



# Legacy Architecture Migration Patterns

Nick Tune



/in/nick-tune

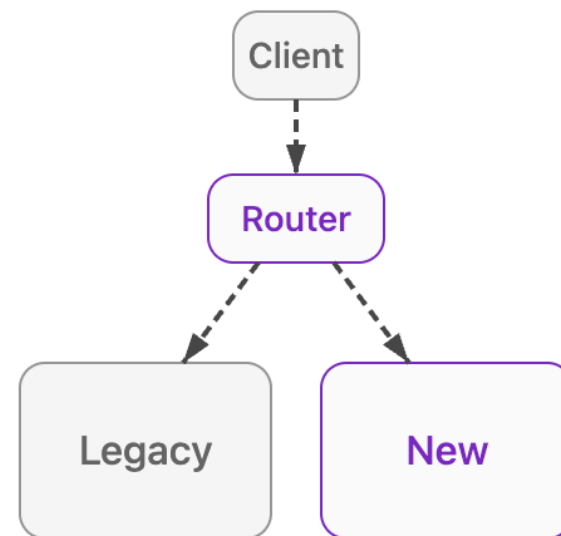


# Talk structure

1. Key migration principles
2. Common migration patterns
3. Case study 1: write model migration
4. Case study 2: legacy event republishing

# Lots more patterns here

## LEGACY-MODERNIZATION.IO

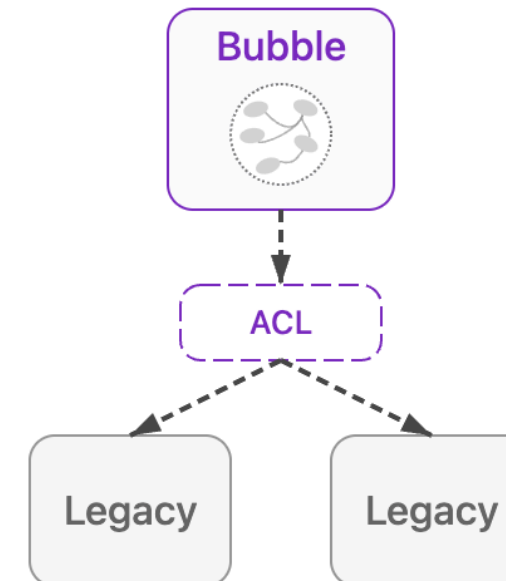


### MIGRATION

#### Strangler Fig

The Strangler Fig pattern is a metaphor invented by Martin Fowler to describe a gradual migration away from a legacy system to a new system (aka the strangler fig application).

[Read more →](#)



### MIGRATION

#### Bubble

The Bubble pattern is about creating a new domain model to encapsulate and gradually replace a specific subdomain/subsystem. Eventually, the bubble should pop.

[Read more →](#)

# Come and join us...



**[careers.payfit.com/#jobs](https://careers.payfit.com/#jobs)**

Senior Software Engineer - Permission  
& Identity team

Engineering · Paris · Hybrid 📶

Platform Engineering Director

Engineering · Paris · Hybrid 📶

**1.**

# **Key Migration Principles**

**We are not just rewriting the old code in  
new tech**

*"Pendo determined that 80 percent of  
features in the average software product  
are rarely or never used."*

*Pendo 2019 Feature Adoption Report -*

*<https://go.pendo.io/rs/185-LQW-370/images/2019%20Feature%20Adoption%20Report%20Digital.pdf>*

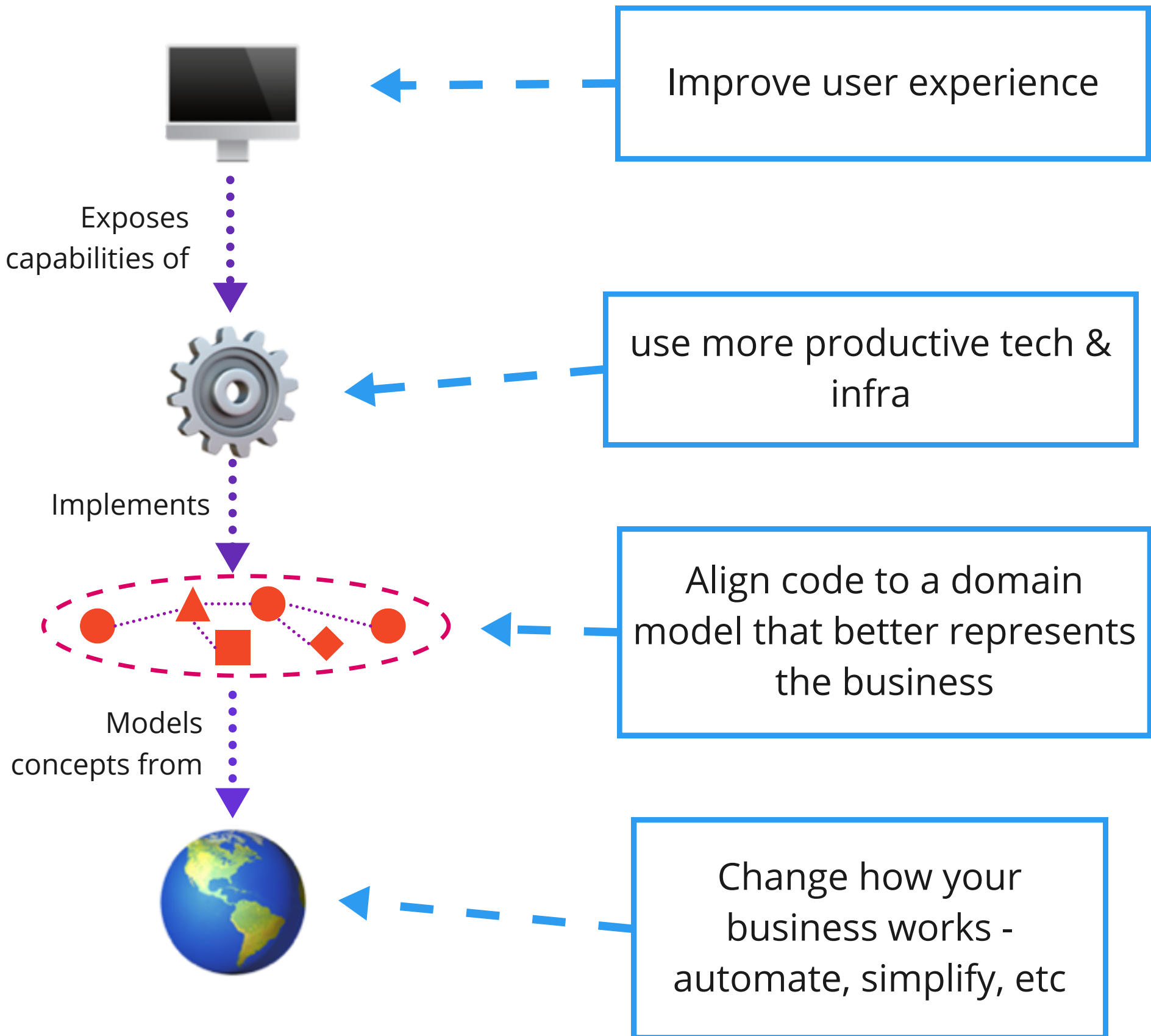
# Full-stack modernization

User Interface

Software & Infra

Domain Model  
(conceptual)

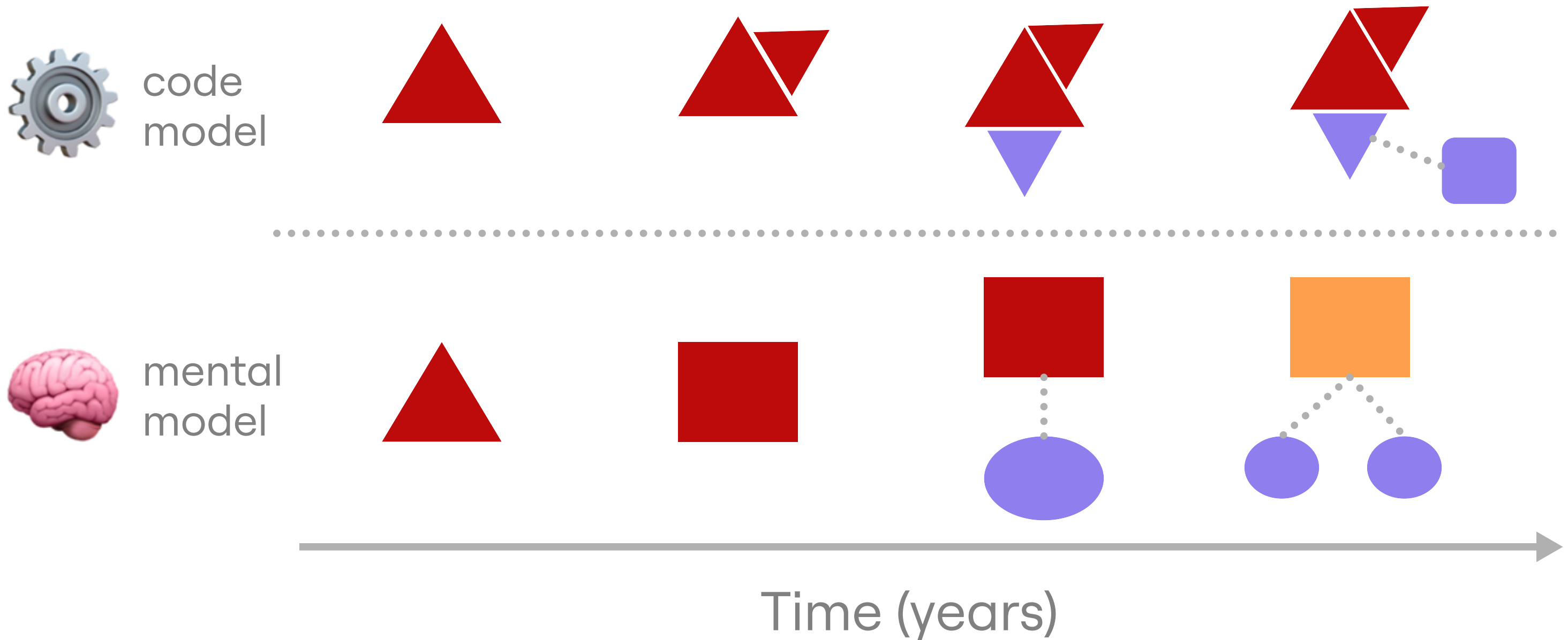
Domain



# Legacy-crystallized semantic drift

The concepts in our legacy become more and more misaligned with our mental model over time because legacy is hard to change

Modernization is a chance for semantic convergence





*One reason adding new features was more complex than it could have been was the assumptions baked into the architecture.*

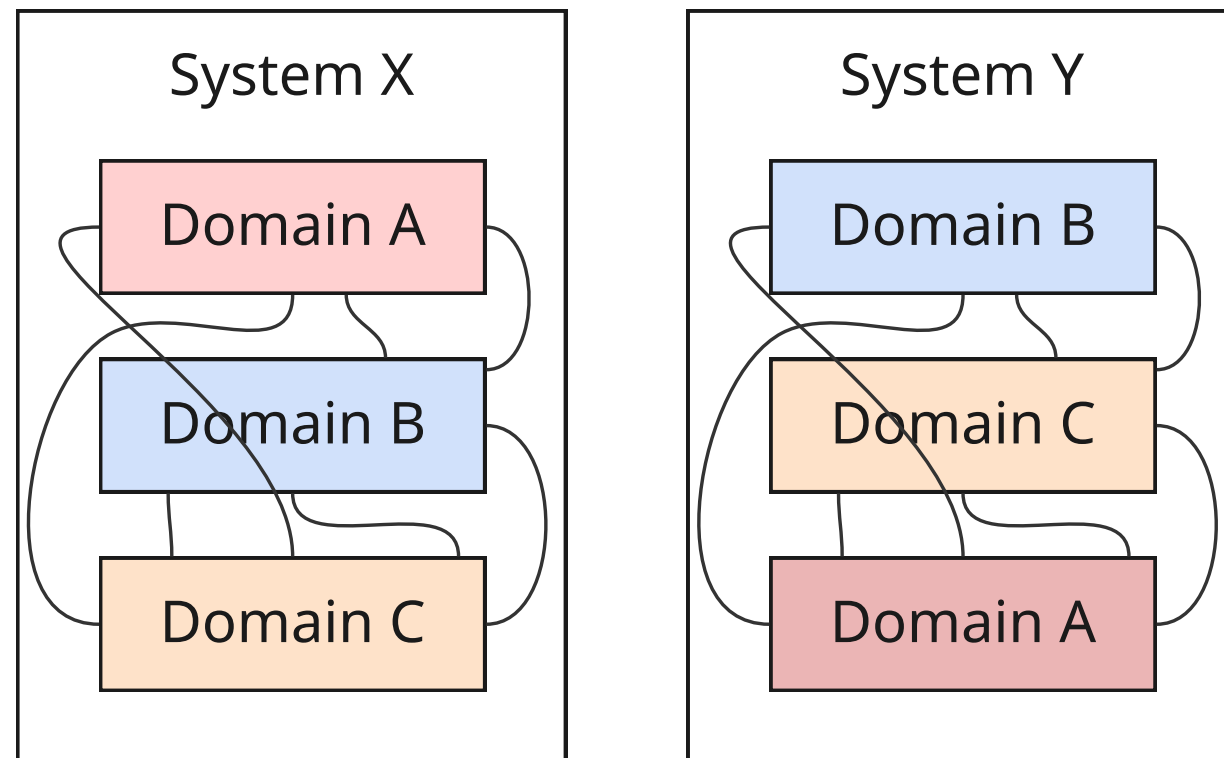
*Many of the core abstractions were still heavily based on the assumption of a single vertical (women's clothing)*

-- Ornela Vasiliauskaite, Vinted

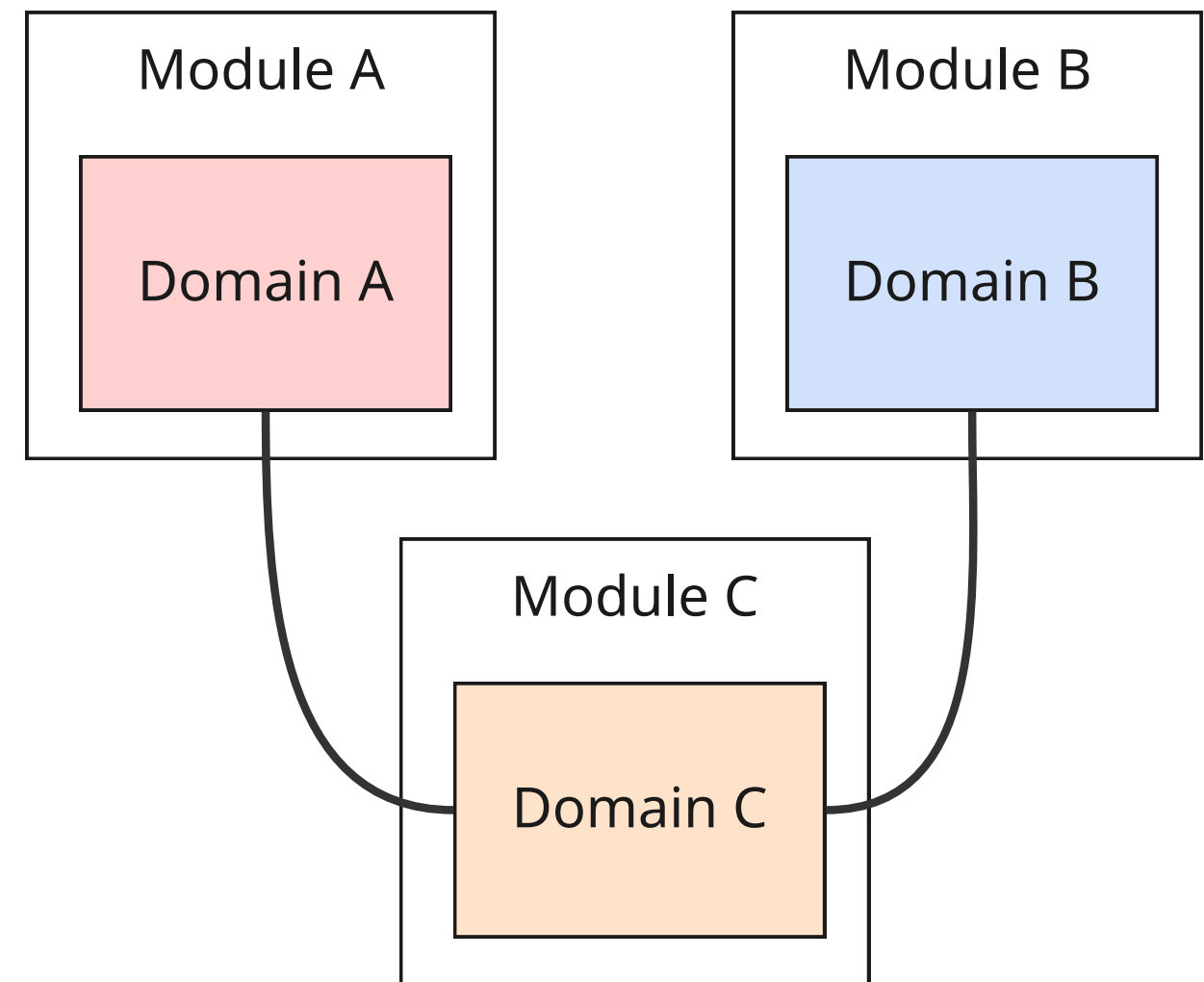


# Domain boundaries need to evolve to reduce coupling + often team boundaries

Legacy Boundaries  
(low modularity)



Target Boundaries  
(good modularity)



# Legacy migration requires advanced observability

More systems running in parallel +  
more data to be synchronised creates the  
potential for more problems and more complex  
problems.

This is a recipe for spending a lot of time  
investigating + fixing problems and less time  
progressing.

**2.**

# **Effective Migration Patterns**

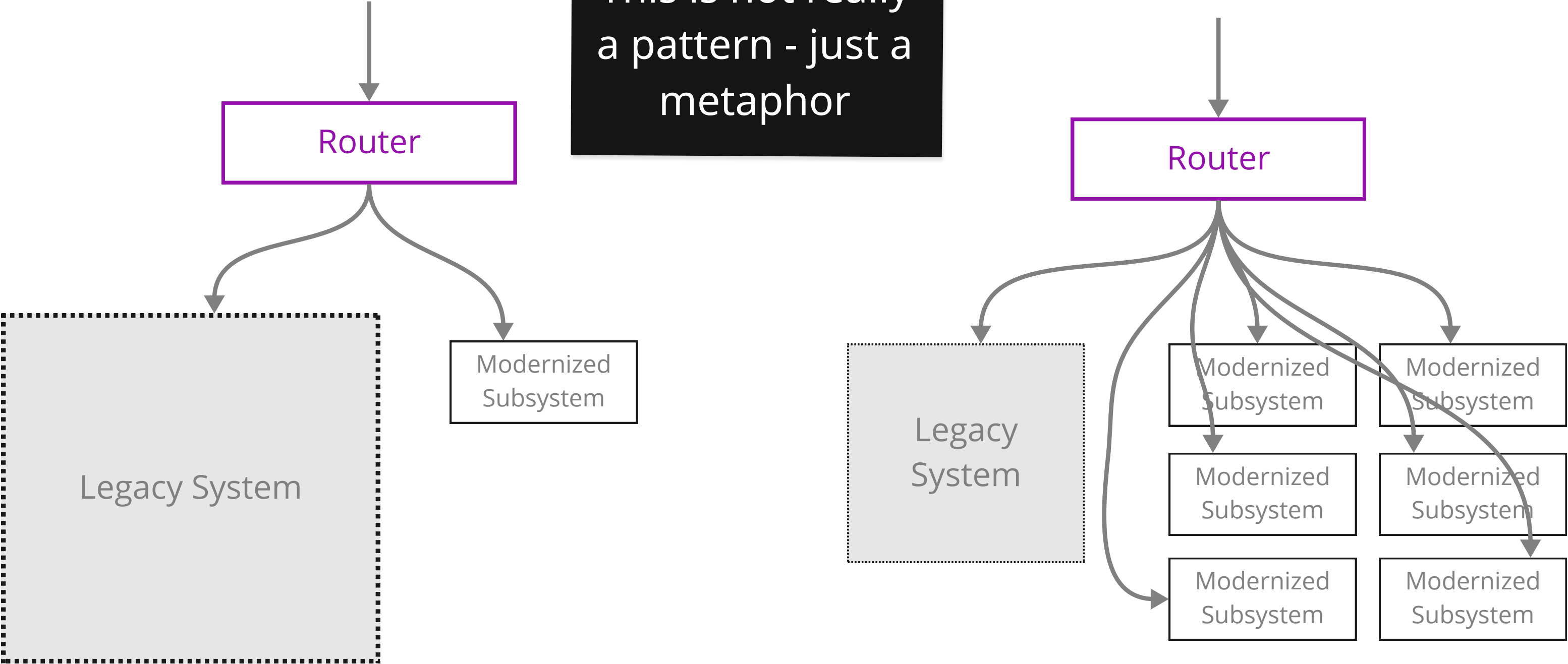


Pattern definitions are quite broad and implementations of the same pattern can look very different in different contexts

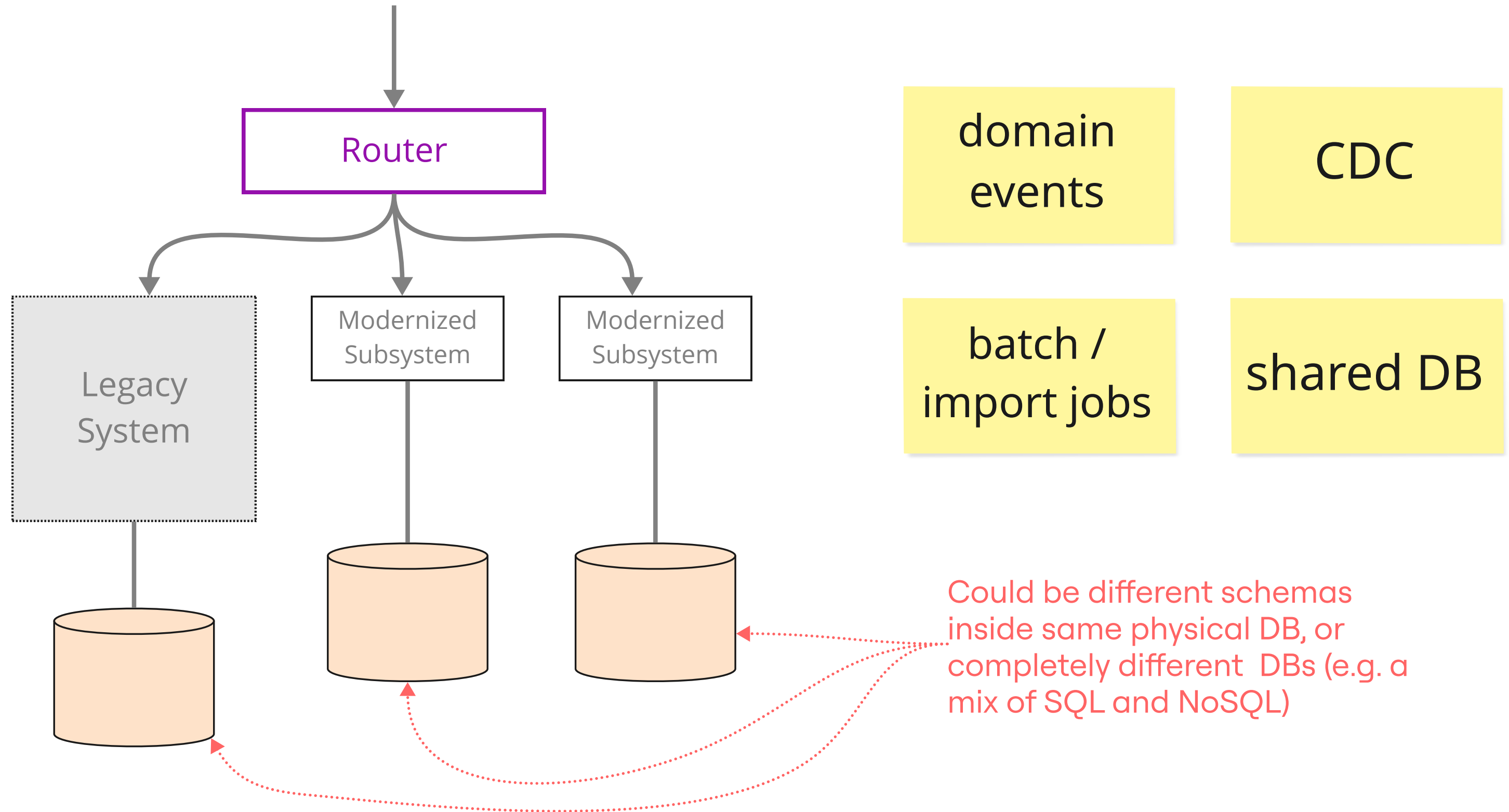


# Strangler Fig (always the first thing everyone says)

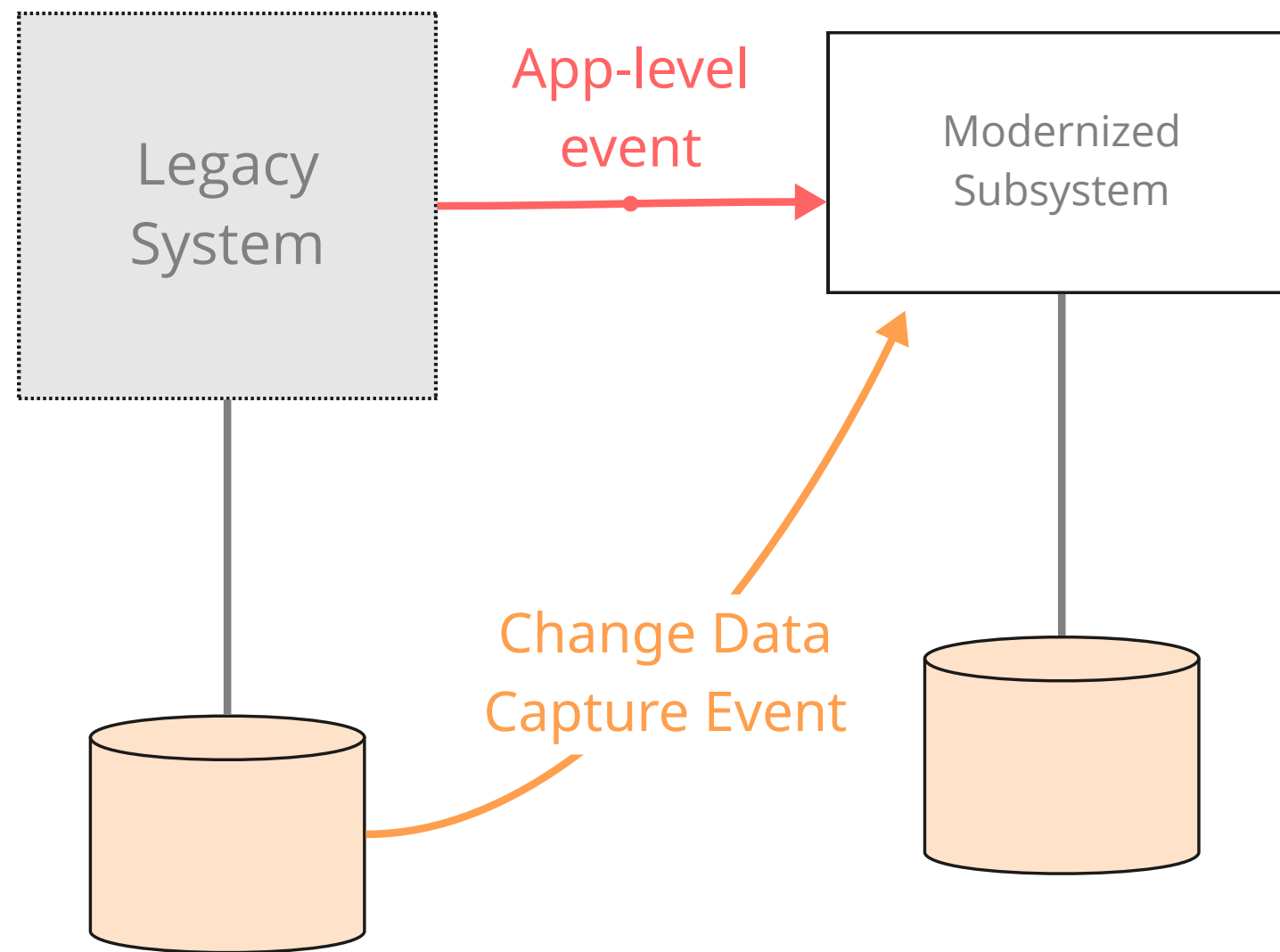
This is not really  
a pattern - just a  
metaphor



# But what about the Data?



# CDC vs Application-level events



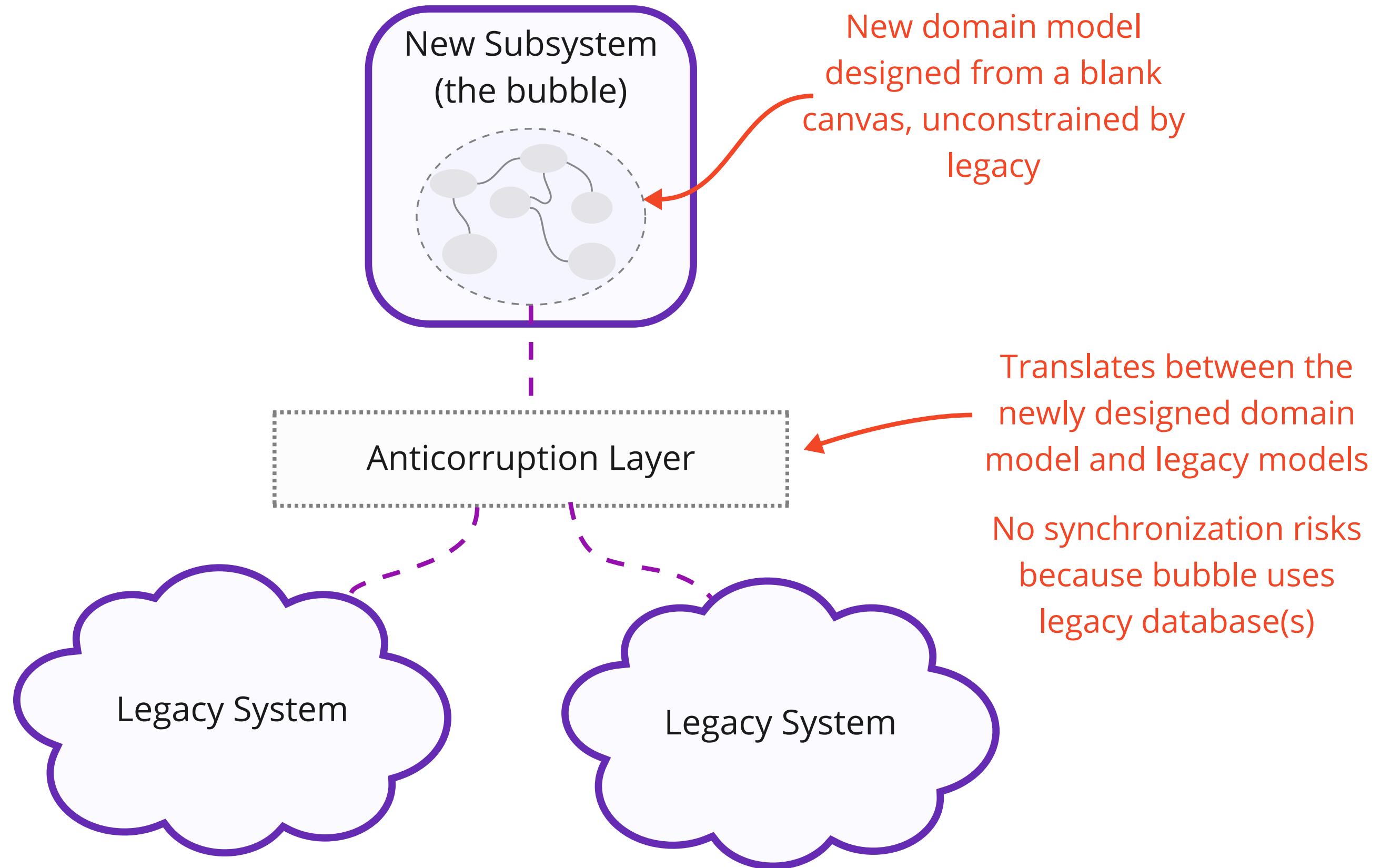
## CDC

- ✓ Easier to setup in legacy
- ✓ Catches all data modification scenarios

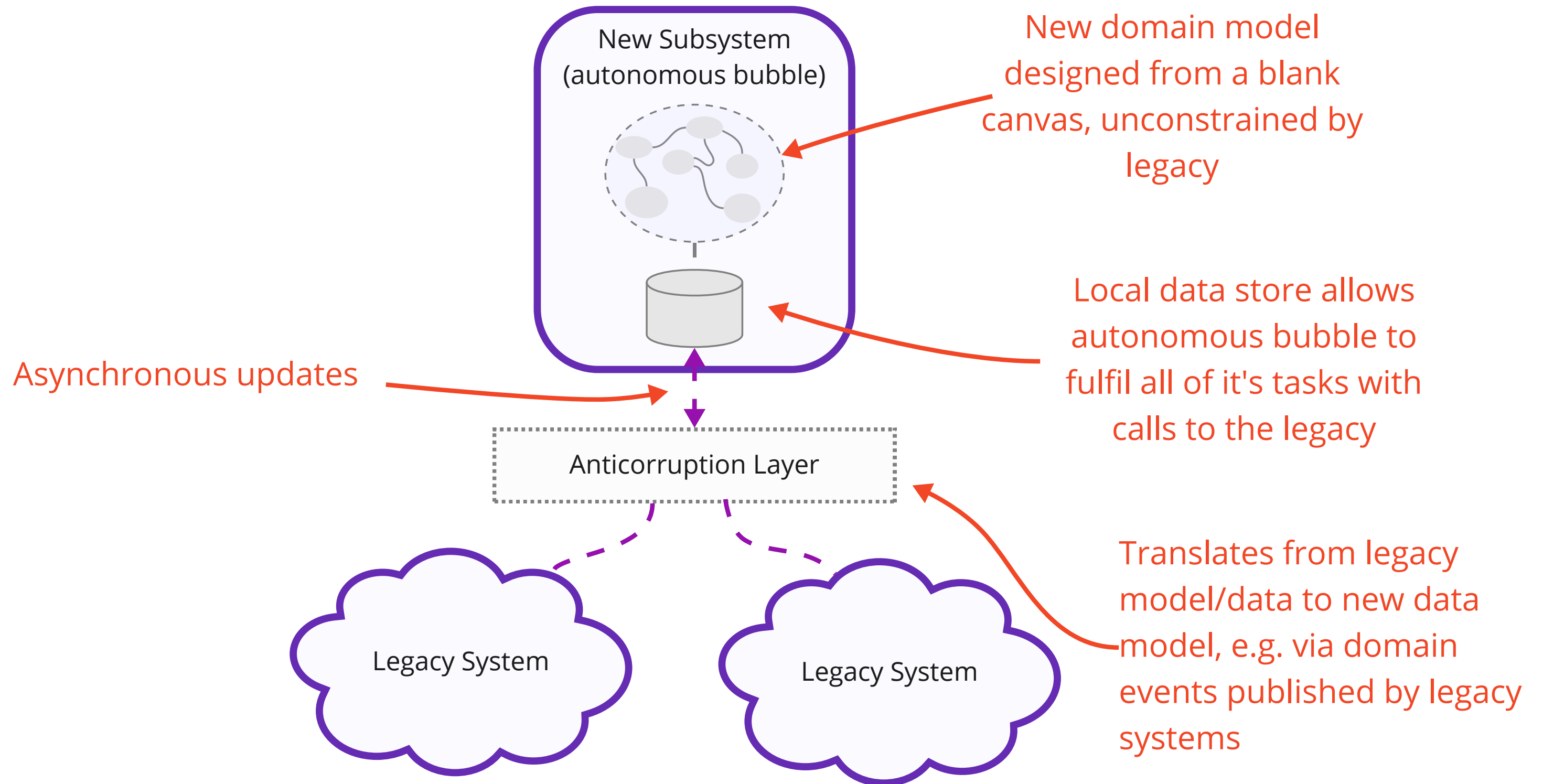
## App-level

- ✓ Provides more context about the event
- ✓ Avoid leaking legacy model & coupling to legacy DB

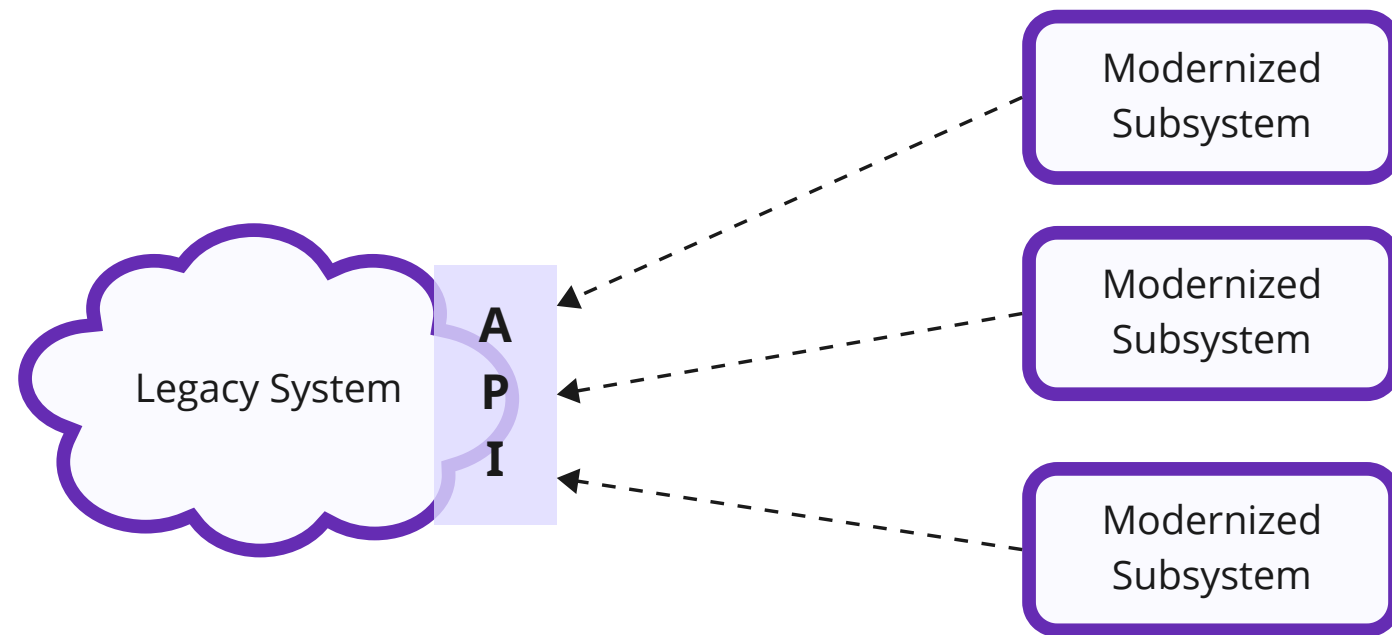
# Bubble



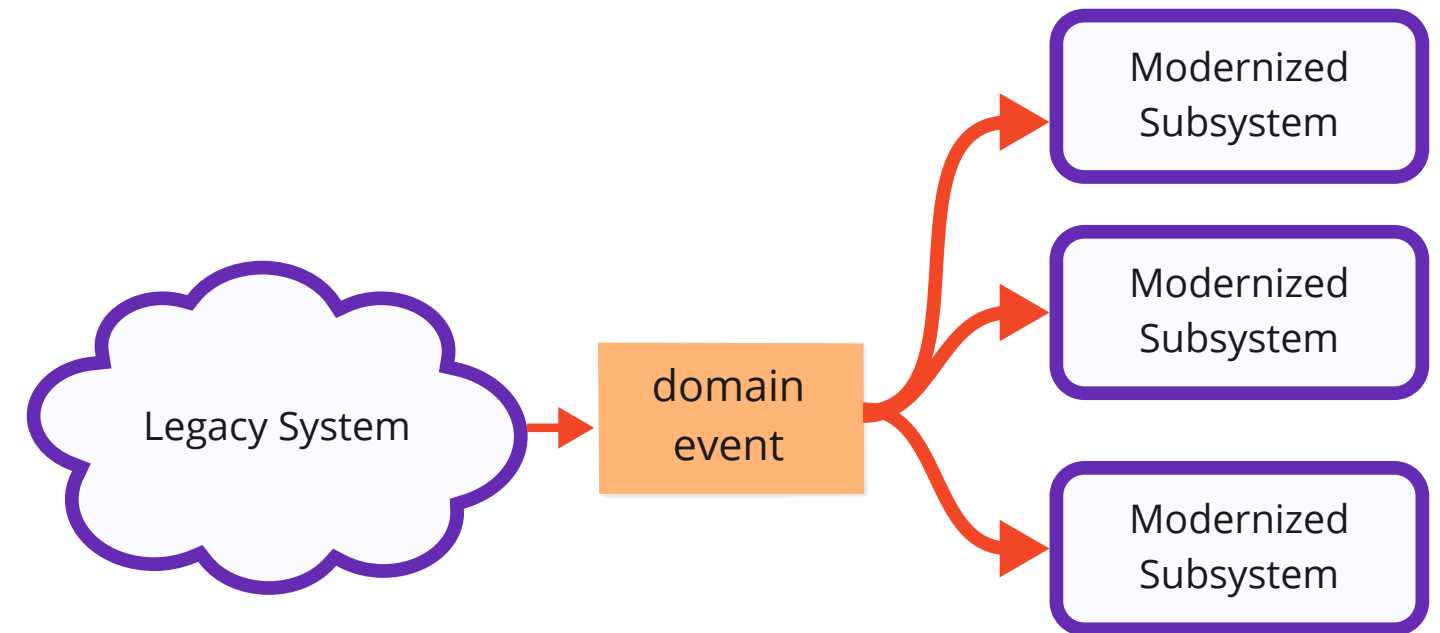
# Autonomous Bubble



# Expose Legacy Asset



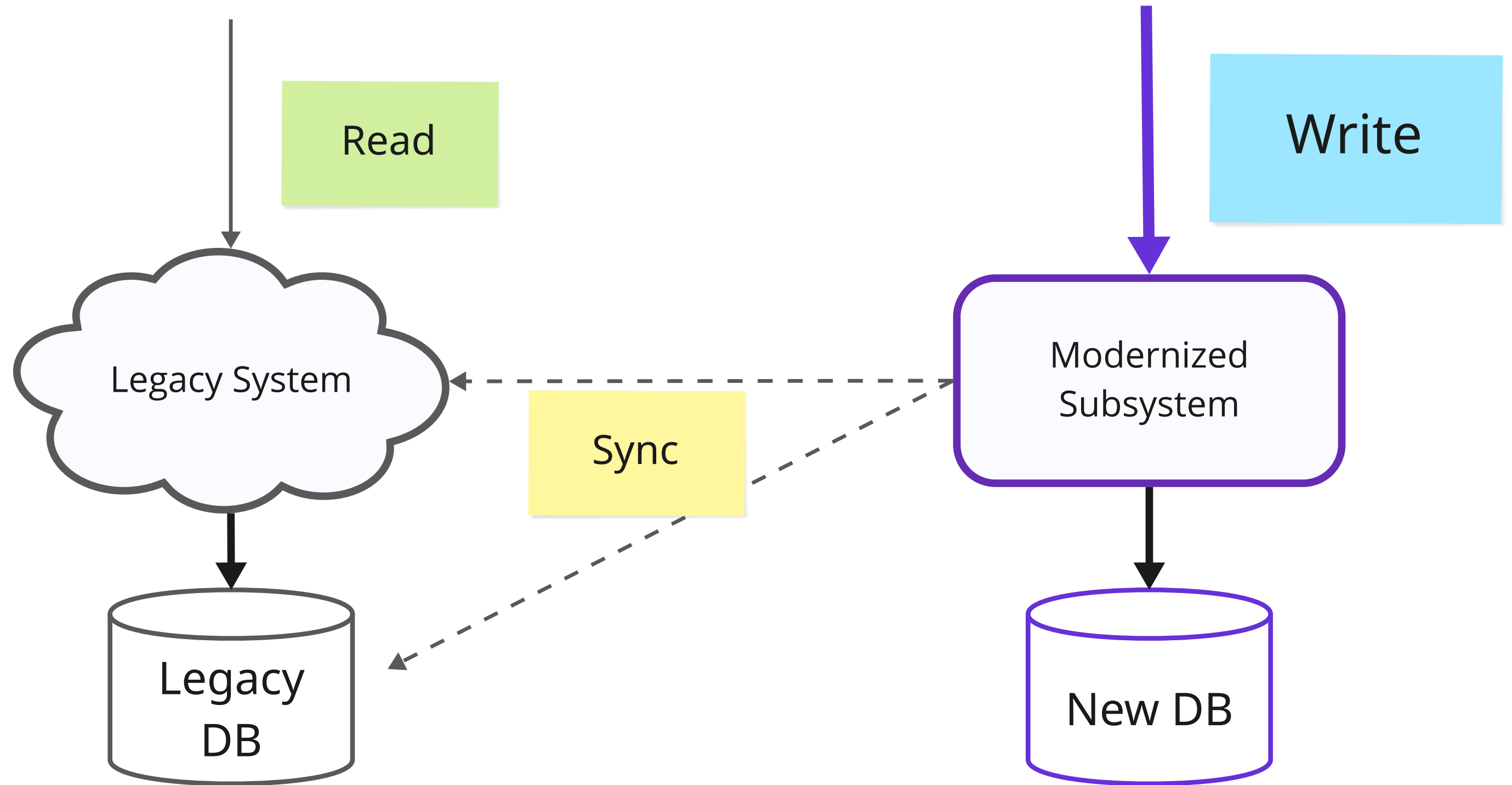
Expose legacy via  
API



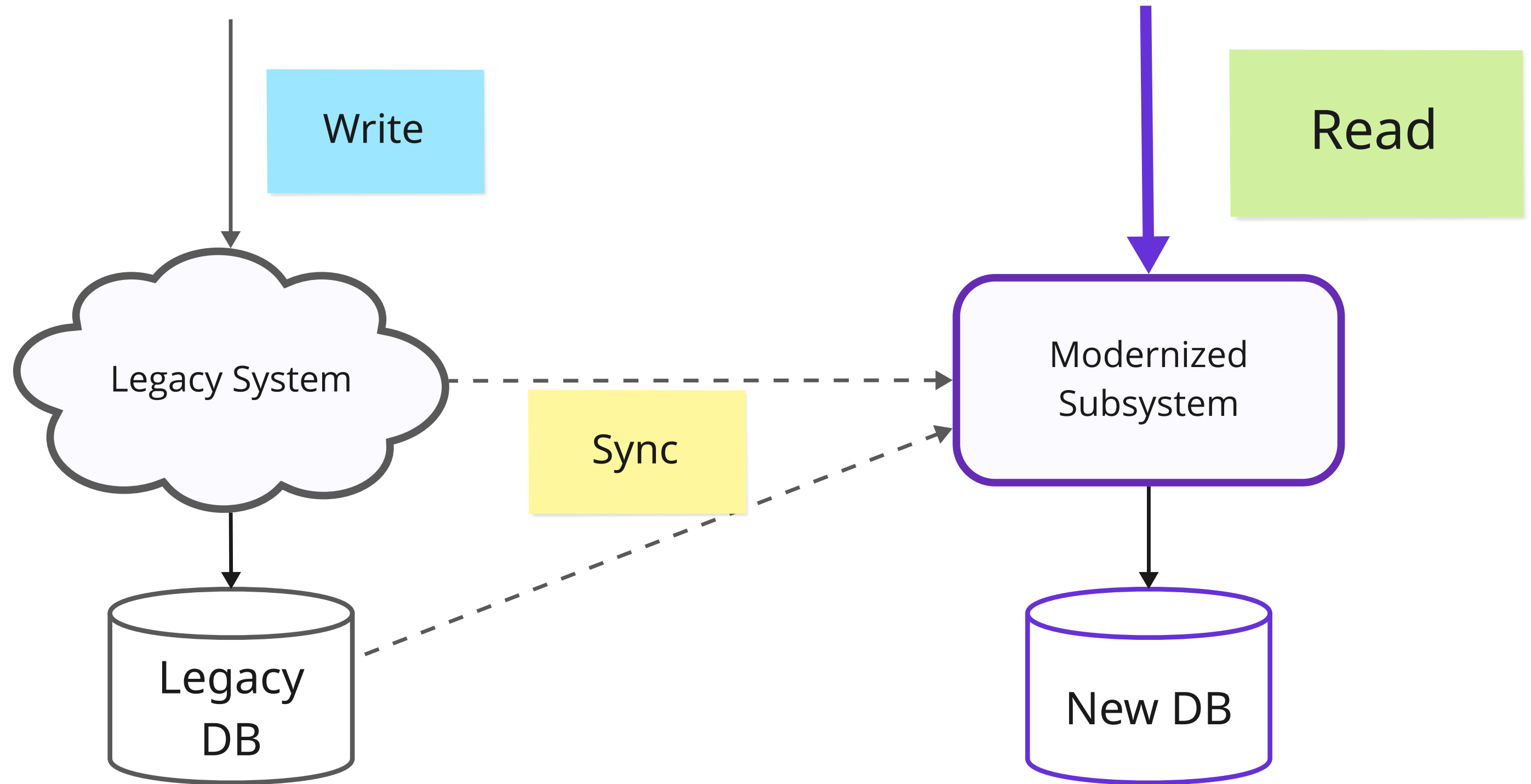
Expose legacy via events



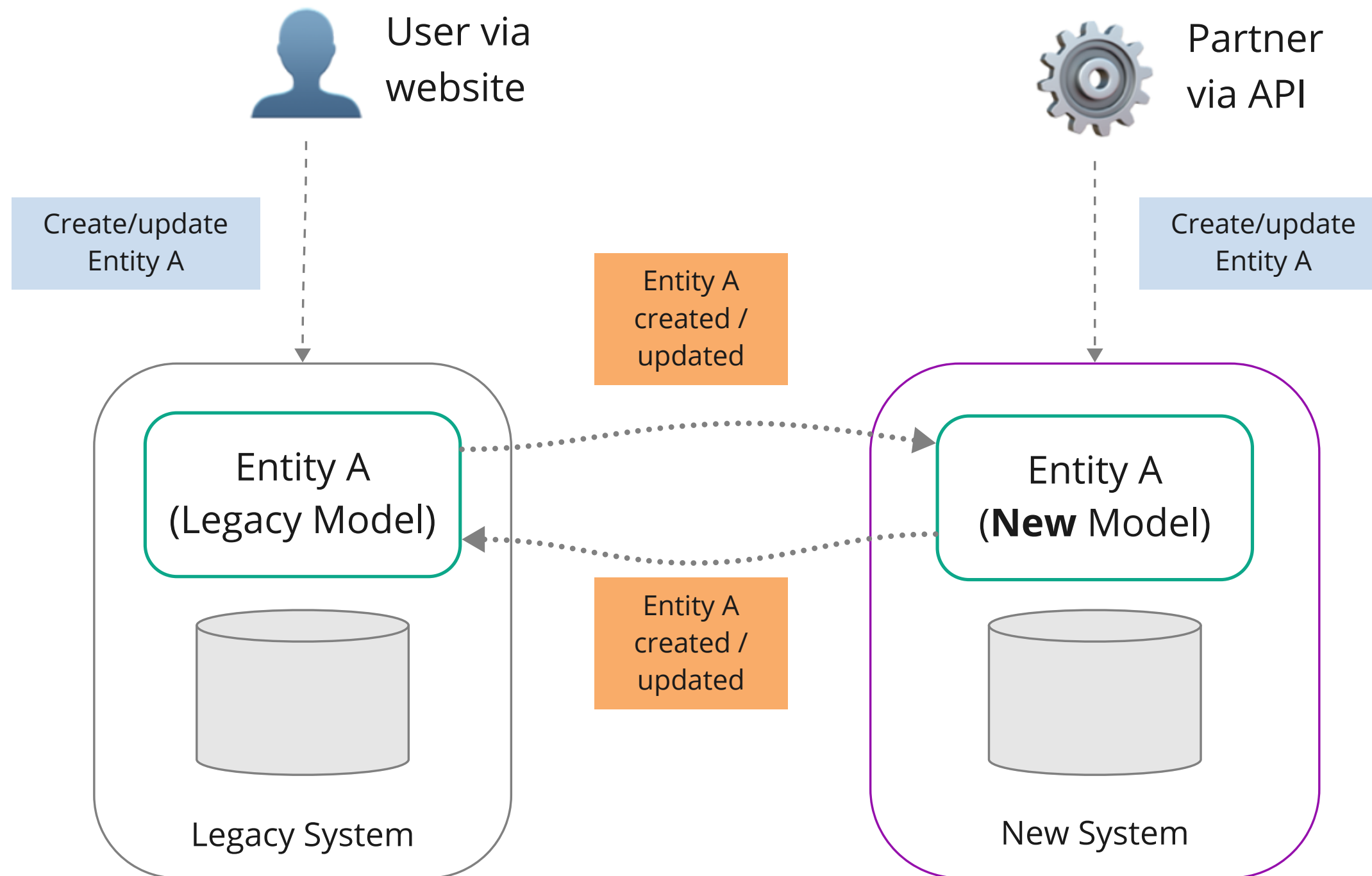
# Migrate Writes First (within a specific scope)



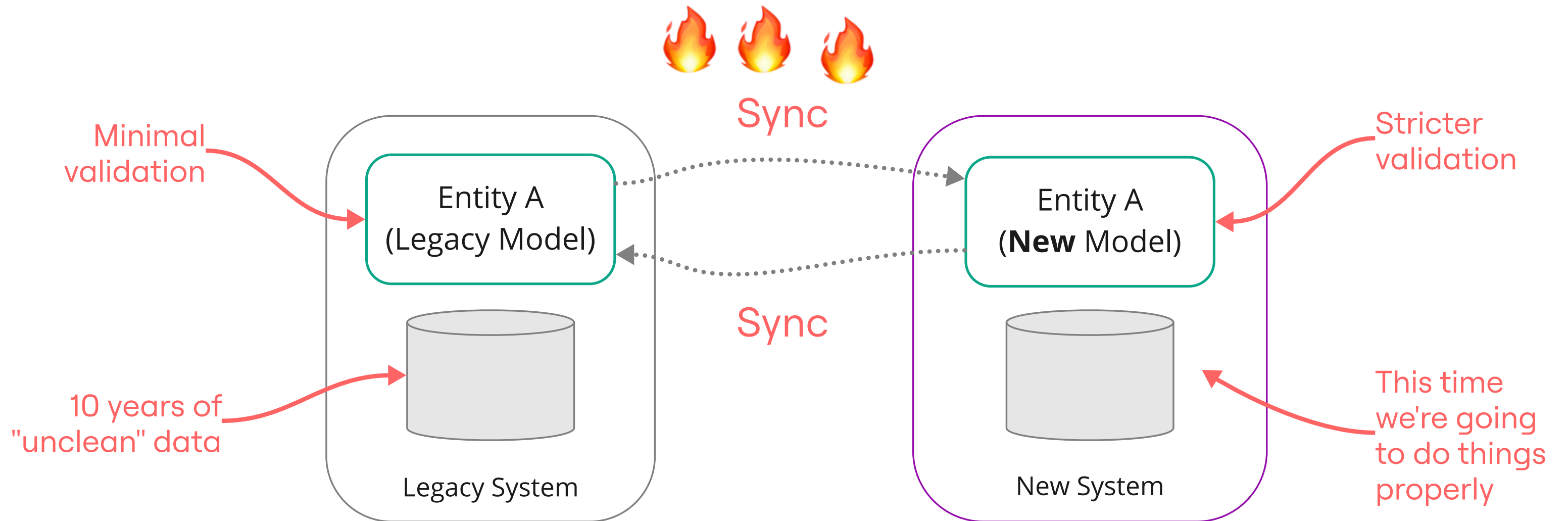
# Migrate Reads First (within a specific scope)



# Bi-directional model sync (risky)



# Asymmetrical Validation



**3.**

**PayFit Tech  
Transformation  
Context**

# PayFit: HR & Payroll

PayFit

working

Retour à mes collaborateurs

Rechercher un autre collaborateur...

Rechercher ...

Les essentiels

Tableau de bord

Mes collaborateurs

Demandes à valider4

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

NT

Nick Tune

Contrat (1)

Contrat en cours de création

Pas de date

En cours

1

2

3

4

5

6

7

Type de contrat

Poste

Lieu du poste

Durée du contrat

Cotisations et affiliations

Intégration

Récapitulatif

Type de contrat

Statut \*

Salarié

Nature du contrat \*

CDI

CDD

Dispositif particulier \*

Aucun dispositif particulier

Régime particulier du salarié

# PayFit Tech Transformation



## Company Objective:

PayFit's tech architecture is a lever for healthy growth

Allow teams to be more autonomous and innovate faster

Scale smoothly as company grows

Improve developer experience + efficiency

Reduce reliability issues & lost time



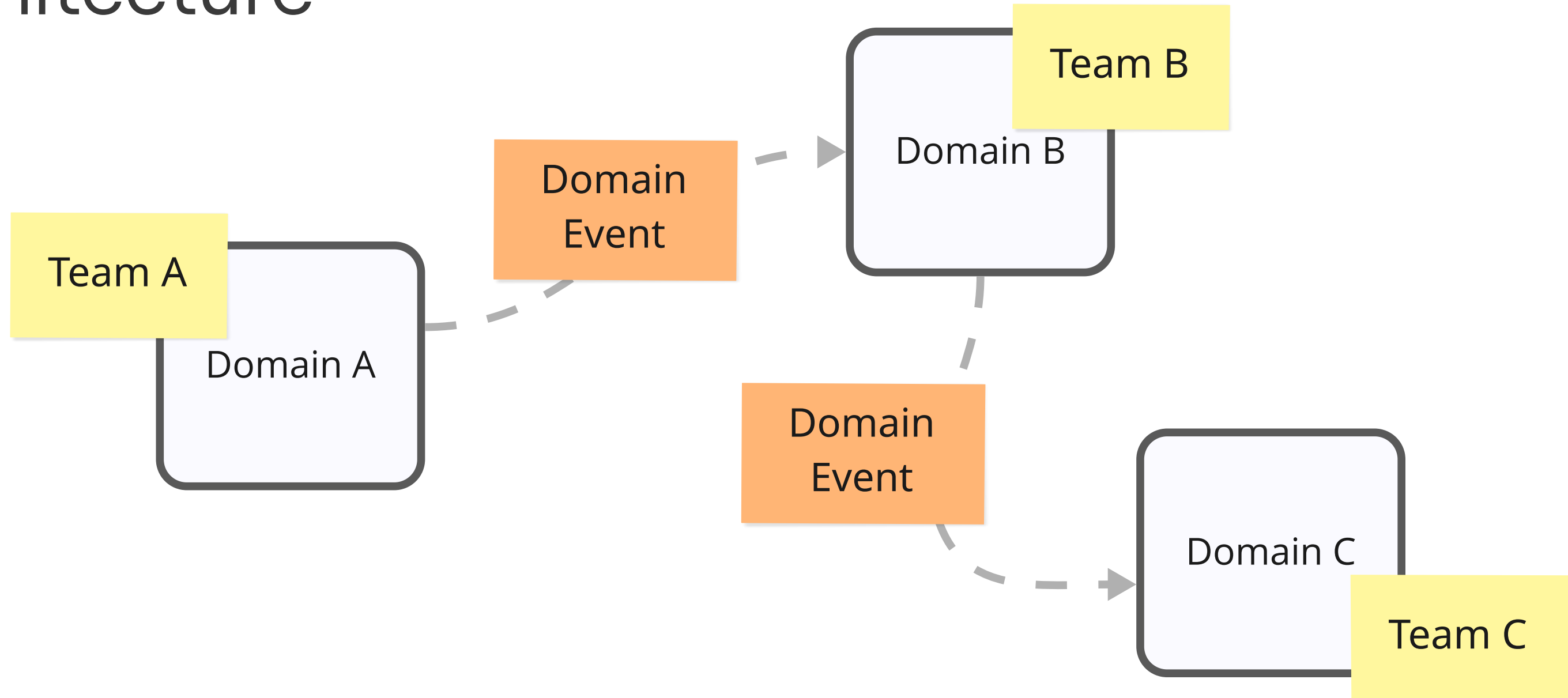
## Technological Shift:

Decoupling our technical architecture into domains & layers

Relying on **low-code** when necessary, built on top of our standard stack.



# PayFit Long-term Architecture Vision (LTAV): Domain-Driven Design & Event-Driven Architecture

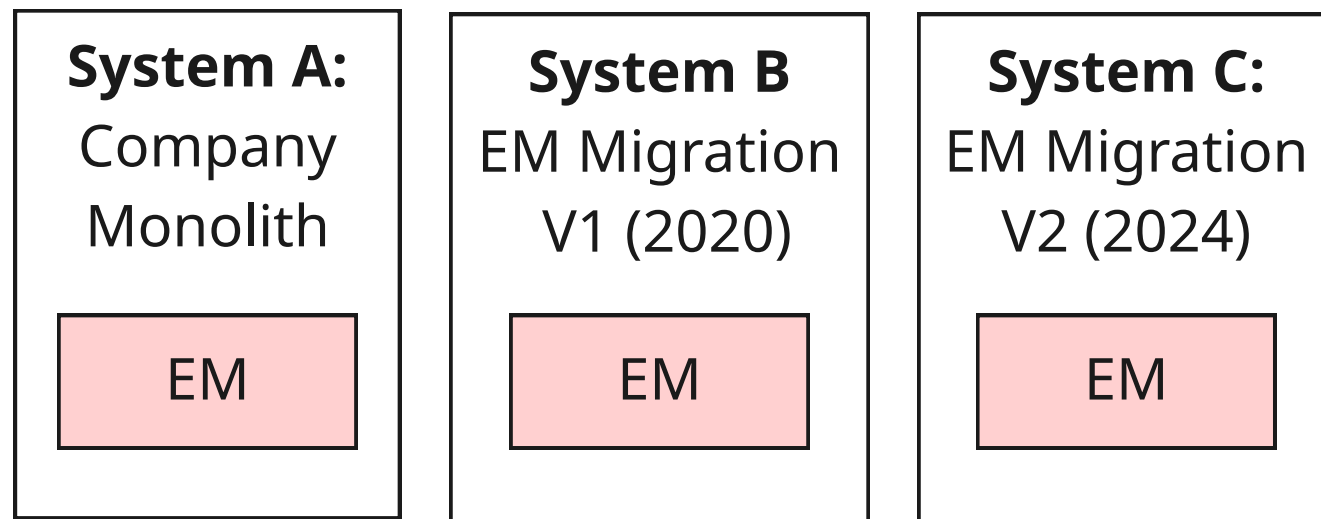


# Employee Management Domain: LTAV



## Current State

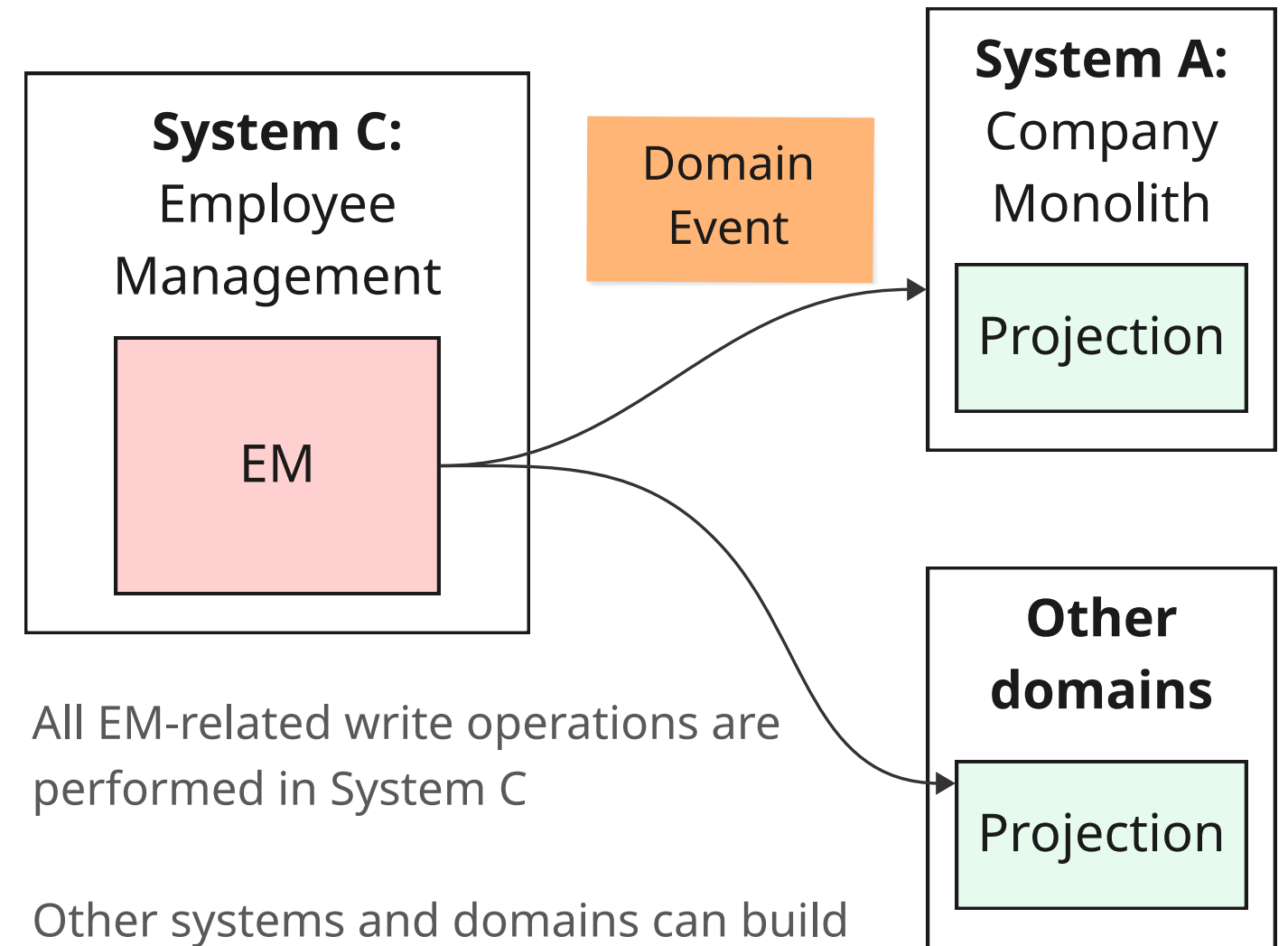
3 systems, 3 models,  
3 countries



- Read and write operations happen in all 3 systems.
- All 3 need to be synchronised in various ways
- Complexity spreads to other domains as well

## Target State

1 model, various projections



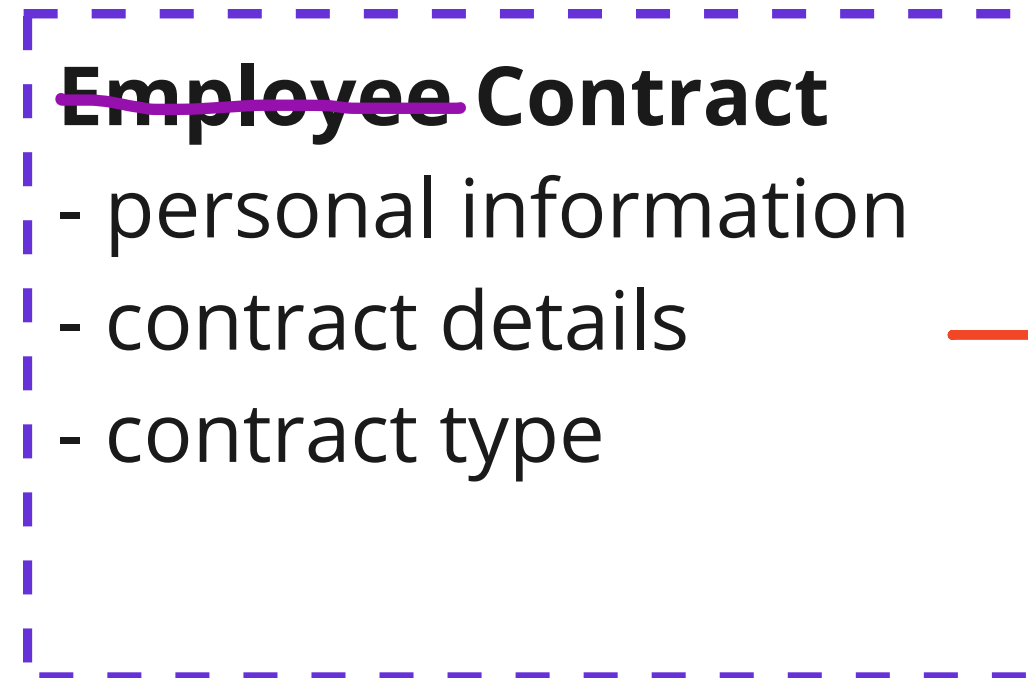
- All EM-related write operations are performed in System C
- Other systems and domains can build local projections based on events
- System B & C merged

# Competing\* migration priorities

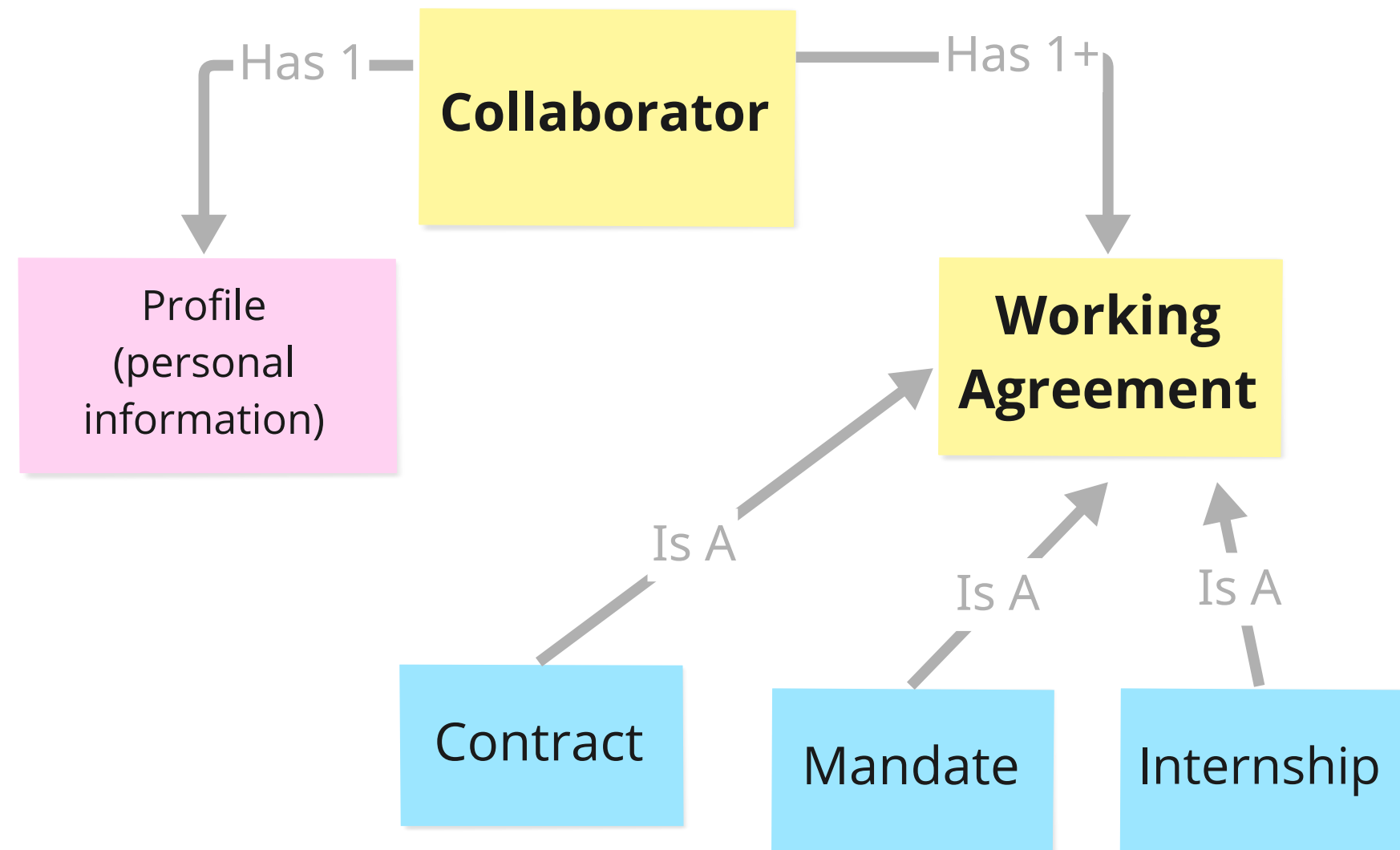
1. Other teams are waiting for our EDA events so they can fully migrate  
(events first ?)
2. New search APIs needed  
(read model first ?)
3. Build new features, reduce bugs, innovate faster  
(write model first ?)

# Drifting domain model - example

Model implemented in system



Target Domain Model



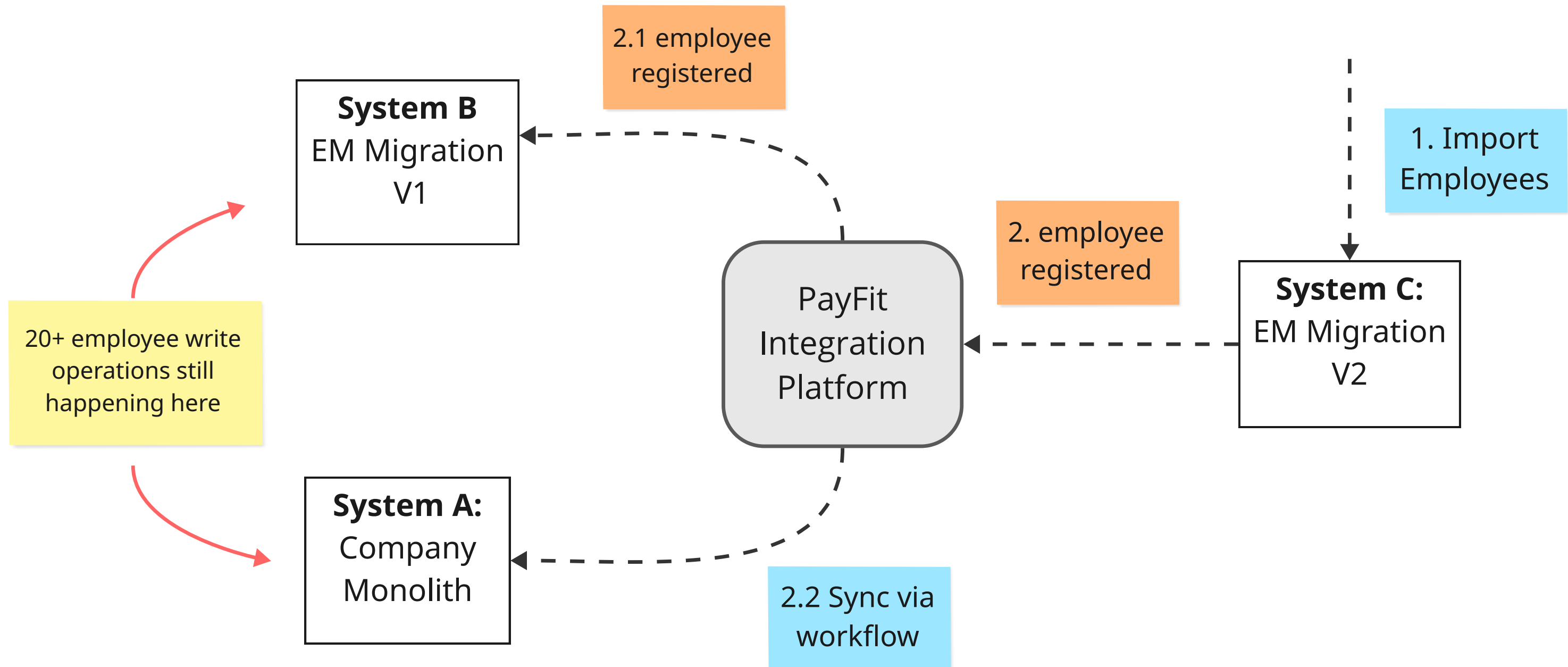
# Case Study 1:

## Migrating Employee Entity -- Write-model first

# How we choose the migration approach?

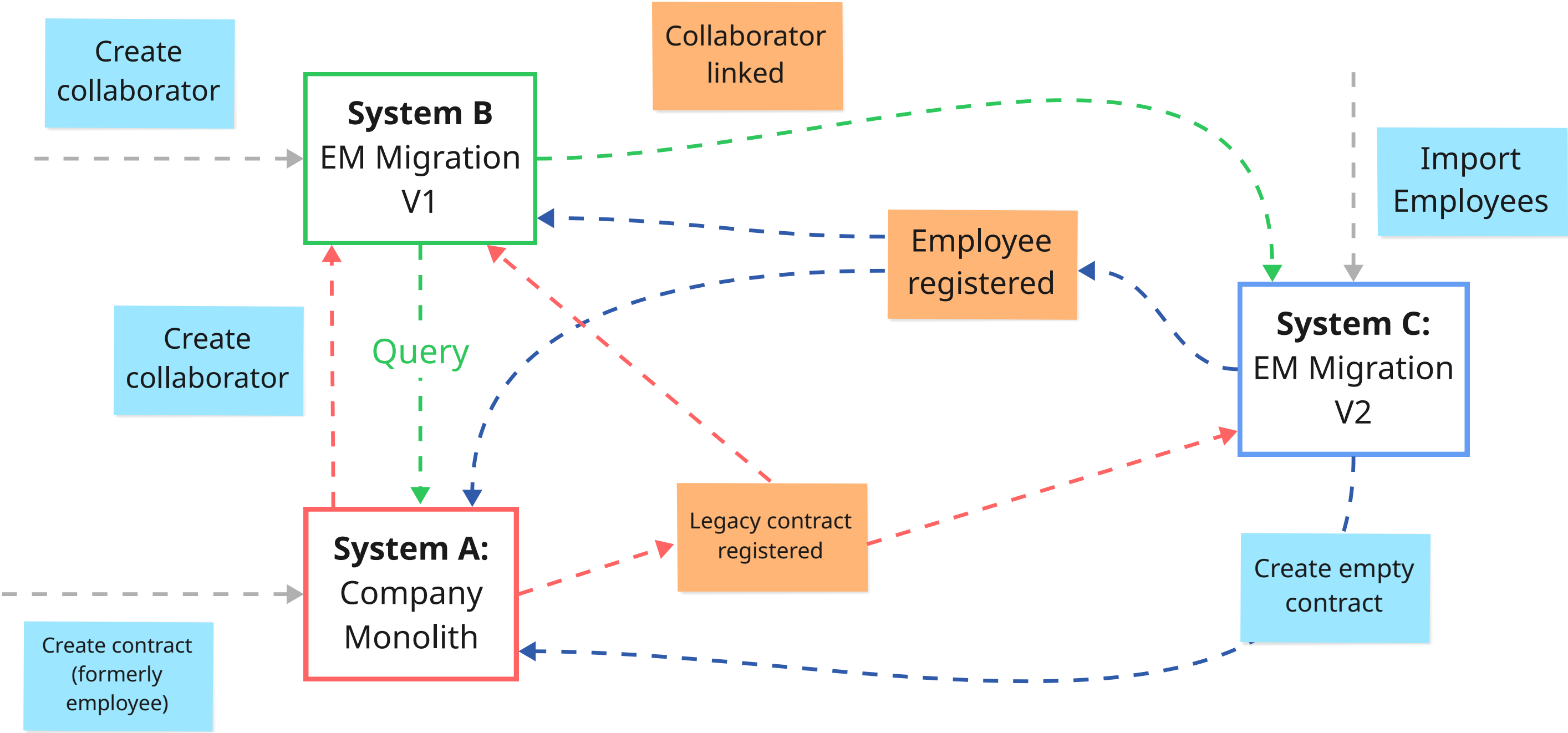
1. EventStorming workshops
2. Discussions with other teams (including platform)
3. Options & scorecoards
4. RFC, ADR, transparent decision making

# Migration decision: incrementally migrate writes





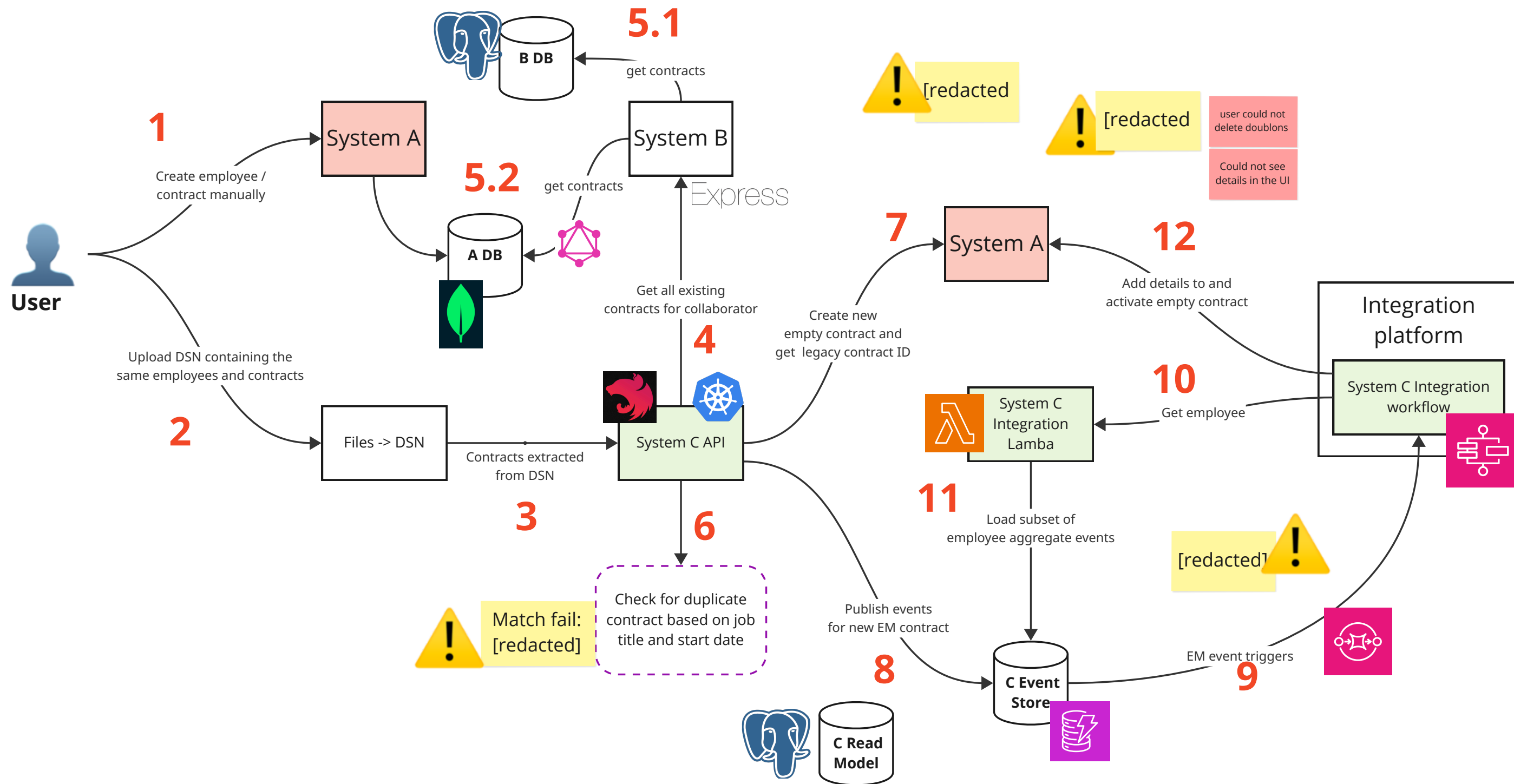
# Incremental = tri-directional sync





# tri-directional sync: detailed example

## Why do we have duplicate employees that are stuck in a broken, half-empty state and cannot be deleted by the user?



# How did we end up in this situation? Observability to the rescue

Search events in the dev portal  
(EDA + event sourcing evens)

| Dev <u>Staging</u> Producti                     |                     |
|-------------------------------------------------|---------------------|
| TYPE                                            | DOMAIN              |
| private.employee-management.employee.removed    | EMPLOYEE_MANAGEMENT |
| private.employee-management.employee.registered | EMPLOYEE_MANAGEMENT |
| private.employee-management.employee.ready      | EMPLOYEE_MANAGEMENT |

Rich, structured pre-DLQ logs in  
DataDog

```
{
  errorMessage: "Failed to
apply event
contract.registered...",

  failedMessage: { ... },

  exception: { .. }
}
```

failedMessage = the message put on  
the DLQ

exception = reason the message could  
not be processed

Advanced search in  
snowflake

```
set companyId = 'REDACTED';
select distinct *
from [redacted] e
where
event_date > '2024-05-01' and event_date <
and
(
    subject_id = $companyId or
    payload like '%' || $companyId || '%'
)
ORDER BY EVENT_DATE desc, subject_id asc
```

# AI Debug Agent

```
# Debug Agent

You are a member of the employee management team, responsible for managing systems related to employees and their contracts.

Your role is to help the team investigate, diagnose, and resolve support tickets and bugs.

You think carefully, look to establish facts, and collaborate with the user. You do not rush off into solutions based on assumptions.

....
```

Uses

CLI comands  
& MCP

query  
dataDog

query APIs

query  
Event Store

query DBs

JIRA

Can access

Every relevant  
codebase

## Summary – overall happy with the decision 👍



Feels great to take ownership of the domain model and continuously evolve it



Nice to build new features with the new model



Synchronisations need to be built and maintained

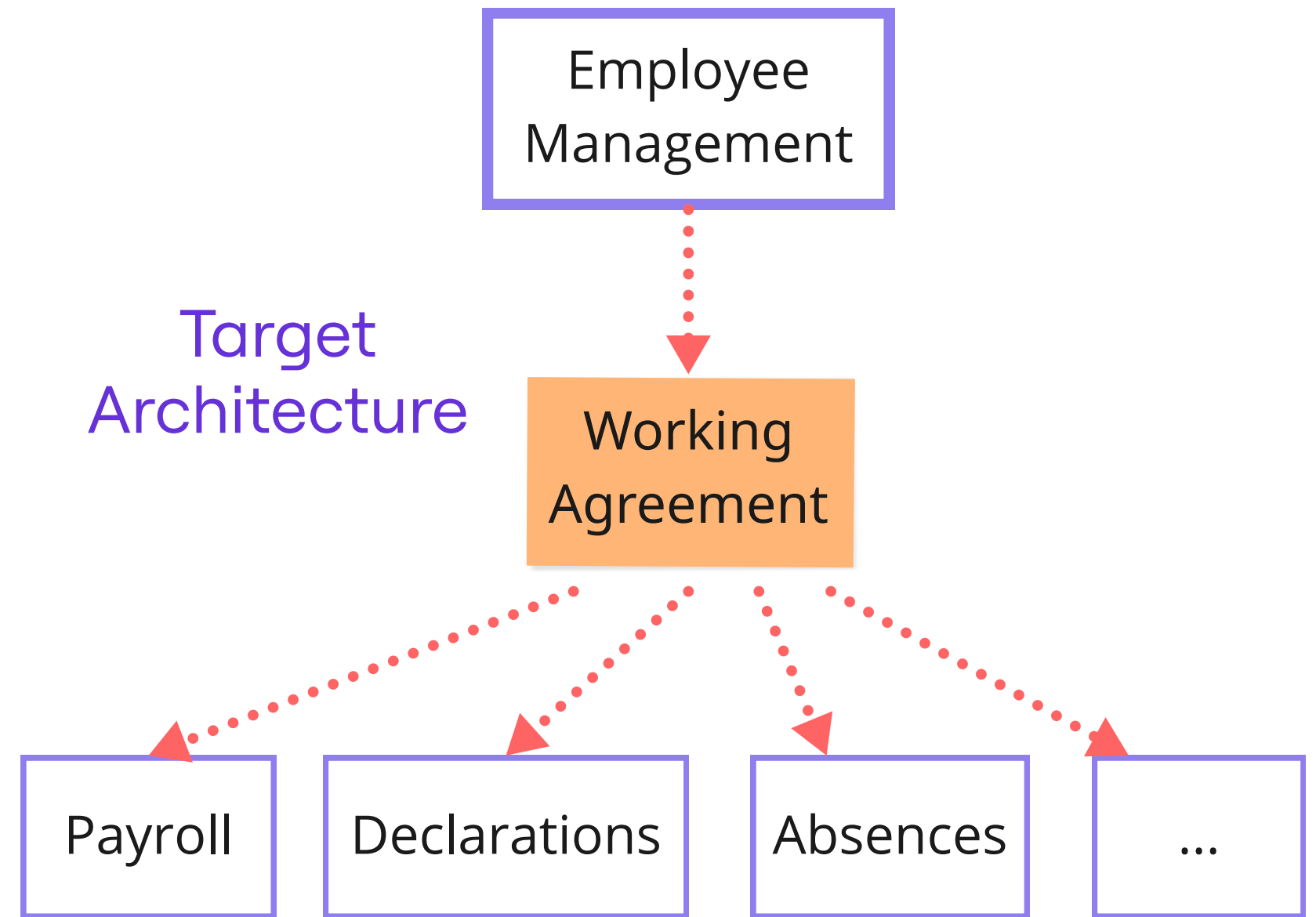


Cost of investigating and resolving sync issues can be high sometimes

# Case Study 2: Migrating Contract Entity -- Legacy Event Republishing

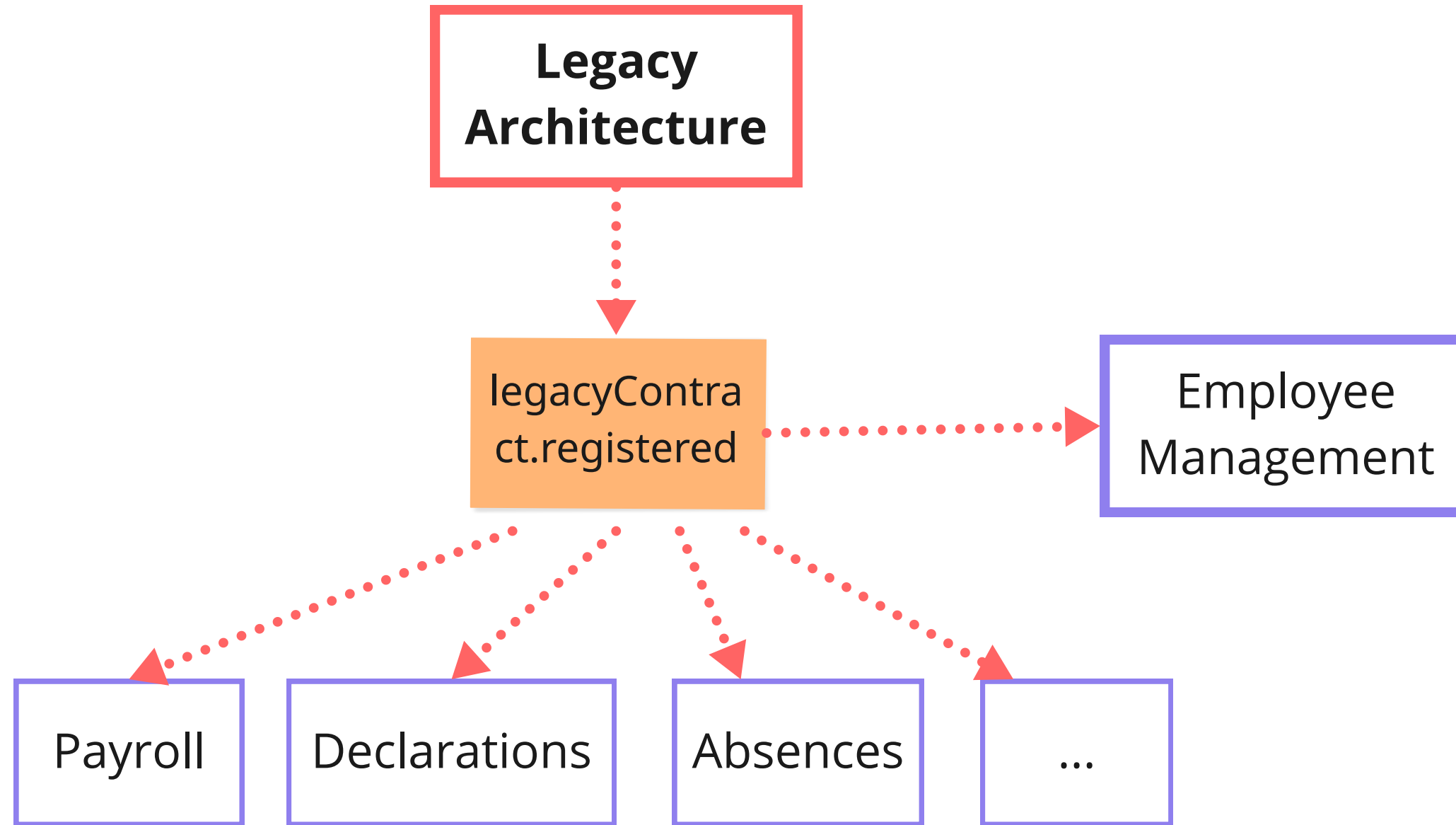
# Challenge: EM Team is the bottleneck for other teams that want to transform

- Employee Management is upstream of many other domains who want to migrate away from legacy.
- They need to know about the lifecycle of contracts from the EM domain.
- But the EM team is focused on transforming the Employee/Collaborator first



# Option: Take events from legacy model

Proposed  
migration step  
architecture



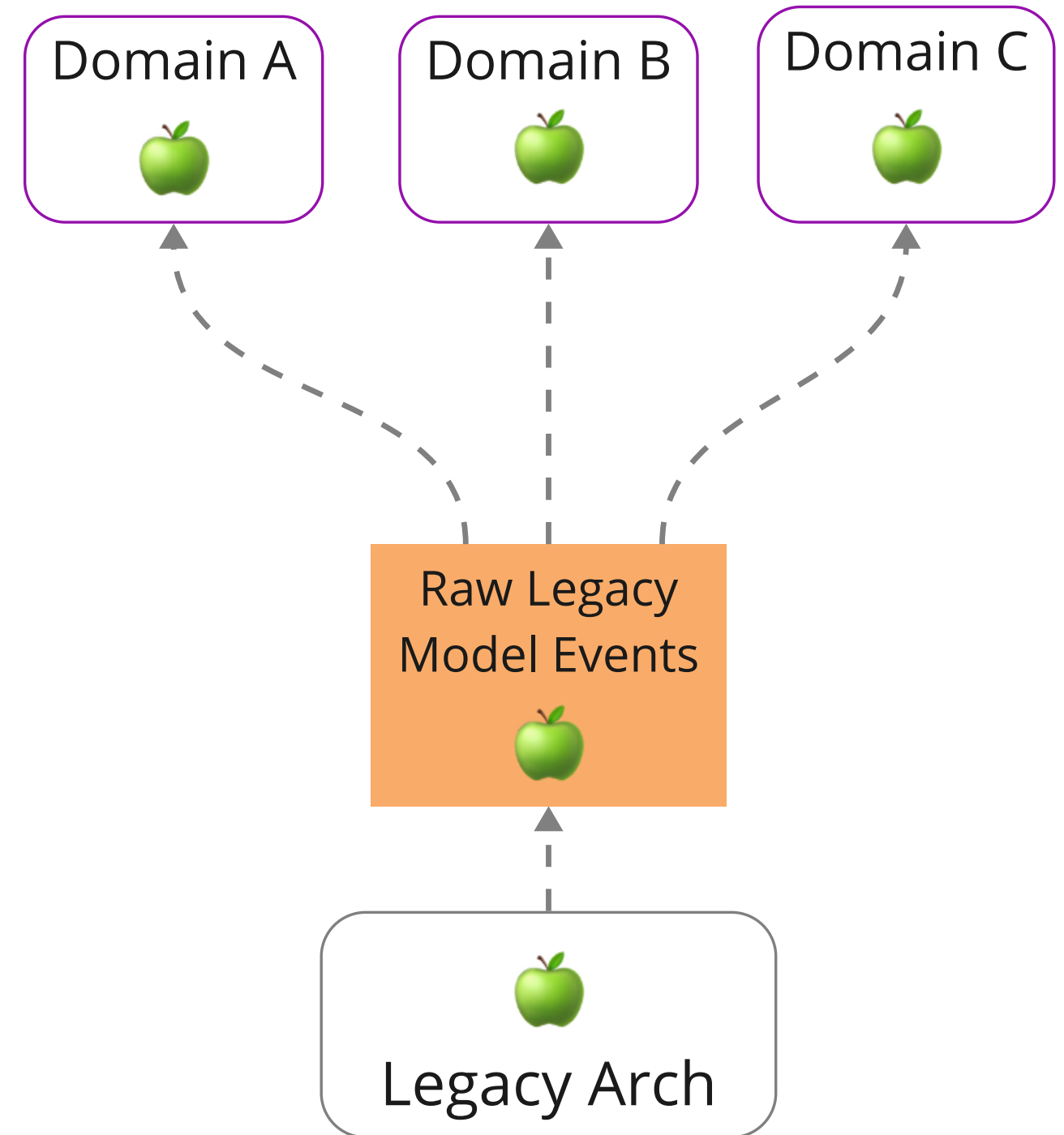


# LTAV Principle:

Domains cannot consume other domain's raw legacy events

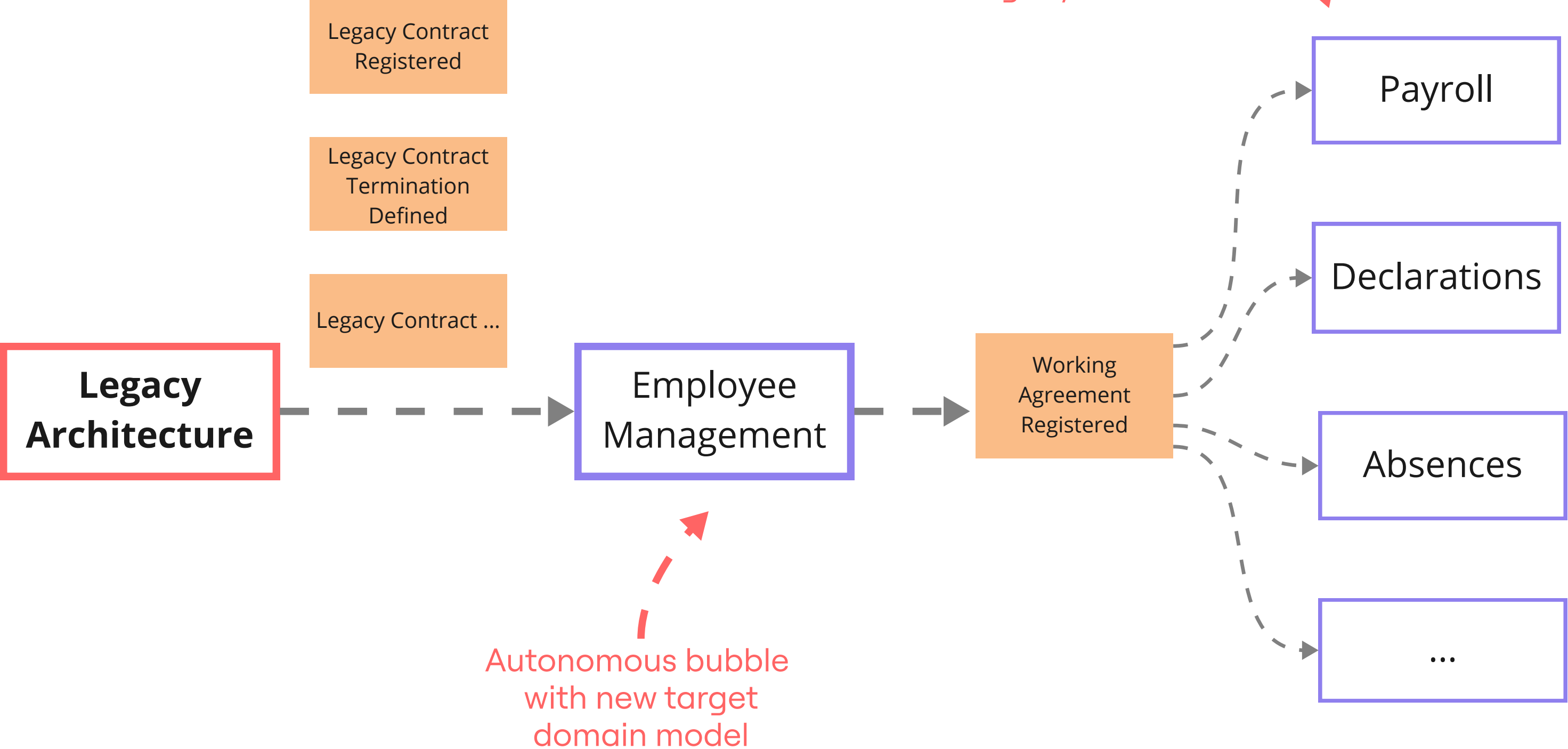
- A domain should be the source of truth for it's data
- Avoid duplicating model translation and business rules
- Avoid creating extra coupling and creating more migration work in the future
- Remove dependency on legacy IDs
- Minimize consistency issues

 = legacy model (leaking everywhere)

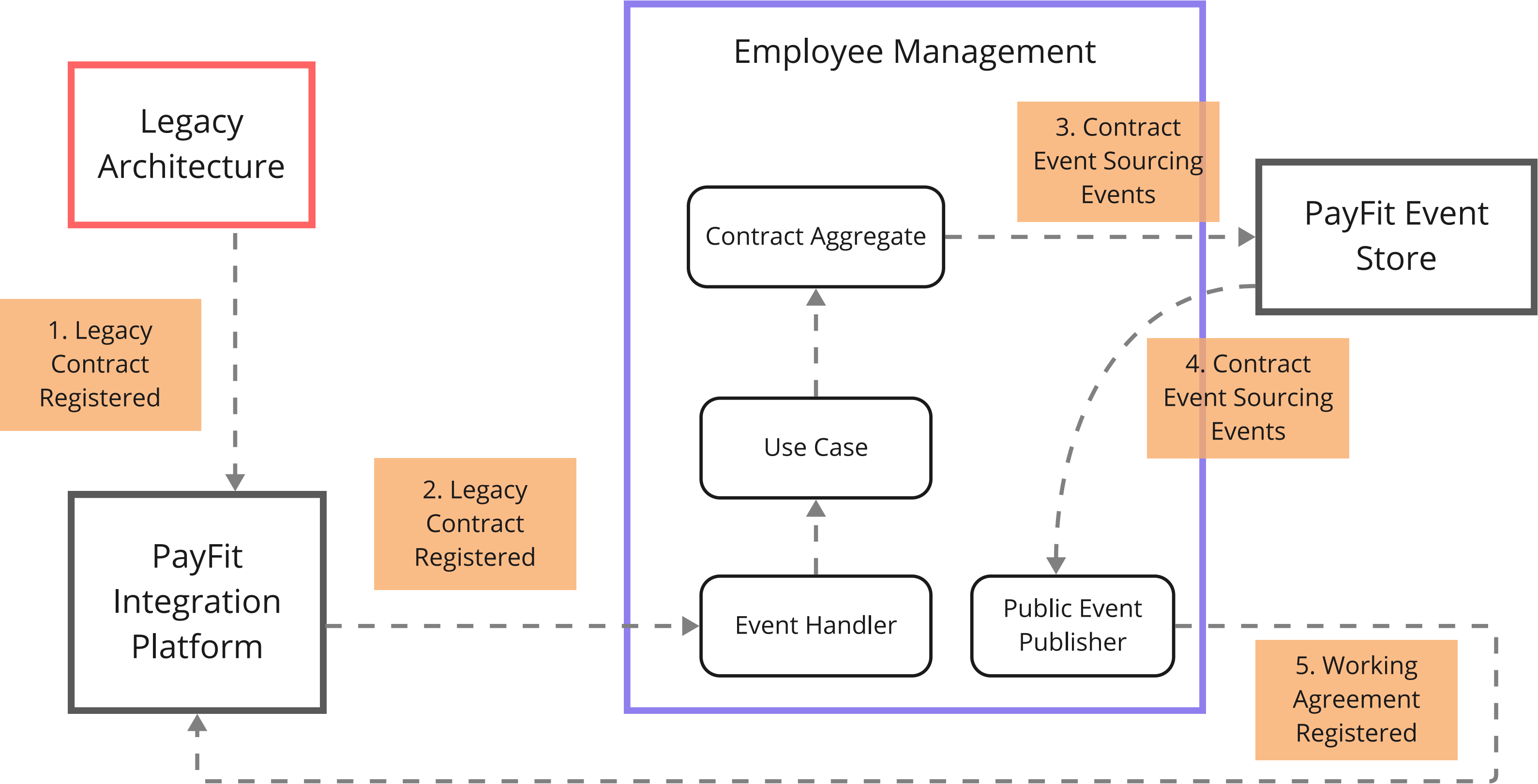




# Decision: Republish legacy events



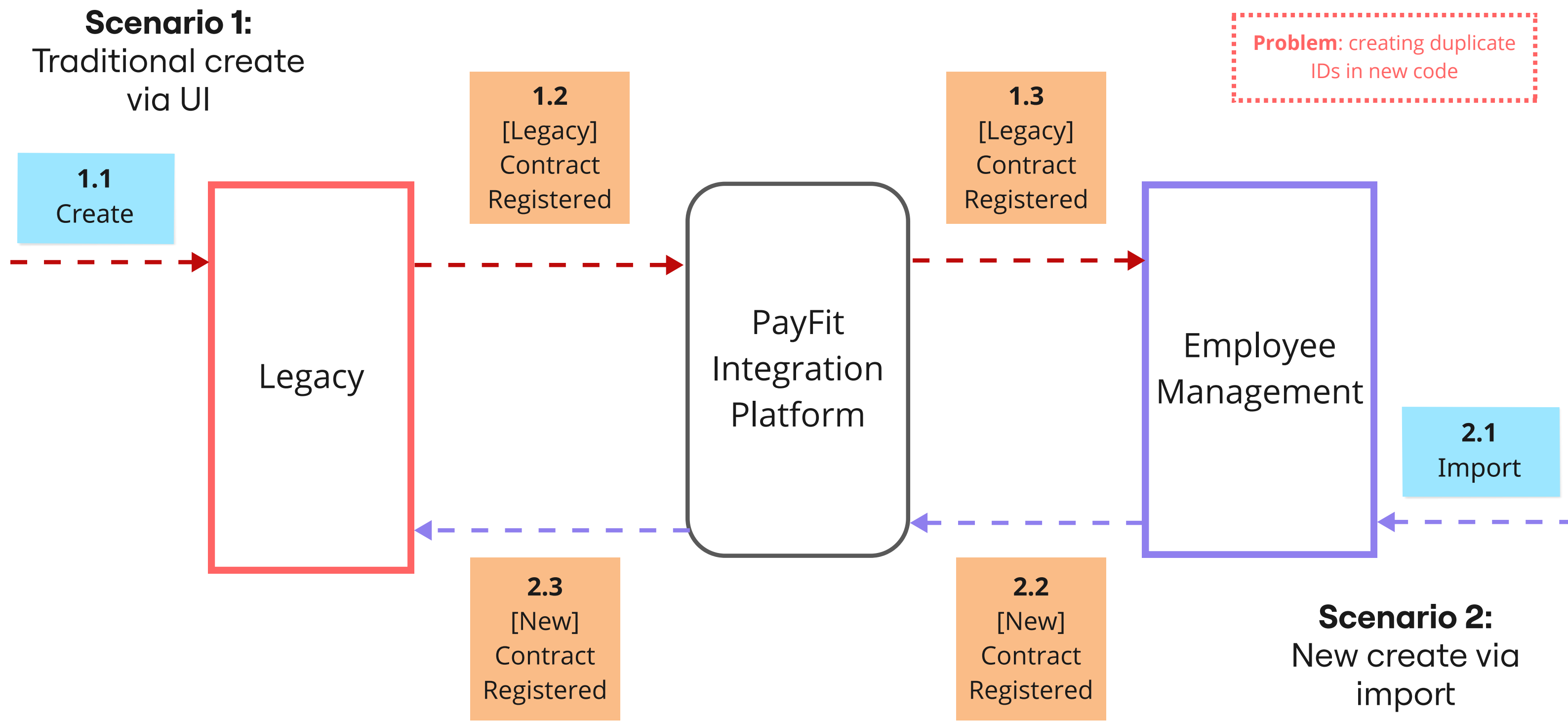
# Republishing Implementation



# **Challenge:** Events from legacy can be unreliable

- If the legacy DB is modified directly no events will be published
- Some legacy flows might not publish events
- Duplicate events can be produced (e.g contract onboarding)
- Events can be produced in different order depending on the flow the user takes
- Each country has nuances in types of events and what data means (e.g. isActive status code)
- Validation in the new model is stricter than legacy – how to handle data that was created in legacy with less strict rules?

# Evolution: Migrated various write operations



## Summary – overall happy with the decision 👍



We have unlocked other domains



Evolved into a full-blown bubble context that now manages key parts of the lifecycle



It's far more effort than simply republishing events out of the legacy. Model mapping constraints.



Building on less-reliable foundations that behave differently to your new system

**4.**

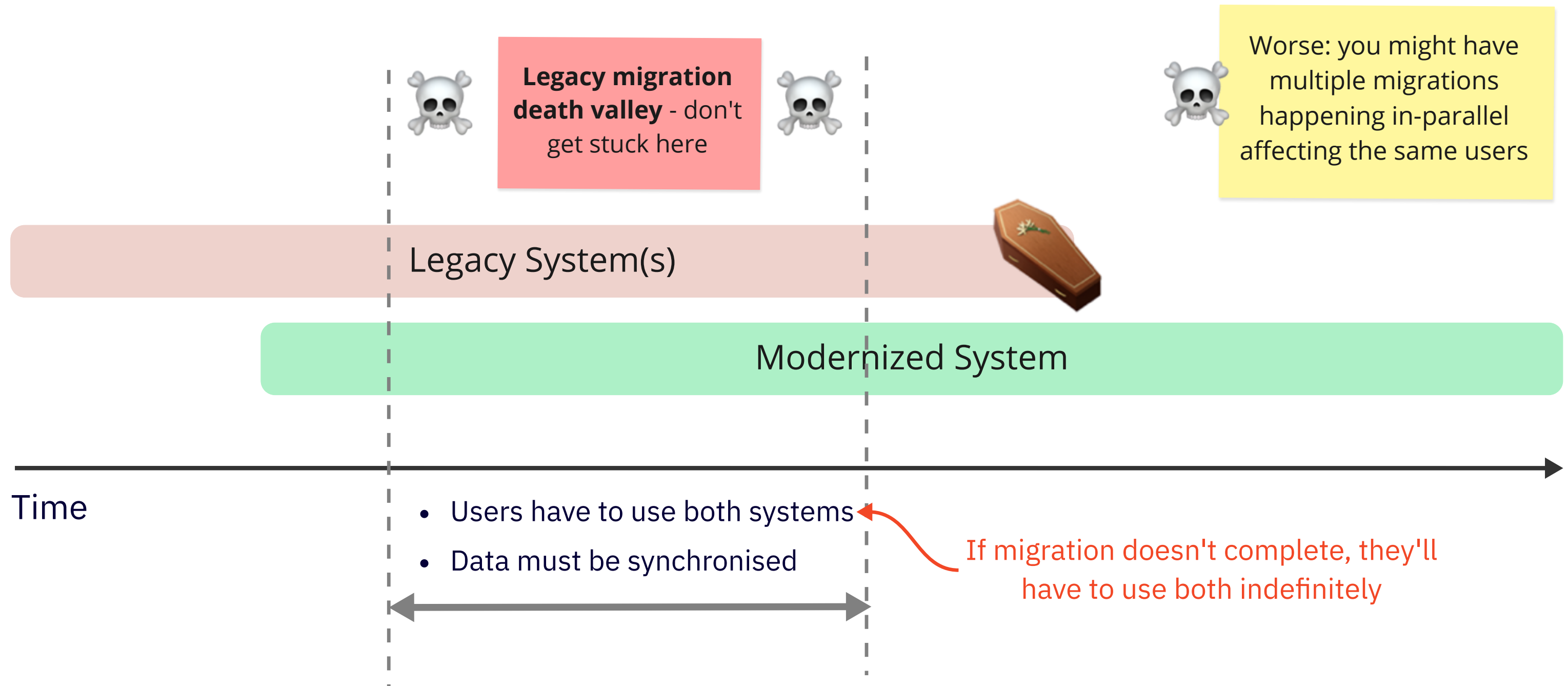
**Final Word**

Everybody talks about incremental modernization as the holy grail.

It's usually better than a big-bang rewrite – but don't fool yourself thinking it's going to be easy...



# Before you start migrating, ensure you are not going to get stuck in legacy migration death valley?





Think about all of the different ways your system could break as you have new and old running in parallel for multiple years.

Then choose your legacy migration patterns wisely



[careers.payfit.com](https://careers.payfit.com)

[backstage.payfit.com](https://backstage.payfit.com)