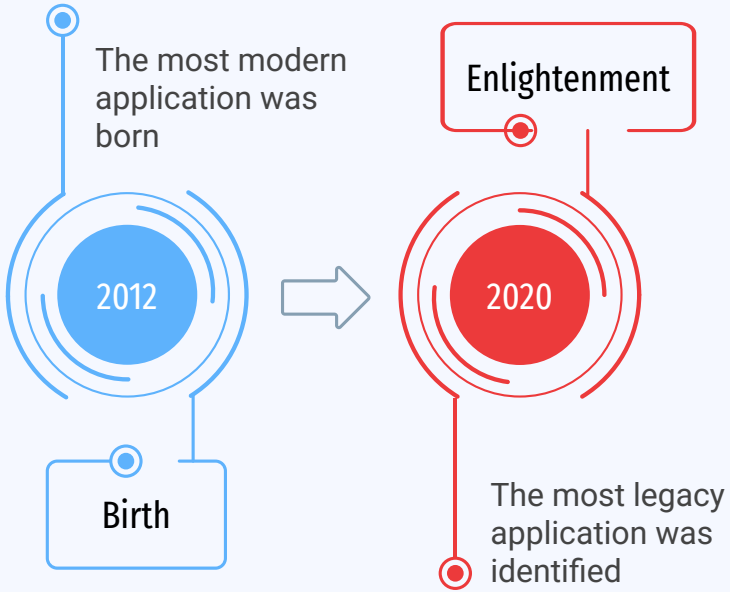
Modernizing the Legacy

From Monolith to Modulith

Aggelos Bellos

- Architecture Team @ Epignosis
- Co-Organiser of Devstaff.gr
- Blogging about architecture.
- Hooked on Socio-Technical Systems

TalentLMS Timeline



Prehistoric state

01

DevOps Silos

How can you scale an app
if you don't know its needs?

02

Monolithic dev team

Any ideas of how long a
10+ dev team daily lasts?

03

Legacy code

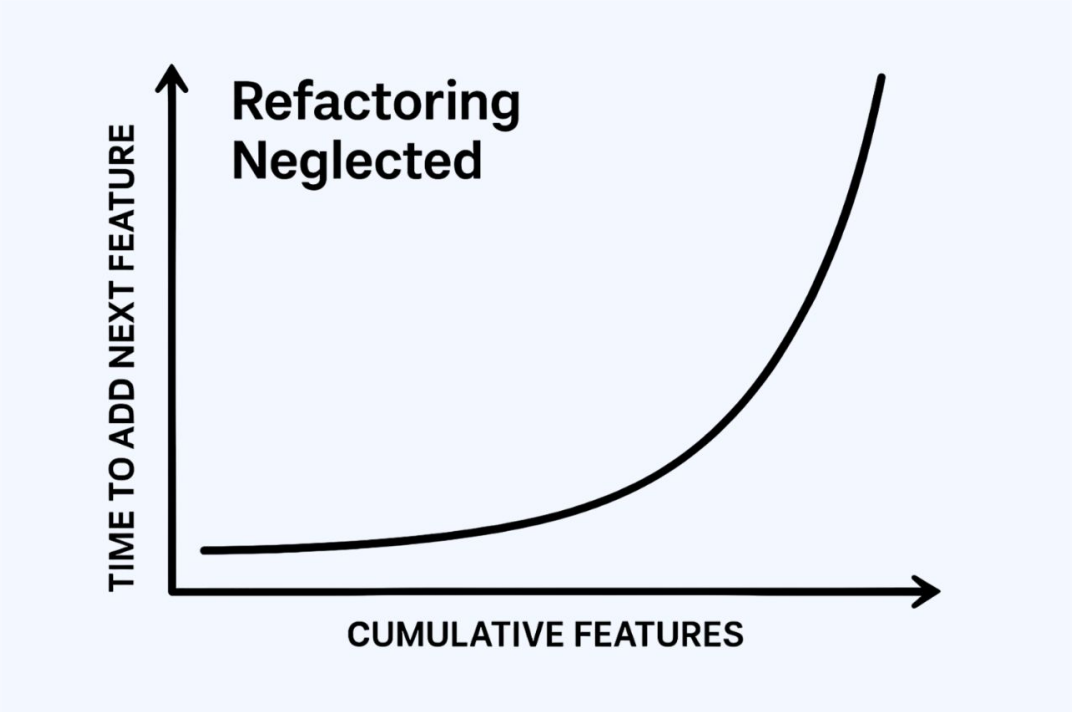
Wait.. keeping adding
features is not a good thing?

404

Documentation not found

Why you want me to write
it? You can just ask me!

Over exaggerated graph



01

DevOps Silos

How can you scale an app if you don’t know its needs?

DevOps Silos = No communication

Environments

Environments were hard-coded inside the code.

Stateful App

Application was tightly coupled with the infrastructure.

Replication

Each environment was built by hand which made replication difficult.

Make infrastructure great again

Decoupled app from storage

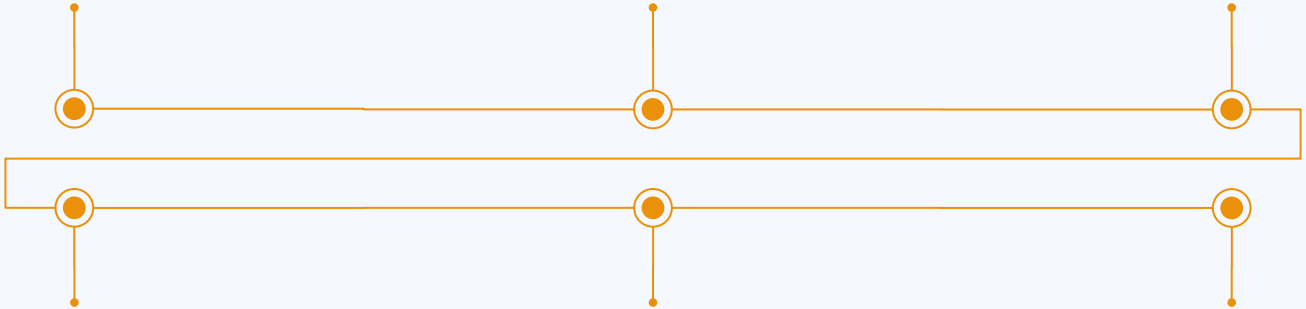
First

Removed sticky sessions

Next

Documented Infrastructure

Next



Next

Cleaned-up environments

Next

Infrastructure as code

Last

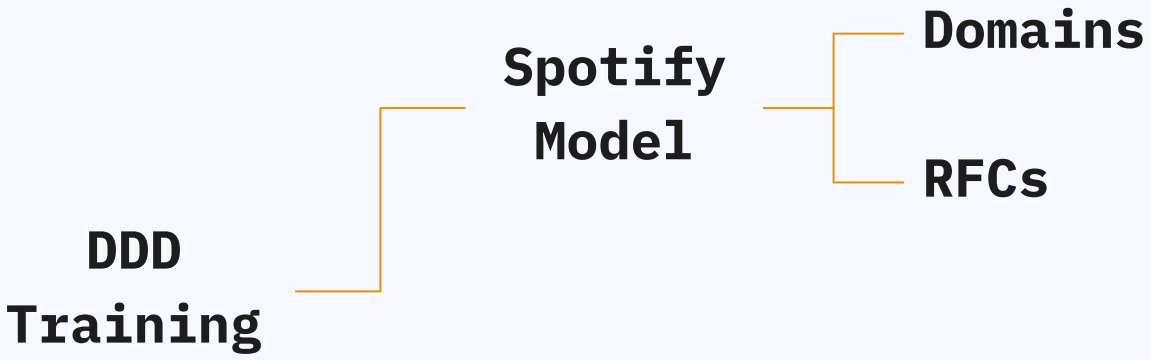
Replaced IPs with DNS records

02

Monolithic Teams

Any ideas of how long a 10+ dev team daily lasts?

It's morphing time



Bounded-contexts for the win

Dictate architecture and knowledge sharing

New structure

Squad A

Courses

- +1 DevOps Engineer
- +1 QA Engineer

Squad B

Business

- +1 DevOps Engineer
- +1 QA Engineer

Squad C

Integrations

- +1 DevOps Engineer
- +1 QA Engineer

Squad D

Core

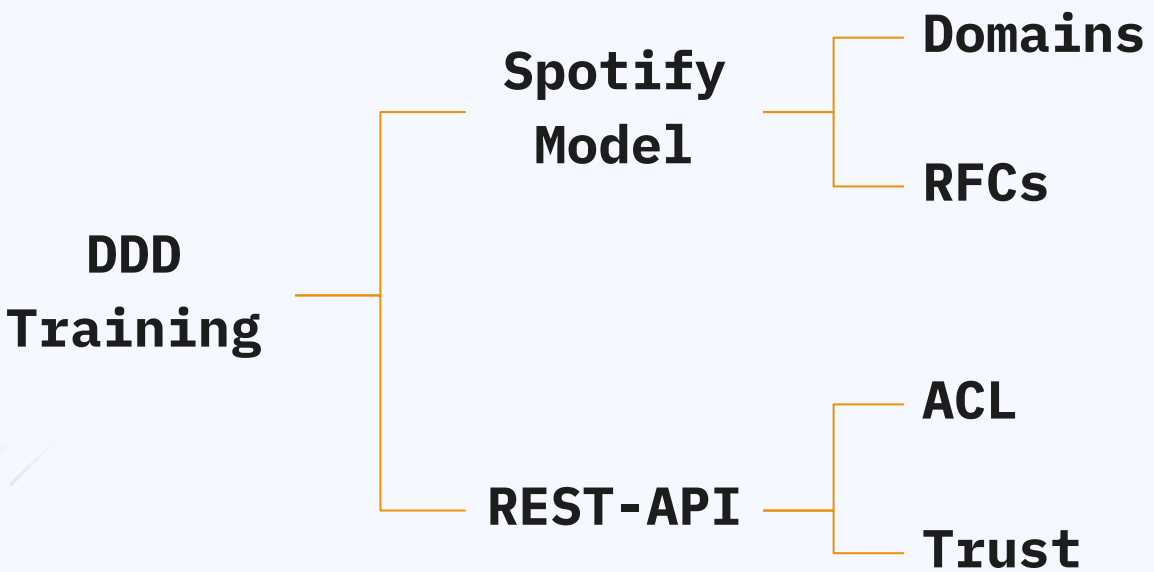
- +1 DevOps Engineer
- +1 QA Engineer

03

Legacy Code

Wait.. keeping adding features is not a good thing?

It's morphing time



Bounded-contexts for the win

Dictate architecture and knowledge sharing

Shiny code stays shiny

Build trust and confidence in the system itself

Foundations - Modern standards

Docker

Consistent, containerized environments

Static Analysis

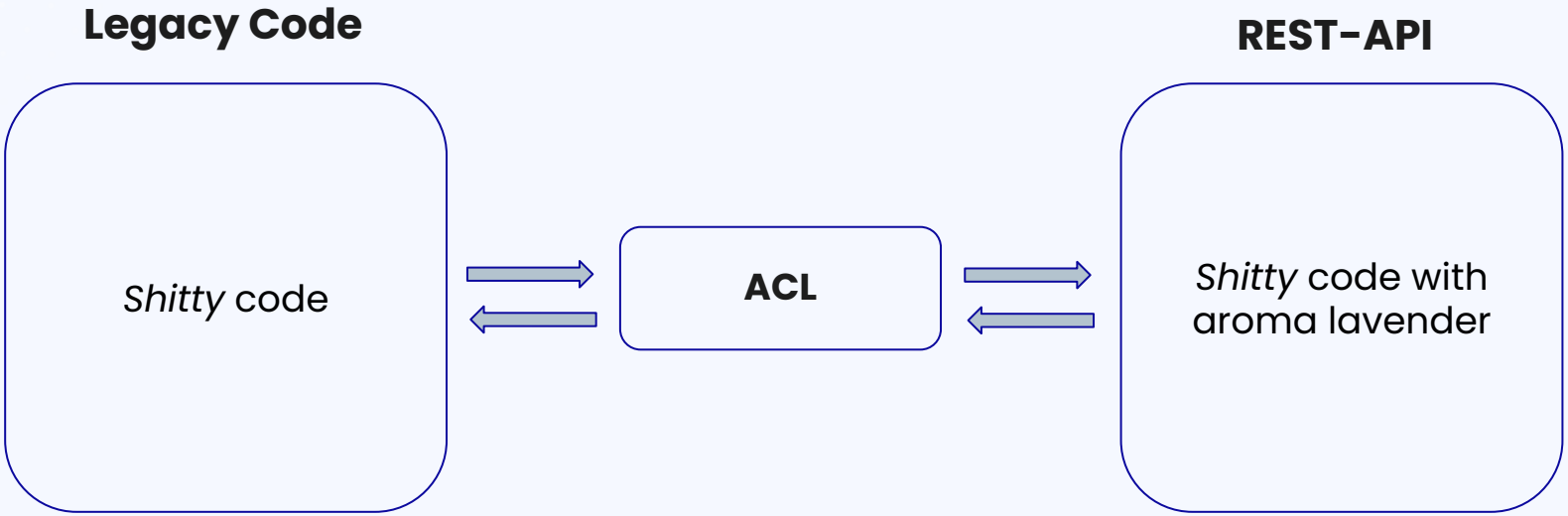
Automated code scans

Tests

Unit + integration suites act as a safety net

Composer

Dependency manager for PHP



- Low standards
- Tight coupling

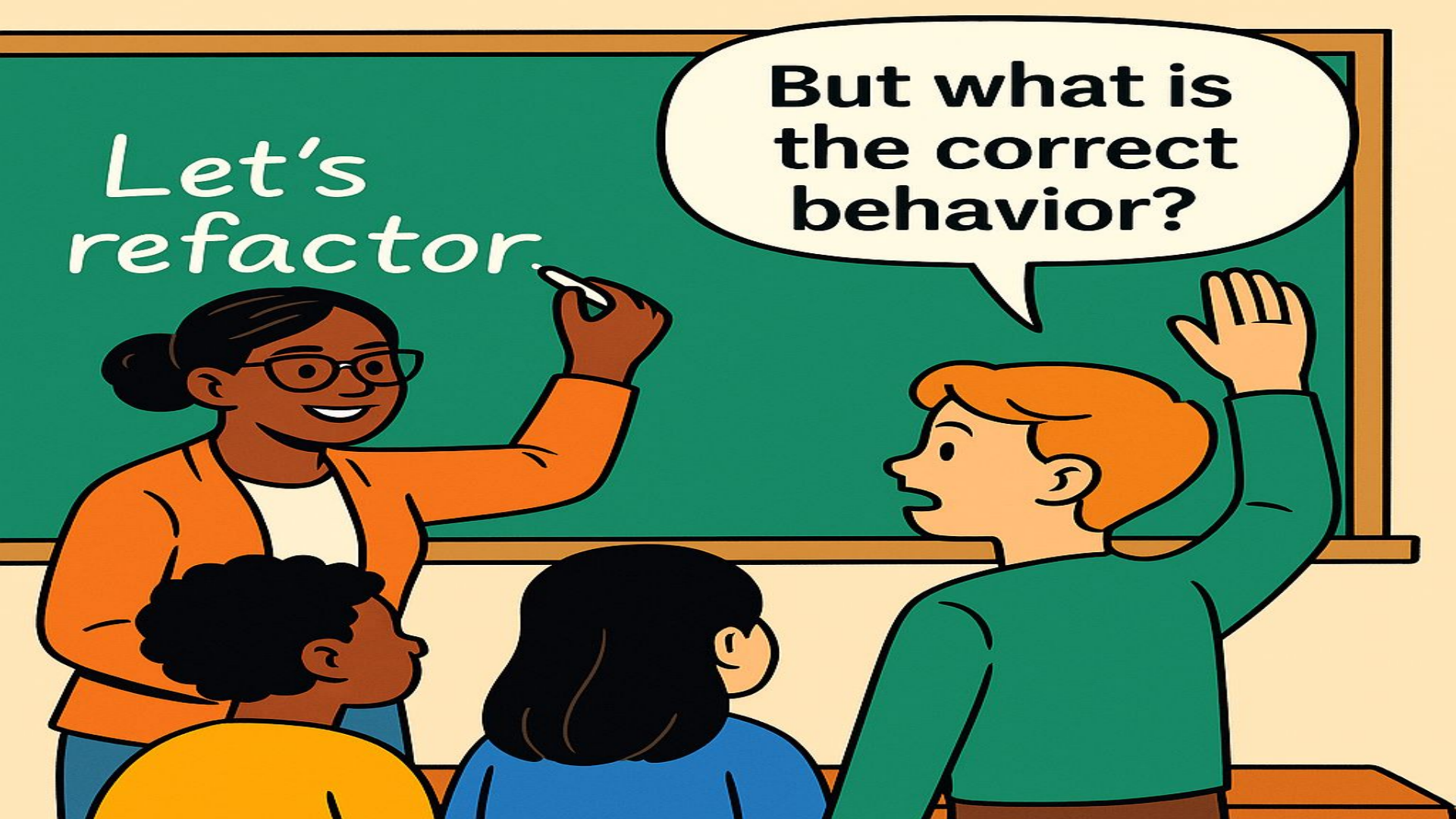
- High standards
- Modular design
- Tests
- Static analysis


REST-API
LEGACY
imgflip.com
Ashe 2016

404

Documentation not found

Why you want me to write it? You can just ask me!


Let's refactor.
But what is the correct behavior?

Documentation Pillars

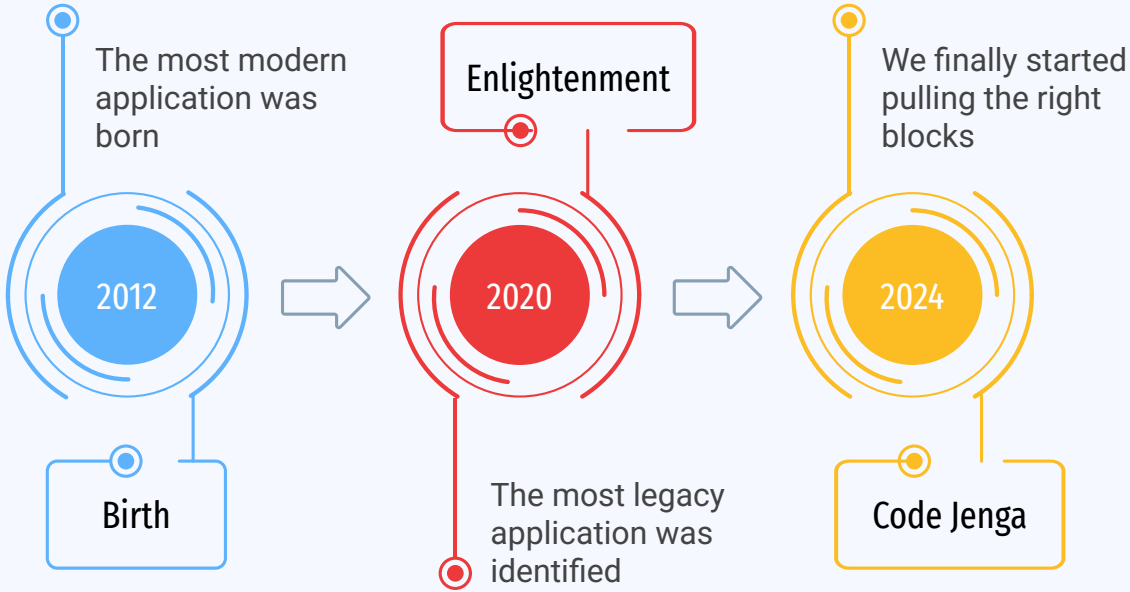
OpenAPI

Easily accessible endpoint catalog for every developer

BDDs

Developers’s source of truth

TalentLMS Timeline



Modernization Blueprint

01

Teams on Purpose

Team Topologies + AMET

02

Modulith Awakens

Event-driven & Public API

03

Turbocharge Ops

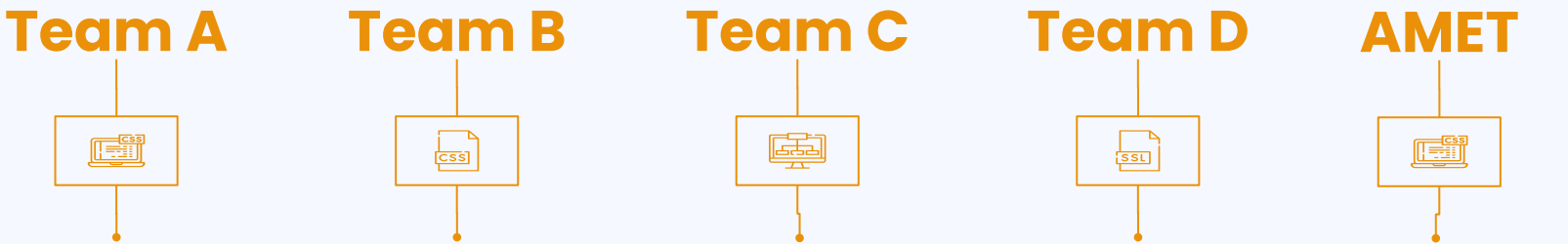
Aurora MySQL 8 and
automating deployments

04

Living Documentation

PRDs, ADR, C4 model and a
DDD-based folder structure

Team Topologies structure



Architecture Team

Event-driven Architecture Team

DevOps Platform

QA Platform

Modulith Awakens

Event-driven

Setting up the foundation for our new Event-driven journey.

Public API

Team and repository are matured enough to accept more projects.

Automating Deployments

IaC

Infrastructure as Code

Ephemerals

Short-lived production environment copies

Github Actions

Trigger deploys and builds on merge

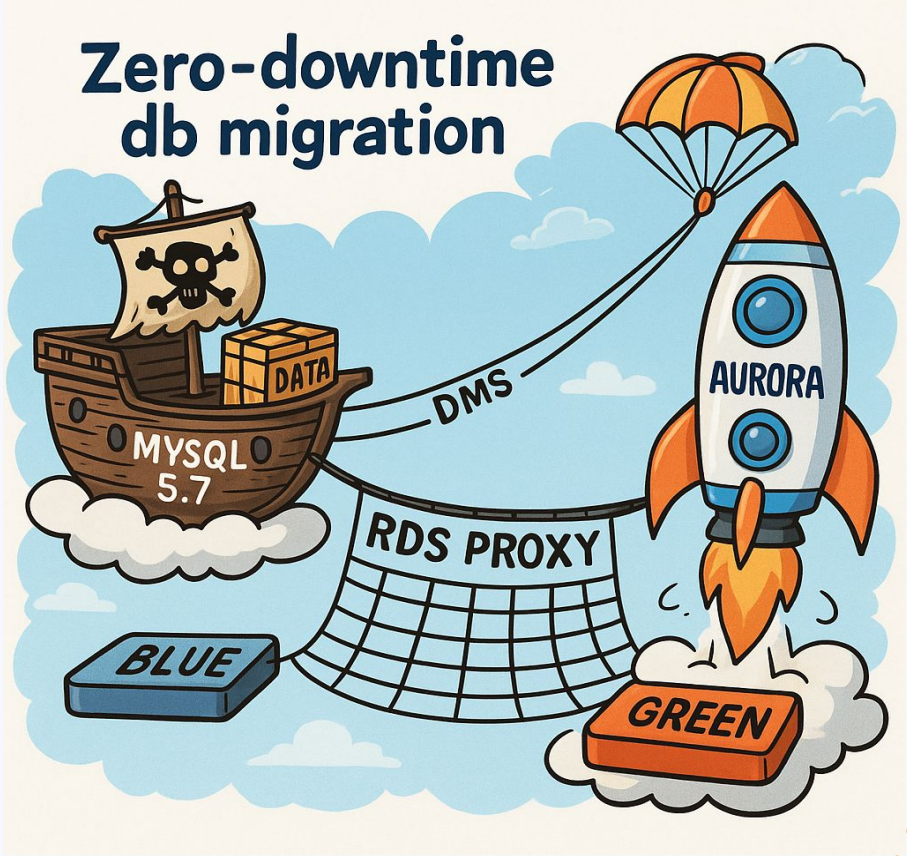
AWS CodeDeploy

Fire and forget artifact based deploys

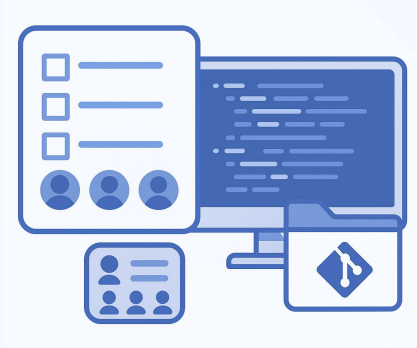
Zero-downtime db migration

From Mysql 5.7 to Aurora Mysql8

- New customers used the Mysql8
- Canary domains migrated to Mysql8 for testing
- Flipped the switch using DNS records

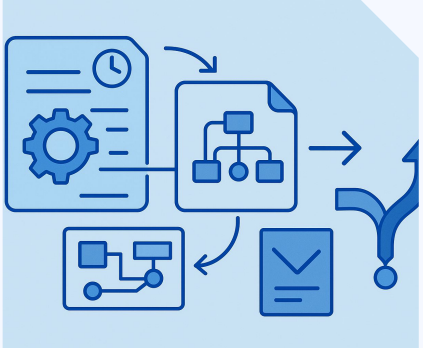


Living Documentation



PRDs

Lean, version-controlled specs that capture why and what right beside the code



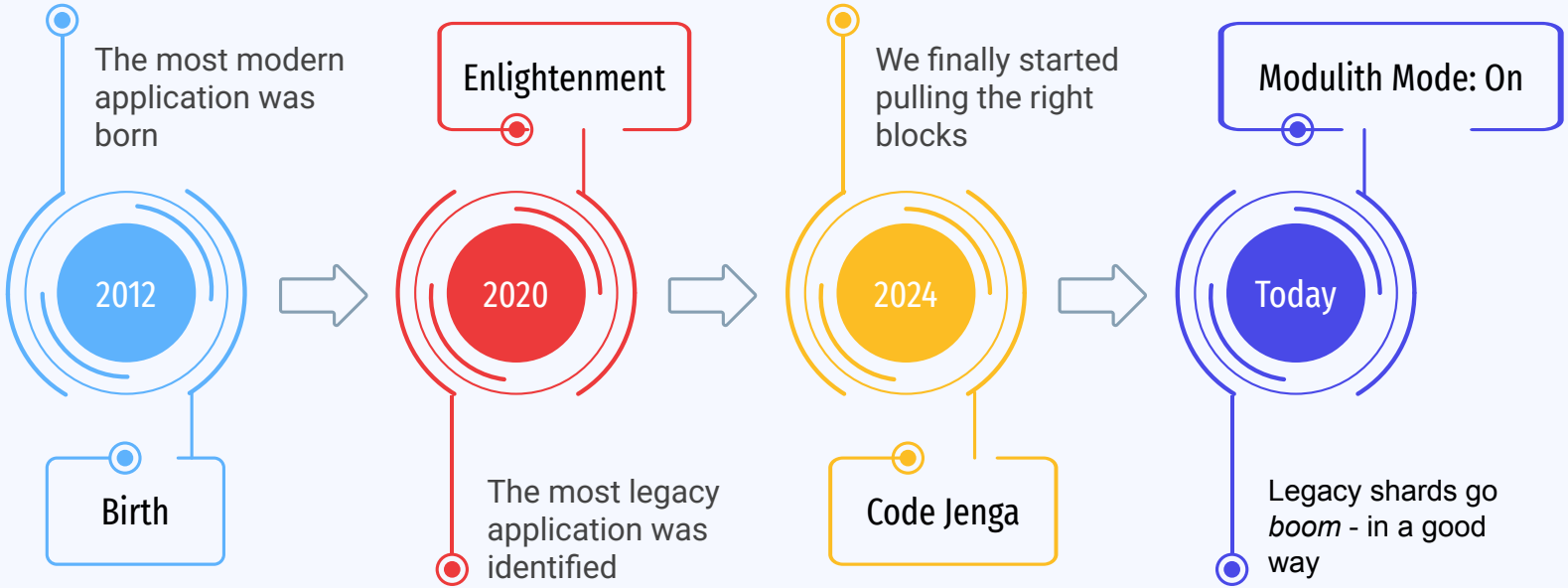
ADRs

Markdown Decision Records capture context-choice-consequences; optional C4 sketches add clarity when needed.

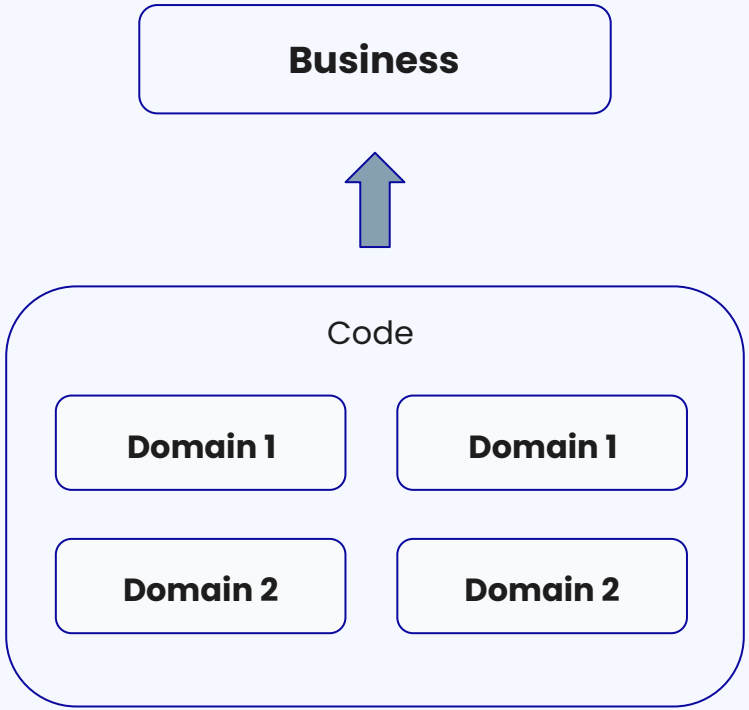
DDD structure

The repo renders its own domain handbook

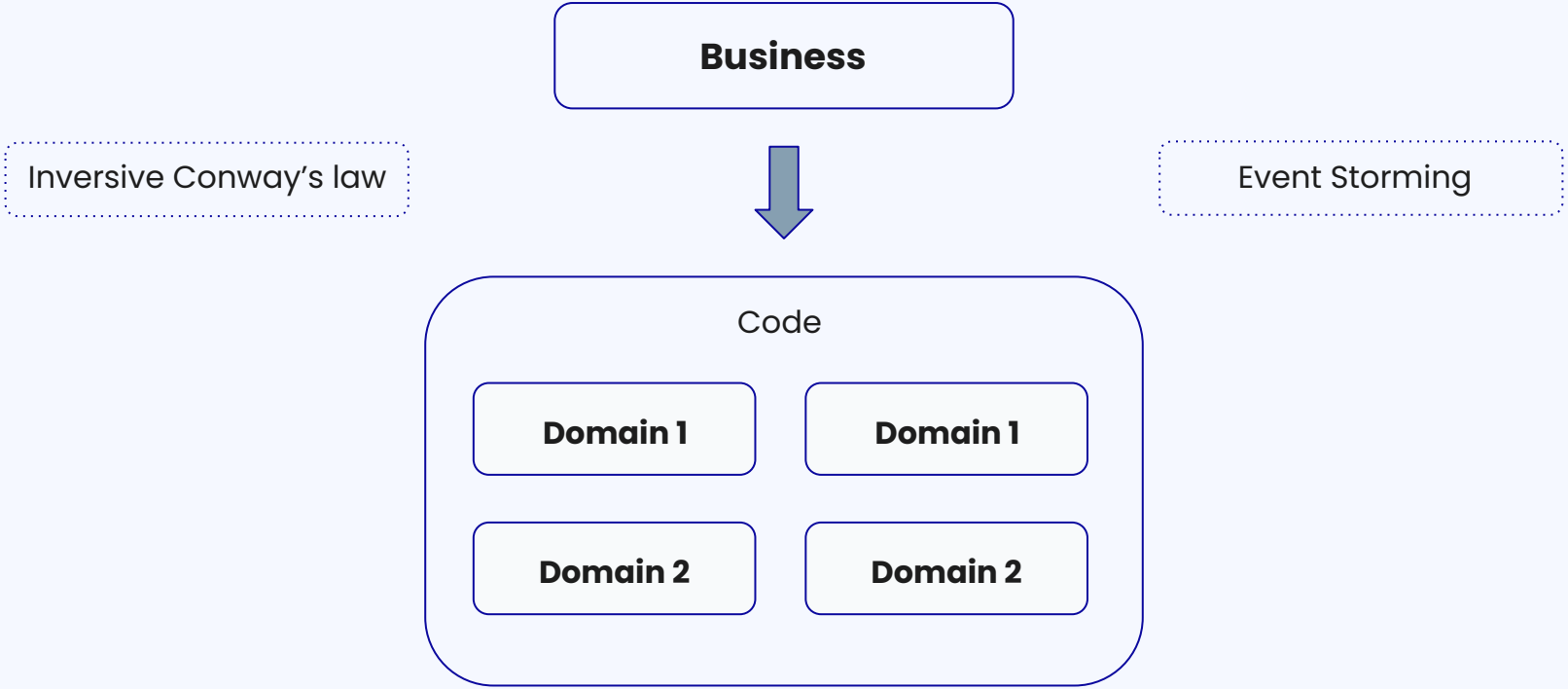
TalentLMS Timeline



Code restrictions are removed



Code restrictions are removed



Embrace the Monolith

Centralised business logic

Your Monolith contains all your business logic, and that's a good thing.

How we used the Monolith to our advantage

Implemented asynchronous workers with minimal changes.
An external Node.js server subscribes to domains events and passes them to dockerized versions of our Monolith.

Microservices or Micro..legacies?

Started splitting the Monolith into different services.
Example: Authentication

These changes enabled us to:

01

Formal mentorship

Every new hire is assigned a dedicated mentor.

02

Enabling People

Internal Hackathons and Mob sessions

03

AI Services

Decoupled modules let us plug in recommendation and analytics models on demand

04

Engineering Metrics

Dashboards put delivery speed, quality, and team health on one screen

“Don't seek magic buttons or magic solutions”

— Nick Tune, Architecture Modernization

Thanks!

Do you have any questions?


linkedin.com/in/aggelos-bellos/


aggelosbellos7@gmail.com


medium.com/@aggelosbellos