As the project progressed, we ironed out problems and there was less human input required

Agent: plan-challenger-integration

Task: Challenge an implementation plan to find what breaks elsewhere and what dependencies are missed.

...

What to Look For

- Impact on other services/consumers not mentioned
- Missing coordination with other teams
- Data consistency issues across service boundaries
- API contract changes that affect consumers
- Event schema changes that break subscribers
- **Hard cutover event transitions without expand-and-contract phase**
- Database changes that affect other services
- Missing consumer migration steps

...

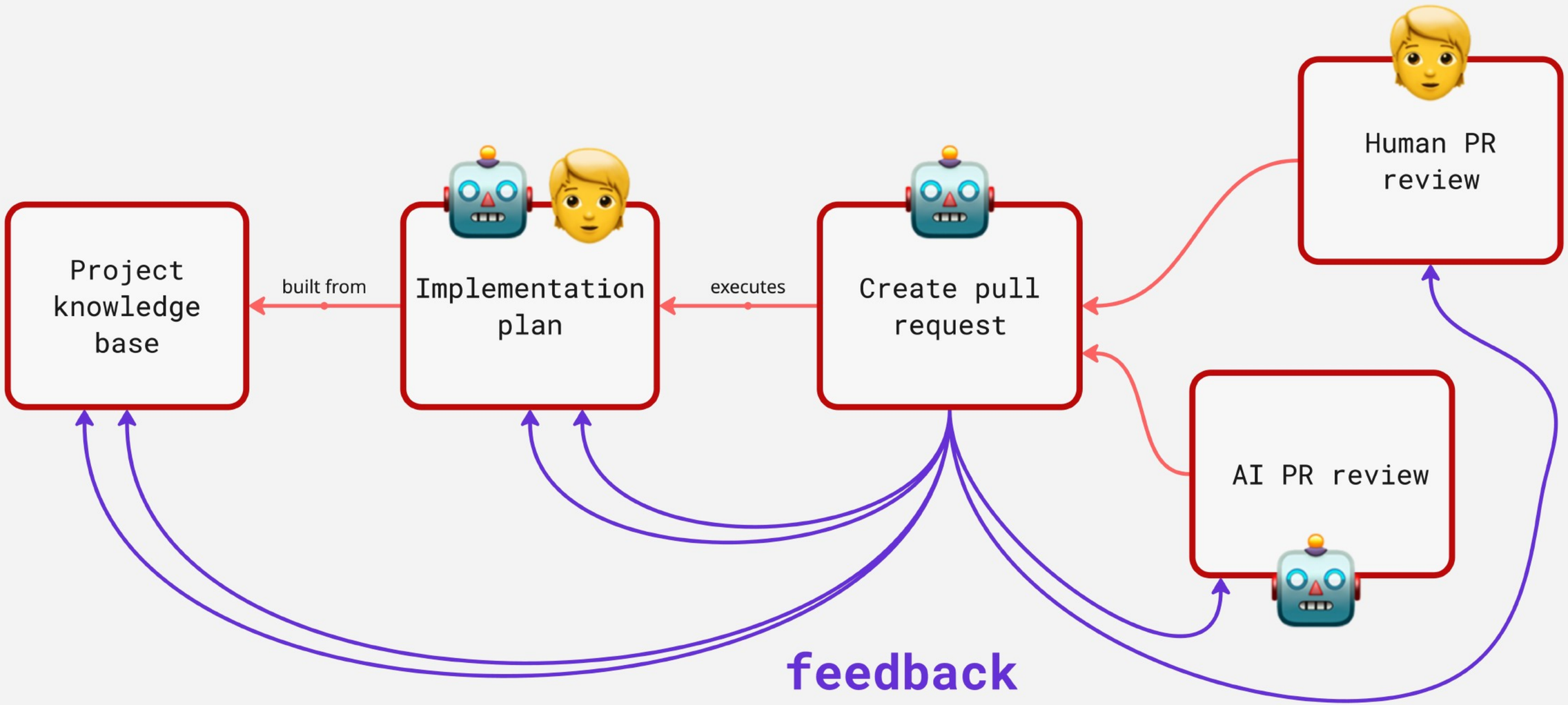
You can't just turn the old API off and turn the new on one. Usually you need expand-and-contract phase => both new and old run in parallel allowing consumers to migrate safely.

Claude Code did propose a hard cutover in one plan.

Our default was to let AI write 100% of the code.

The goal of this project was to push the limits and see what breaks.

Feedback loops: fix the system, don't patch it



Big batches can help you go faster & learn faster

1. We span up 3 - 5 agents in parallel creating PRs
2. At one point we had 15+ open pull requests
3. This helped us learn faster: new insights that broke our plan and required deeper insight
4. It helped us beat the bottleneck....

The biggest bottleneck is other teams. They need to review, approve, deploy, test PRs.

And they are busy with other things.

AI is limited here, but there is potential

DIY Kit

We created a skill for teams to do their own migrations if they didn't like our default approach.

```
# Migrate Consumer to EM API

Migrate a consumer service from `<system 1>` SDK to Employee Management
API. Up to three places change: **consumer code** (always), **env var
config** (always — ArgoCD or in-repo), and **EM infra** (...).

## Auto Mode

When invoked with `--auto` (e.g., `/migrate-consumer-to-em-api --
auto`), skip all interactive questions and use defaults:

| Decision | Auto default |
|----------|-------------|
| HTTP client | Fetch |
| Feature flag | No |
| PR mode | Draft |

...
```

Wrap up

Where AI genuinely helped

1. Build our optimal workflow & tooling
2. Project assistant who knows everything about the project and can assist with any topic
3. Map out current state 10 - 20x faster in more detail
4. Execute 2x - 3x faster
5. Reduce cognitive load of context switching -> allowed us to mage progress on multiple streams in parallel

AI isn't a magic wand. The models by themselves won't give you those benefits

Building the knowledge base and workflow was more important than the model

Where AI needs more investment to be helpful

1. Complex dependency chains that span multiple repos & require higher-level architectural knowledge

2. Cross-team dependencies

3. Domain / data model mismatches

4. Codebases and services with their own special way of doing things or infrastructure setup

My Personal Recomendation

Anthropic marketing department want you to believe Claude is a "competent senior engineer" who you can delegate to. I disagree.

I think that AI can provide huge benefits to your modernisation project, but you should rely on the model as little as possible.

1. Turn your expertise into commands and workflows

2. Continuously bake-in learnings so that your workflow produces higher quality & fewer mistakes each time

3. Stay close to the code and understand what AI is doing

careers.payfit.com/