Platform Engineering's Inferno

Agenda

Introduction

This journey will start with an introduction of your Virgil

Circles & Demons

The major mistakes that can be made

The kind of people you will encounter during your journey

How to deal with (some of) the difficulties

Heaven?

What you can do to finally reach Heaven

Matteo Bianchi

@mbianchide v

Cloud Native Platform Consultant

- Consulting companies on Cloud Native & Platforms
- CNCF Ambassador
- Organizing Cloud Native events in the Netherlands
- Kubernetes v1.32 comms lead

Former startup CTO, Staff DevOps/SRE, SWE

Some fun facts...

- I used to live in Ravenna
- I'm a metal(core) singer
- I hated literature until reading the Divine Comedy
- I am still jet lagged from KubeCon NA, lol.

TAG ENVIRONMENTAL SUSTAINABILITY

Disclaimer

Some references to the Divine Comedy could be inaccurate for the purpose of better comparison with Platform Engineering.

— “Lasciate ogne speranza, voi ch'intrate”

— “All hope abandon ye who enter here”

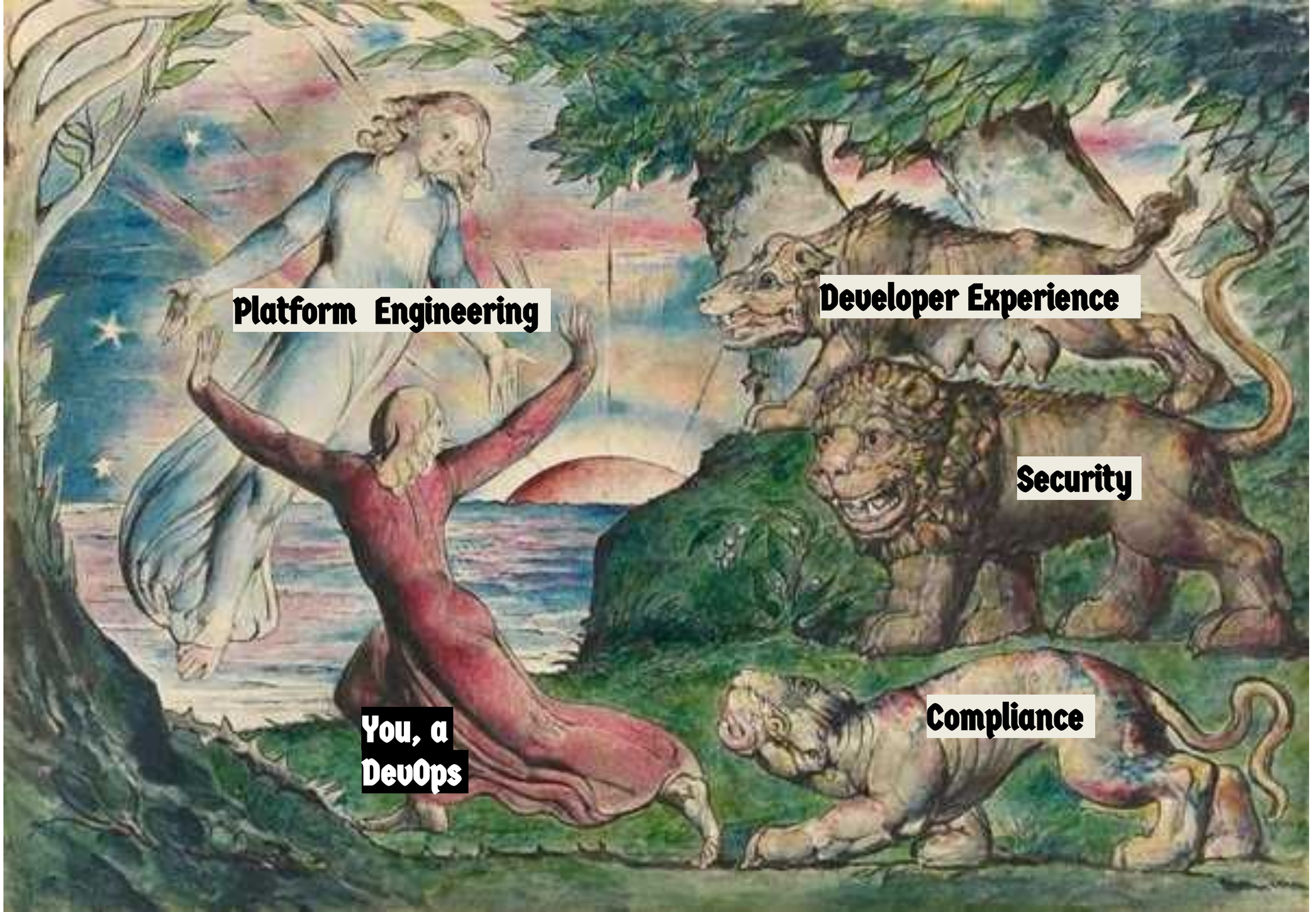
Disclaimer II

Whenever I say “users” or “customers” I mean **developers** .

Disclaimer III

Which is actually just a big TLDR;

You already have a platform.
You just don't treat it like one, **yet.**


Platform Engineering
Developer Experience
Security
Compliance
You, a DevOps

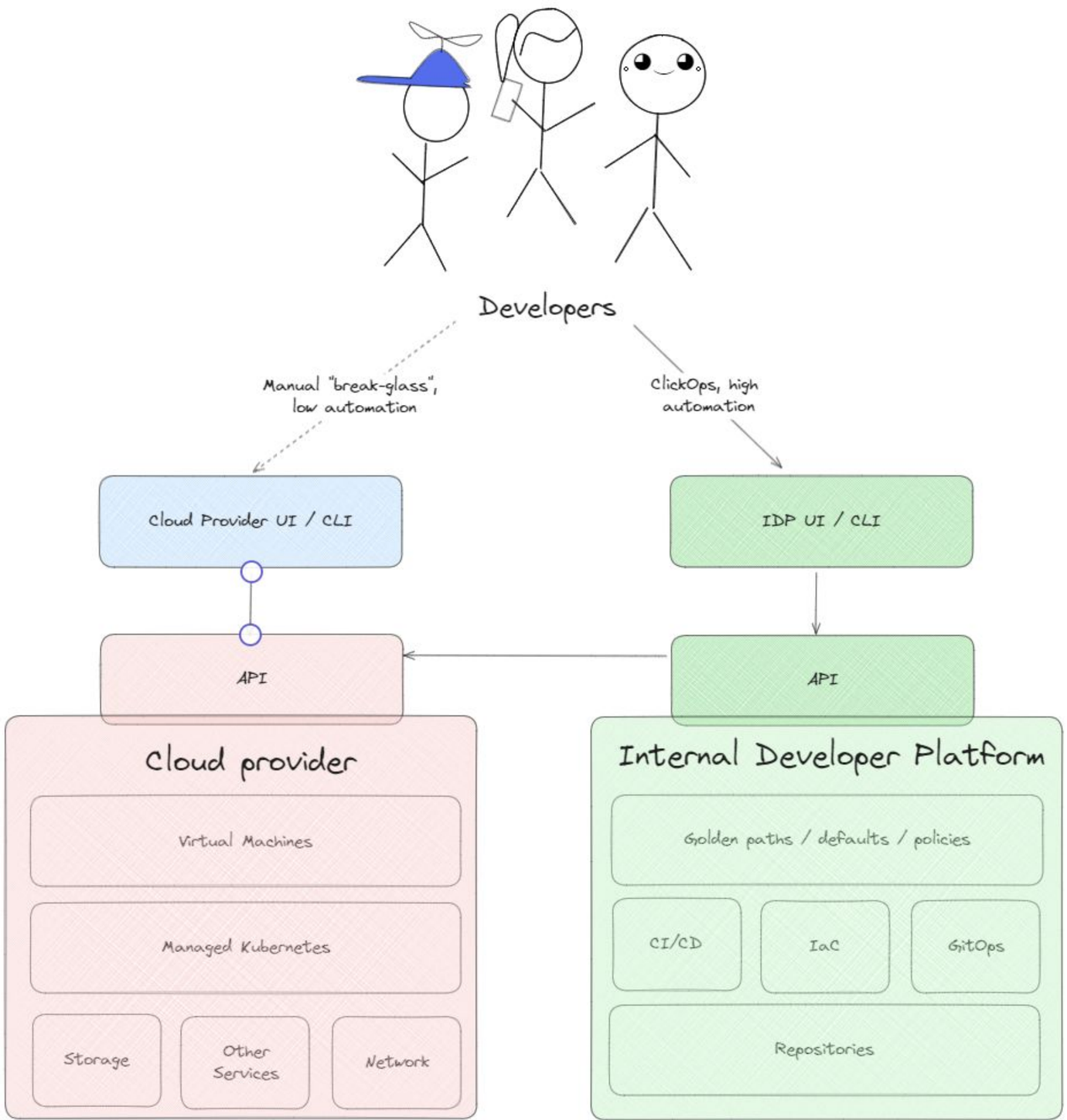
What is this Platform?

An Internal Developer Platform (IDP) is composed of many different techs and tools, integrated. Its aim is lowering cognitive load on developers and enabling self service, without giving up control, security and compliance.

–

It's an internal product, built and maintained for the developers, with the developers.

— source: internaldeveloperplatform.org

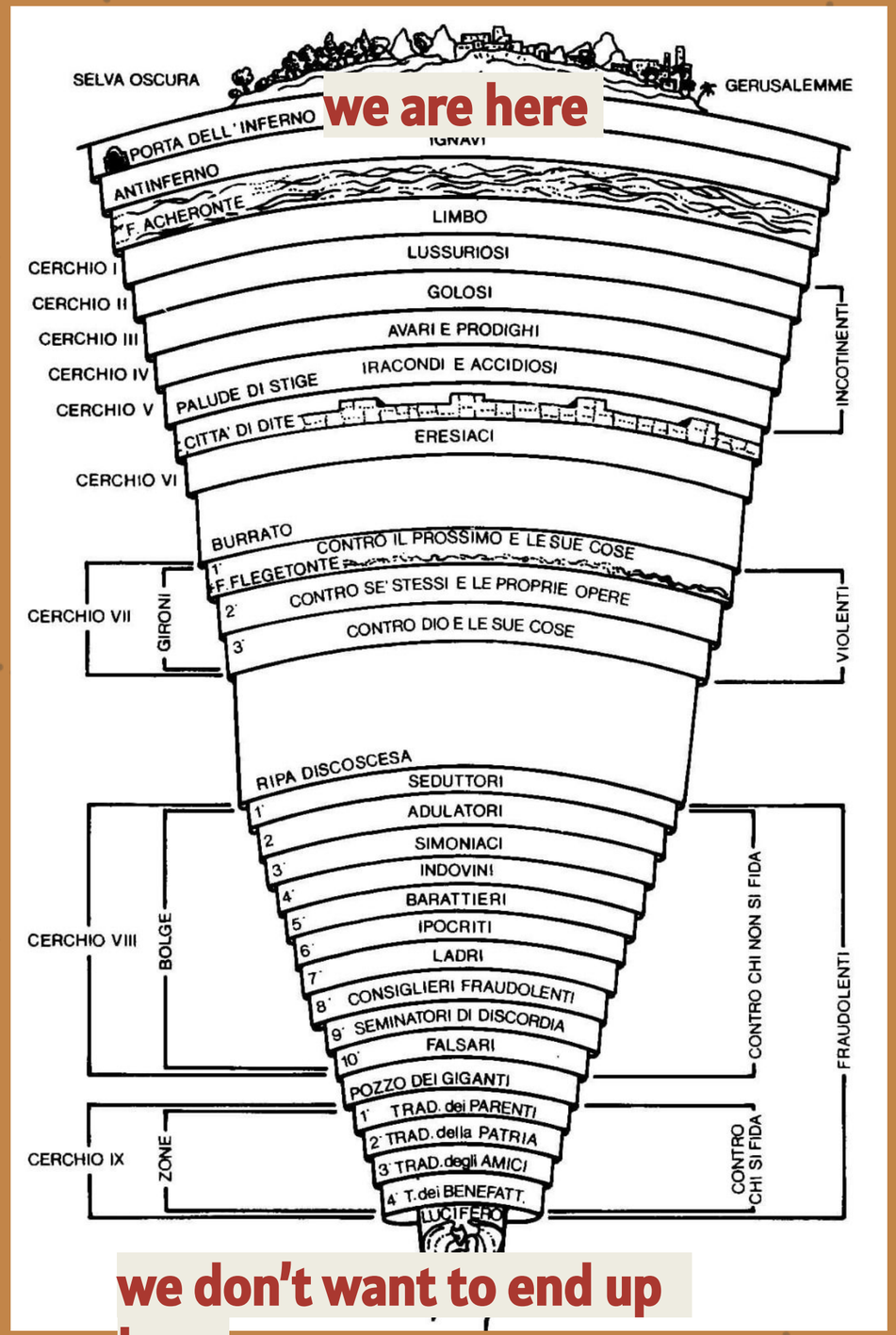

Developers
Manual "break-glass", low automation
ClickOps, high automation
Cloud Provider UI / CLI
IDP UI / CLI
API
API
Cloud provider
Virtual Machines
Managed Kubernetes
Storage
Other Services
Network
Internal Developer Platform
Golden paths / defaults / policies
CI/CD
IaC
GitOps
Repositories

Inferno

Each circle represents a different sin, adding a layer of agony and despair for both Developers and DevOps people.
We will venture through poorly designed platforms, chaotic labyrinth of configurations, operational bottlenecks and vulnerabilities.

Trapped in a nightmare of misaligned priorities between business requirements and developer needs.

Are you ready to count your sins?



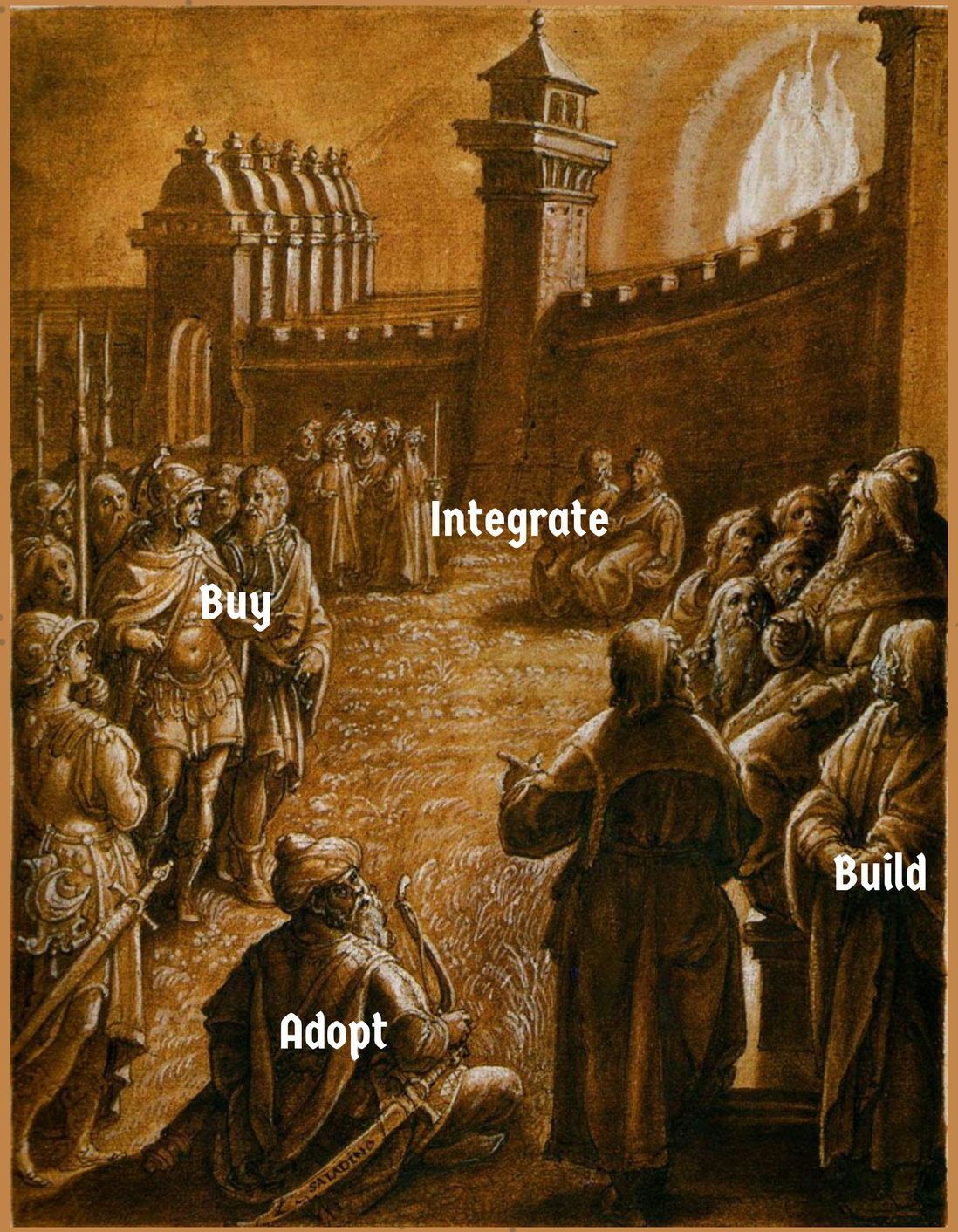
I - Limbo

This place is inhabited by legacy developers whom we can't blame for missing out on Platform Engineering.

At the same time we have an eternal discussion going on between the other guests of this circle.

Build vs Buy (vs Adopt vs Integrate)

Developers discussing this dilemma are doomed. Here to stay, until the dice reads 5 or 8.



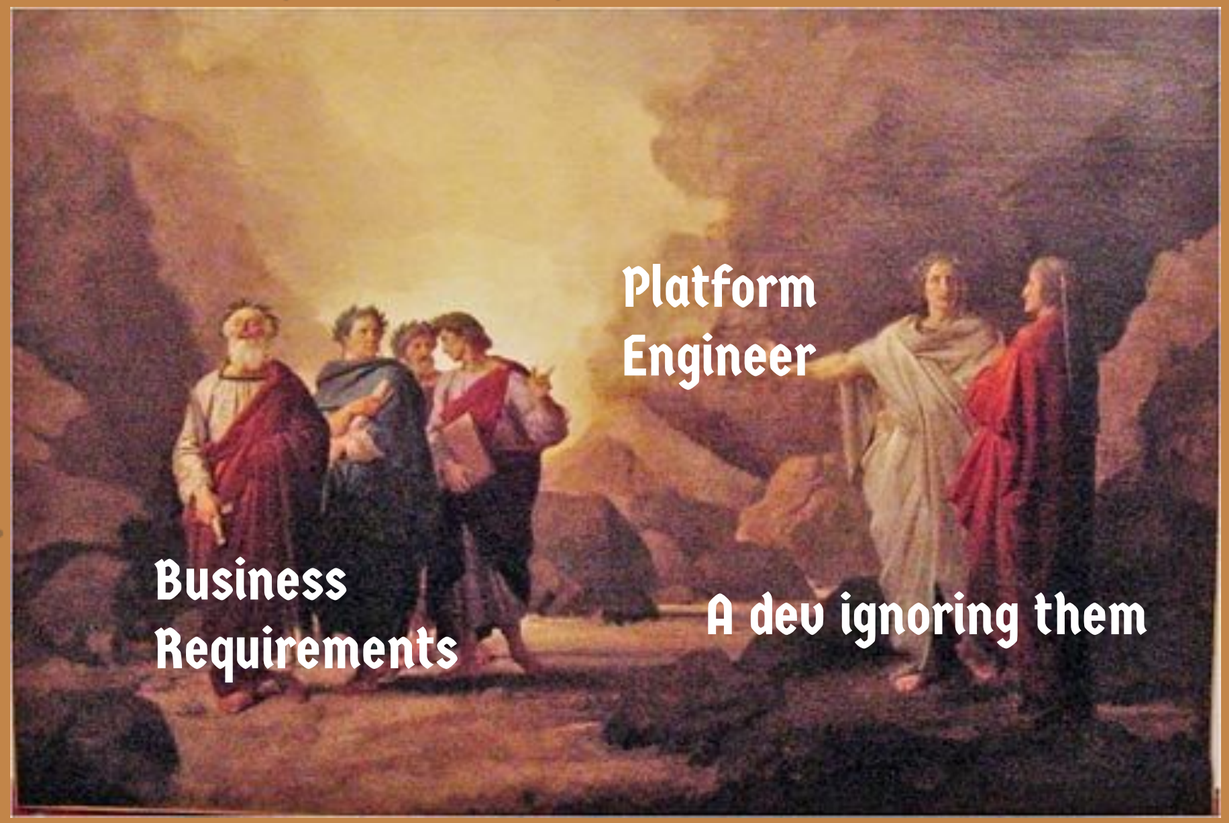
I - Limbo

What if I told you that this is a false choice?
There's no such dichotomy.

The purpose of a platform is not to be a greenfield amusement park for you developers

A platform enables your team to focus on your core business, increase the output and decrease time lost on undifferentiated and repetitive tasks.

Ask your developer what they need, observe where they spend most of their time and act.

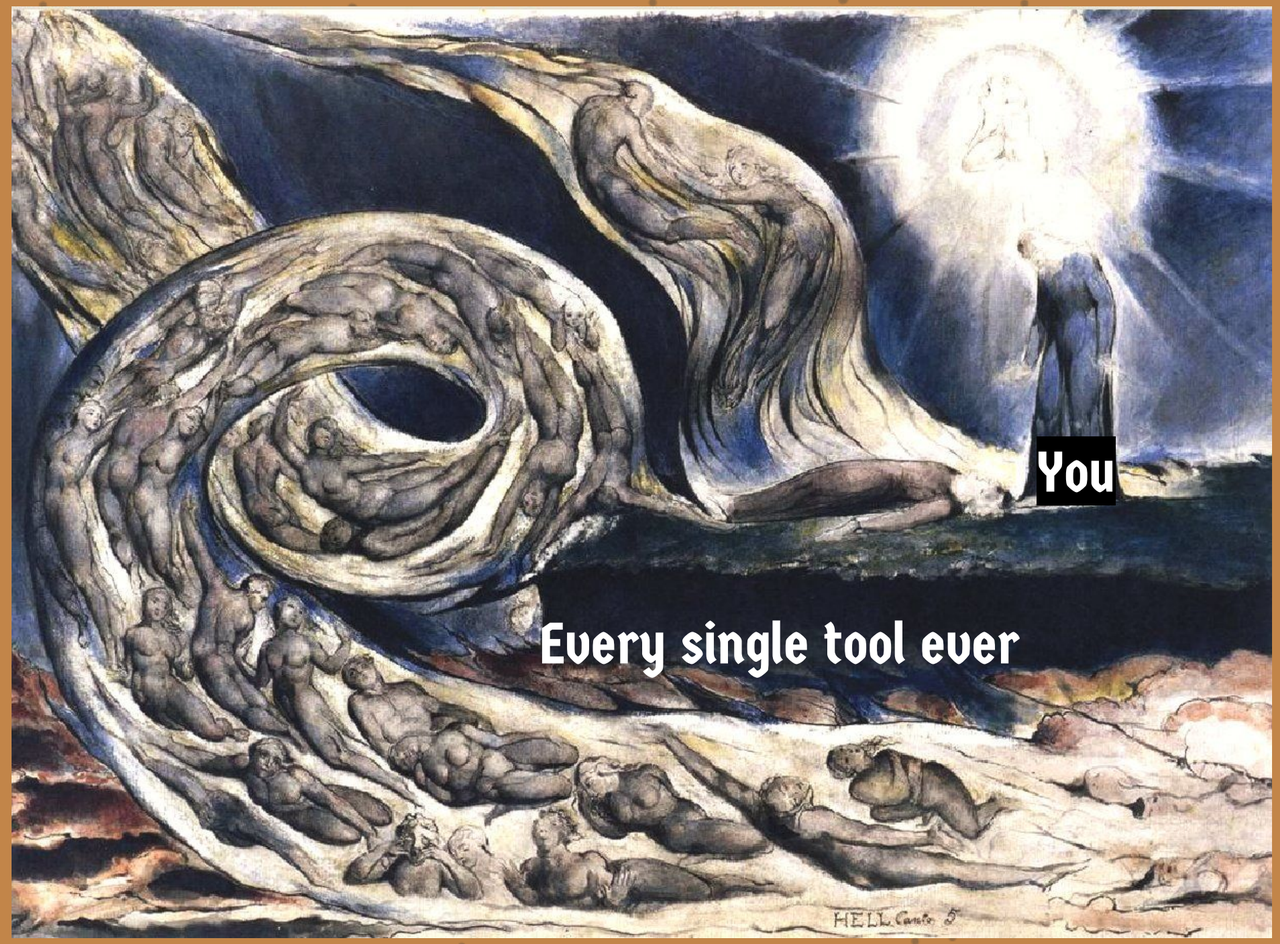


II - Lust

Excessive focus on extravagant and advanced features at the expense of core functionality.

A vortex of cutting-edge tools, elaborate architectures and over complicated components, difficult and costly to maintain and scale.

Particularly true when building more than buying because bought features are immediately expensive, while building them comes up with the bill much later.



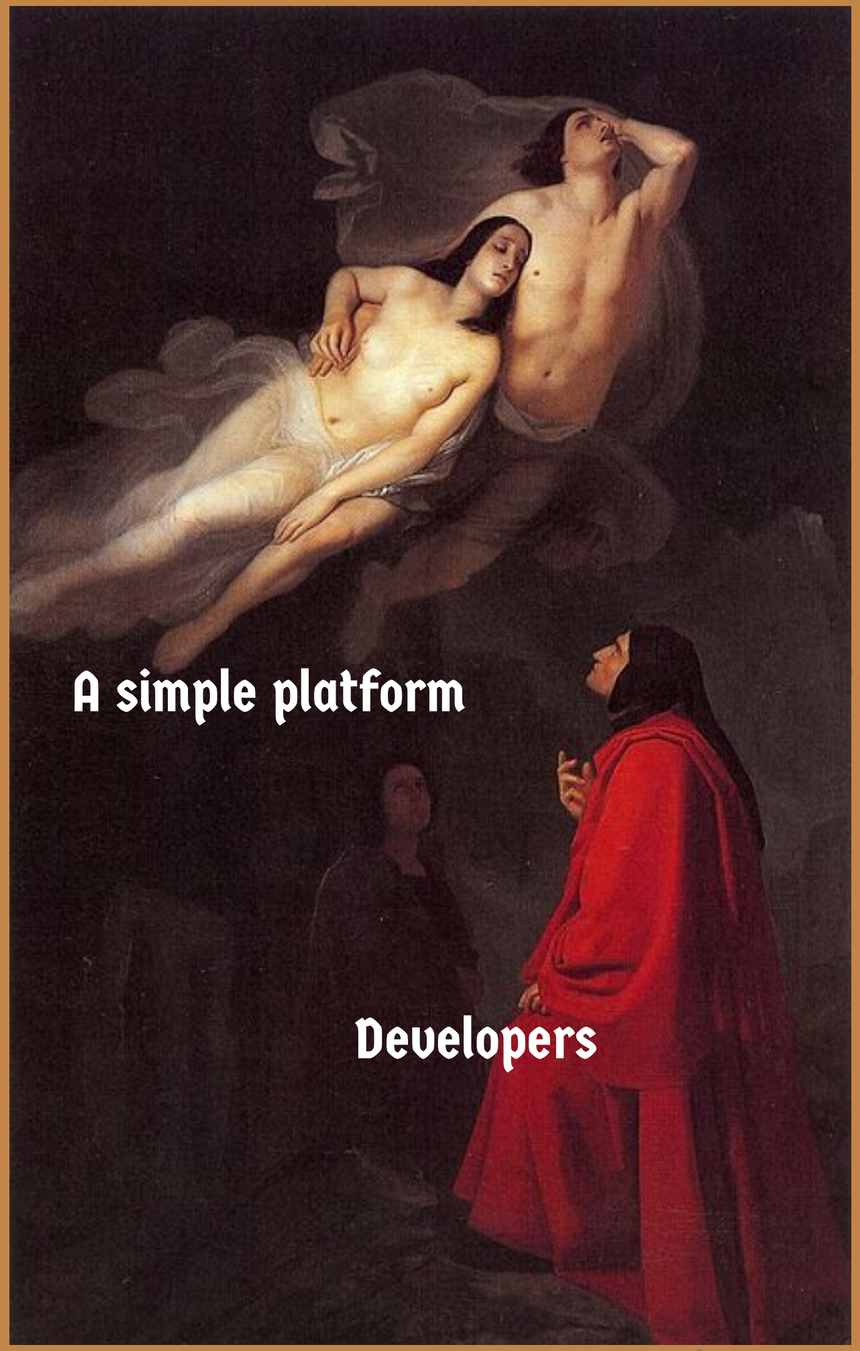
II - Lust

The goal of a platform is to be simple, reliable and efficient.
In order to achieve these, a platform has to start with the Thinnest Viable Platform (TVP)

"*the smallest set of APIs, documentation, and tools needed to accelerate the teams developing modern software services and systems.*" [1];

In short, a platform is whatever your team needs to go fast(er).

[1] github.com/TeamTopologies/Thinnest-Viable-Platform

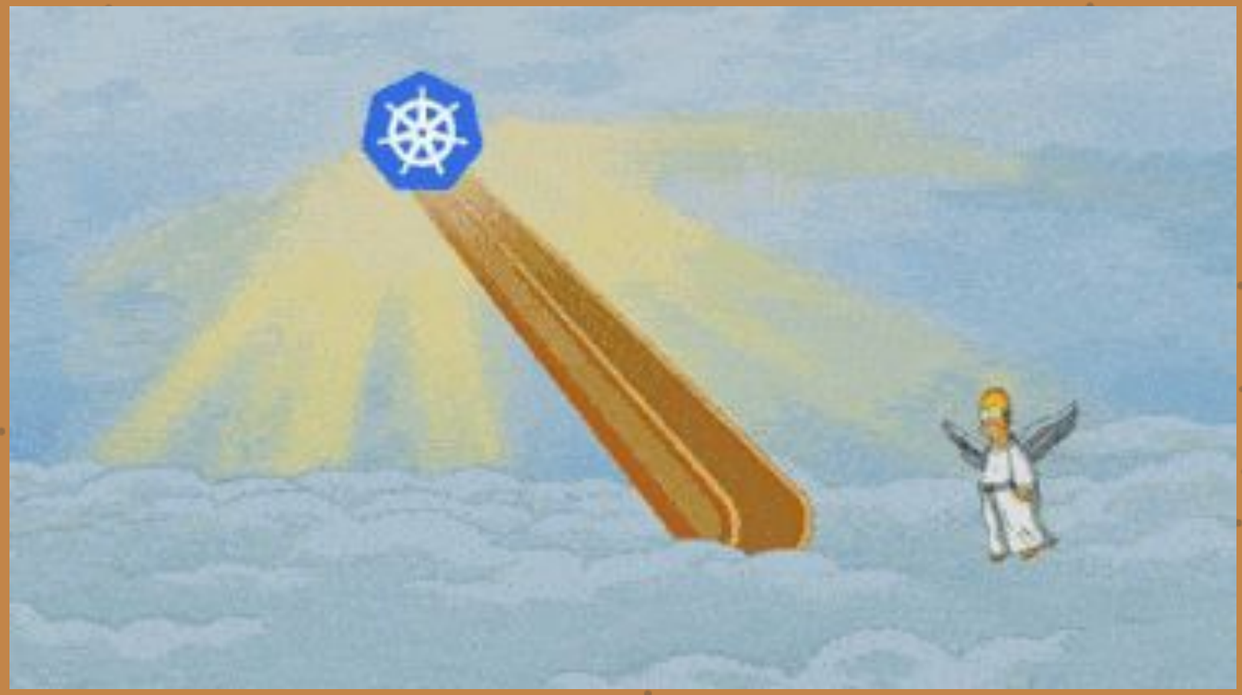


III - Gluttony

There is an insatiable appetite for integrating the largest amount of tools and services, without a rationale.

Leaving developers (your customers) with an overwhelming array of options to choose from.

This leads to an excessive amount of time, budget and resources spent for both building such integrations and creating any standard.

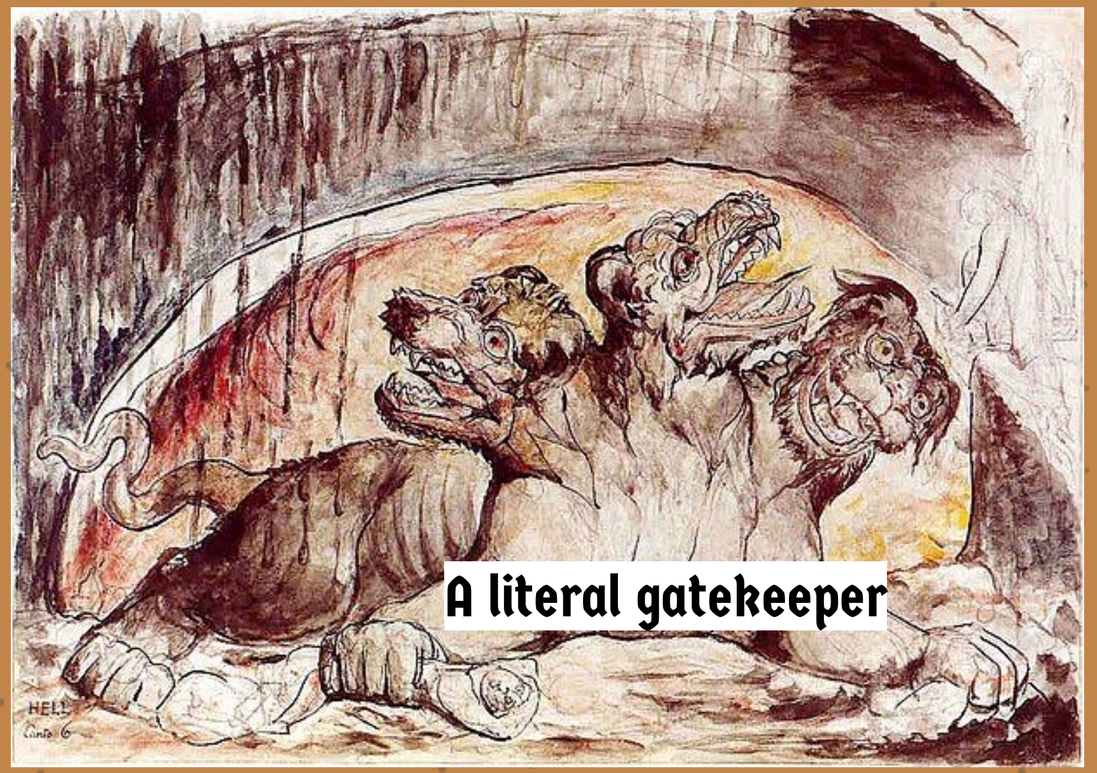


III - Gluttony

To avoid tool sprawling a Platform Engineer should act like Cerberus, actively gatekeeping developers from having too many options.

This helps to create a platform where performance, usability, and DevEx are satisfying.

Finding a trade-off with your developers, when evaluating both legacy and new tools is key.

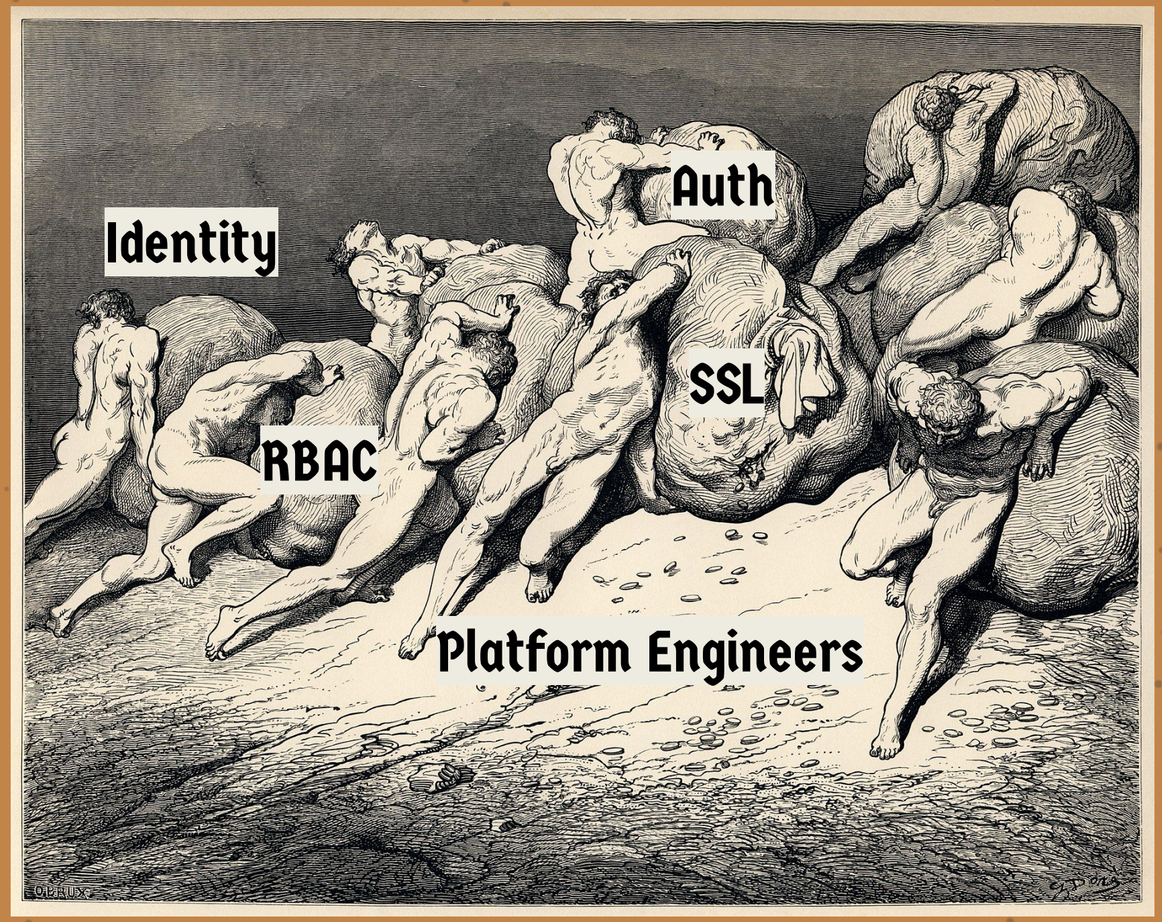


IV - Greed

Filled with an insatiable desire for control and dominance over the platform, Platform Engineers end up moving around maintenance burdens over and over, reinventing the wheel with each iteration.

Here we also find the turn-key dreamers that bought a platform thinking it would magically solve their team topology issues.

Duplicated efforts, increased complexity and lack of control, compromising the platform effectiveness and long term sustainability.



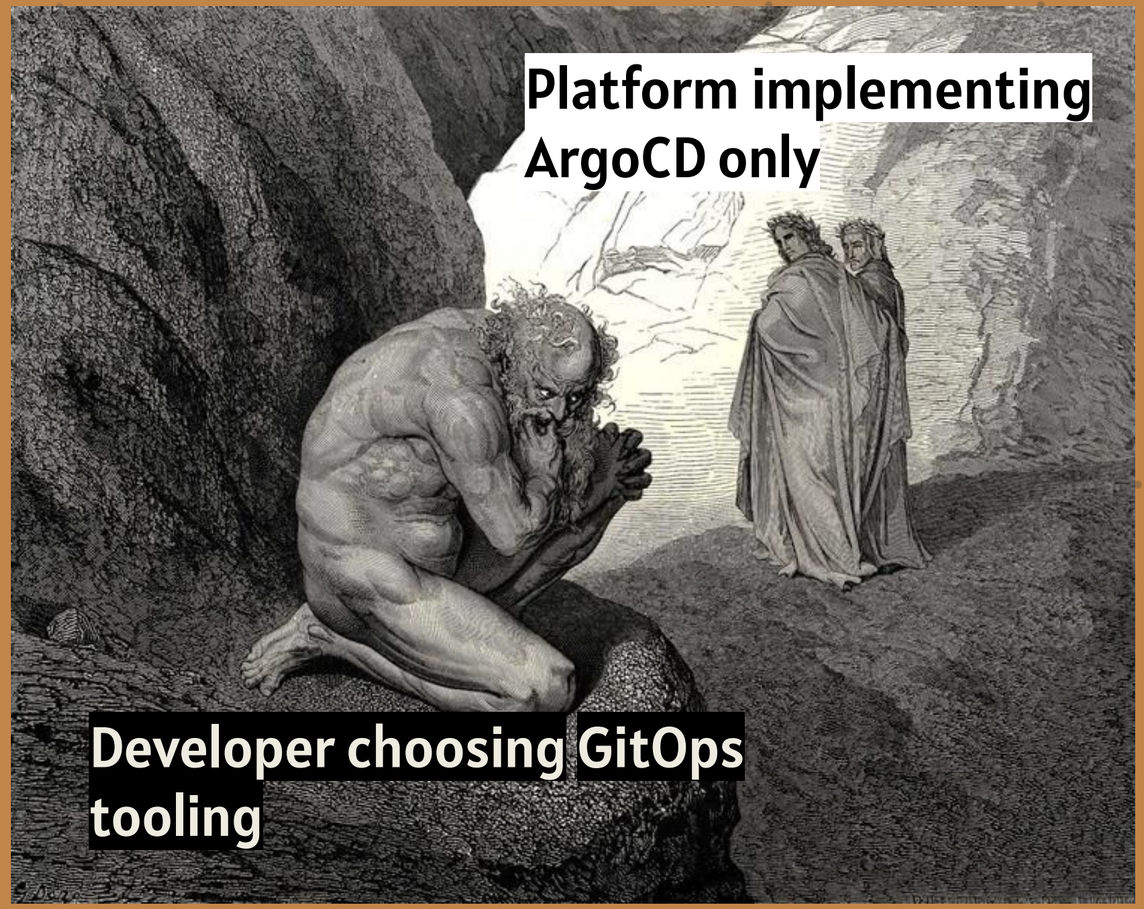
IV - Greed

To avoid such mistakes we need to define golden paths or the fastest way for a developer's code to reach production.

Golden paths include templates for various components, starting from repositories to infrastructure pipelines and anything in between.

The key is to eliminate unnecessary choices* and diminish the complexity budget.

*That can still be treated as edge cases if needed

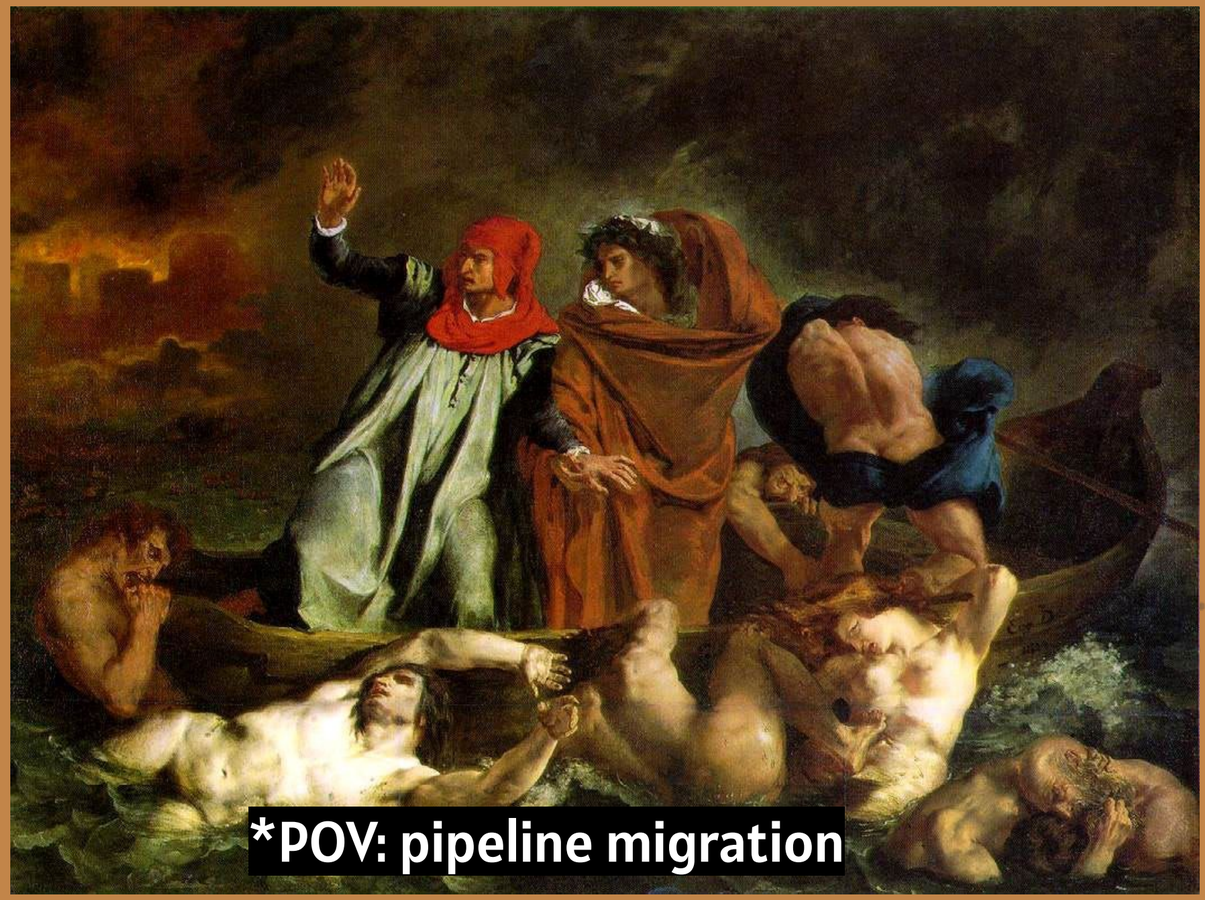


V - Wrath

Internal friction and competition will be devastating for the platform, dooming its destiny.

Building it while trying to get adoption, avoiding competing implementations, migrating from A to B, lack of support after adoption, poor onboarding and no enabling team, scarce to no documentation, missing API interfaces, zero ownership, lack of product definition and no roadmap.

In a word: platform decay [2].



[2] Syntasso's blog on platform decay
*yes, I know that's not a POV.

V - Wrath

To minimize friction we can follow these steps:

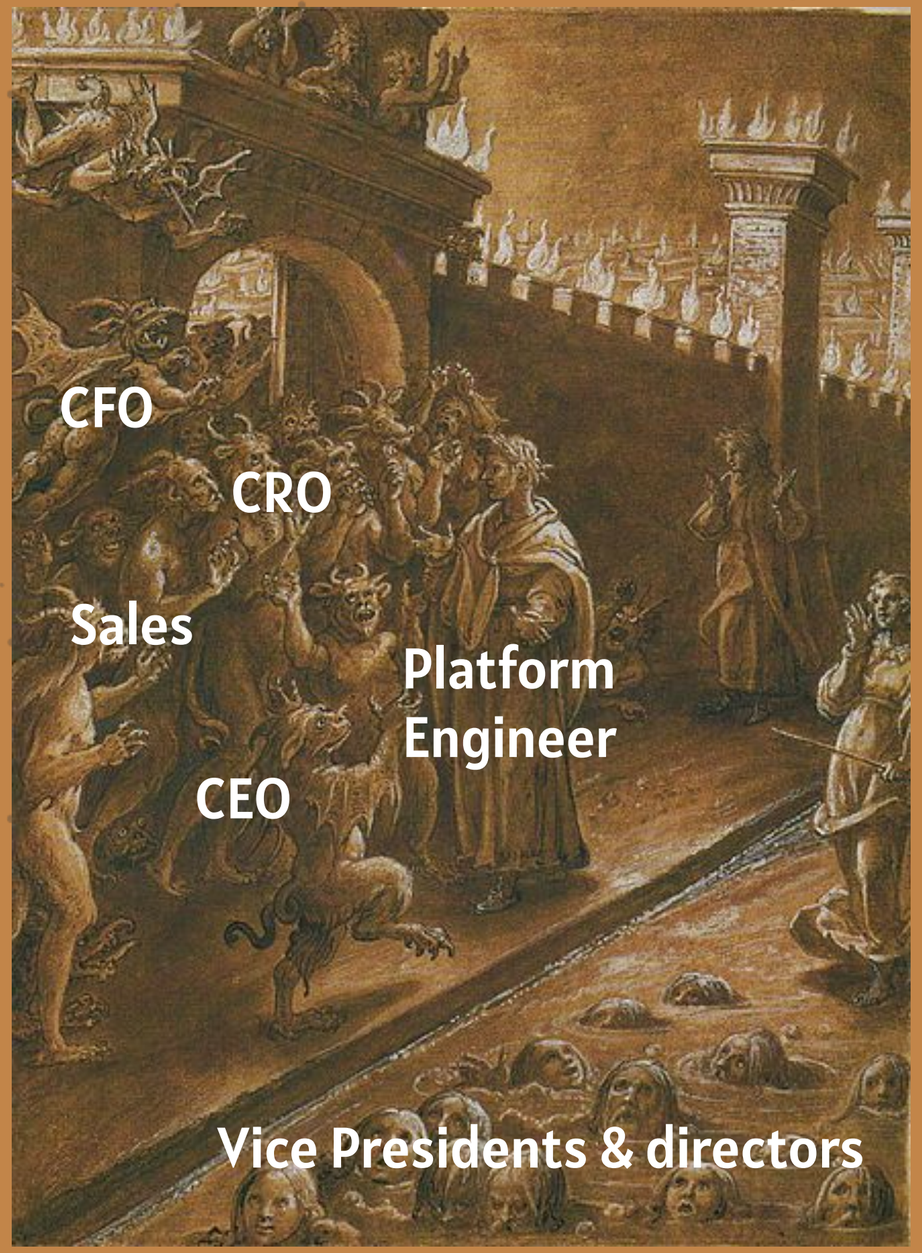
- start with small and value-driven steps - *what is the most time consuming activity for my dev team? Who is my pilot team/product?*
- avoid all-in-one automagic and big bang releases - *what is the minimal set of features to increase my platform value for the next version?*
- survey users after each (major) iteration
- keep day-2 in mind to drive adoption
- define simple and future-proof API interfaces

VI - Heresy

Platform Engineering is seen as heretic by management and rightfully so.

In this wicked place there is little to no mention to ROI (Return on Investment) nor any metric to measure the impact of the platform.

Budget spending is unjustified in the eye of the finance or FinOps team.

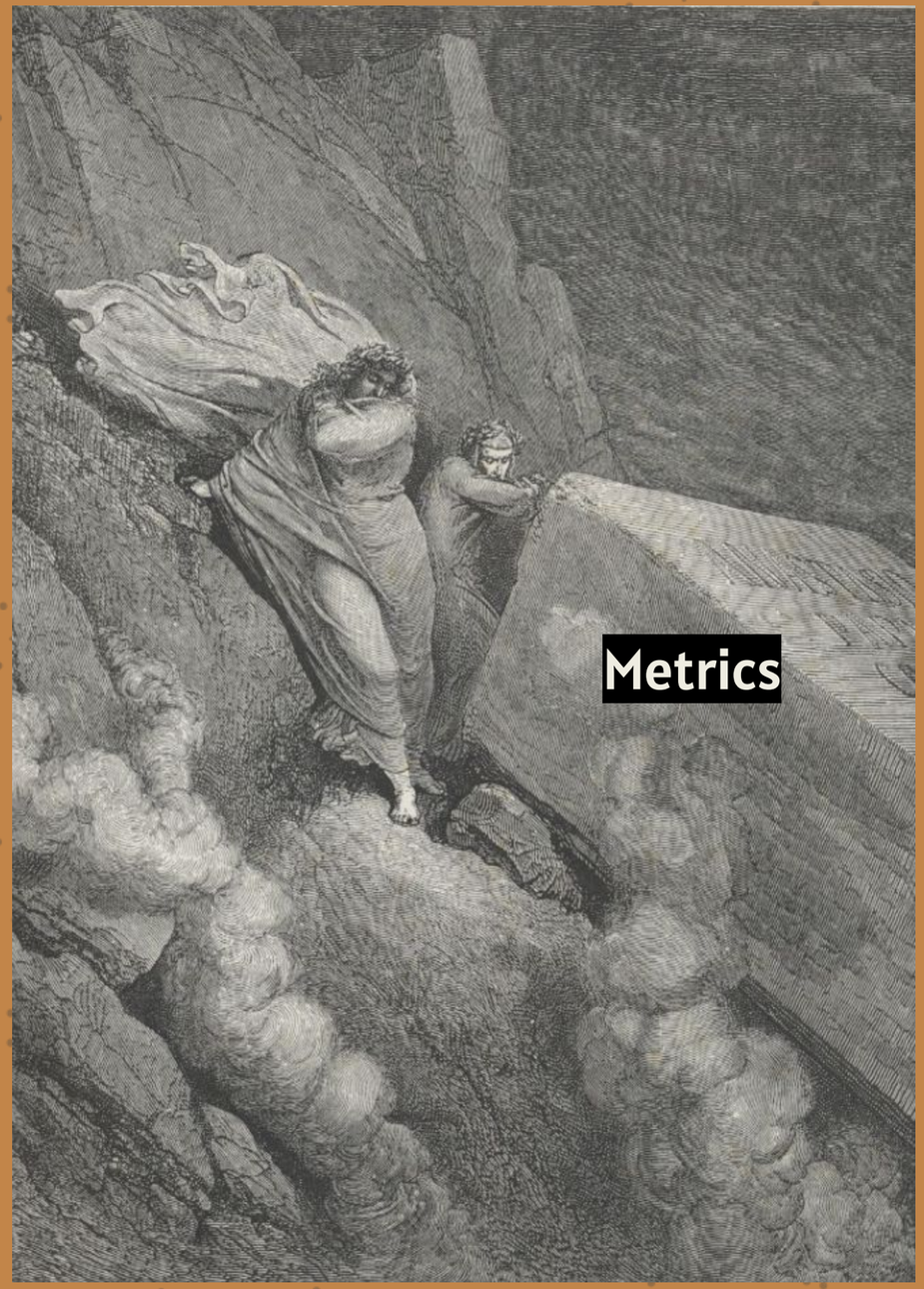


VI - Heresy

As we experienced before, communication is key and so is aligning the platform goals with business objectives (OKR).

Measuring against some form benchmark is a fundamental step to understand the entity of performance improvements brought by the platform.

We could leverage the DORA metrics and survey the developers' satisfaction periodically.

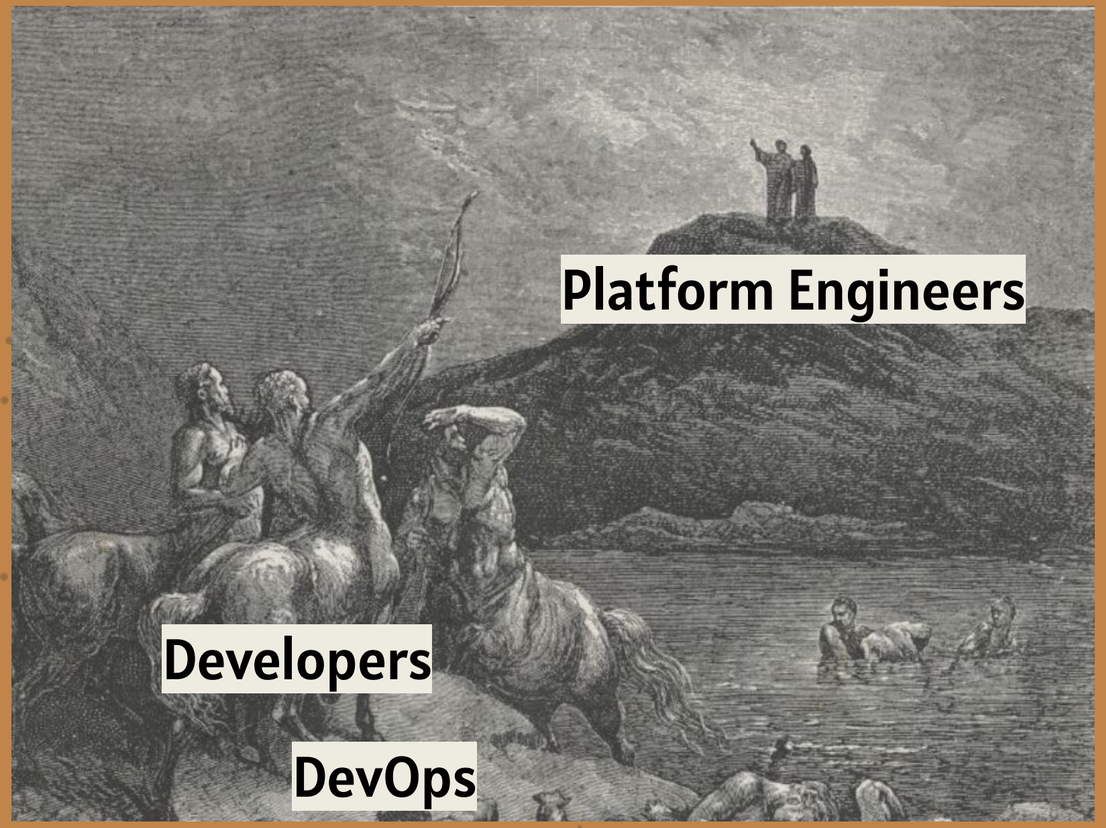


VII - Violence

Developers against your platform are equally dangerous.

This circles are filled with
"*but we've always done it this way*" developers
and
"*won't that platform steal my job?*" DevOps.

You have to fight resistance within your teams and convince them that this platform is only going to be good if they buy (and help build) it!

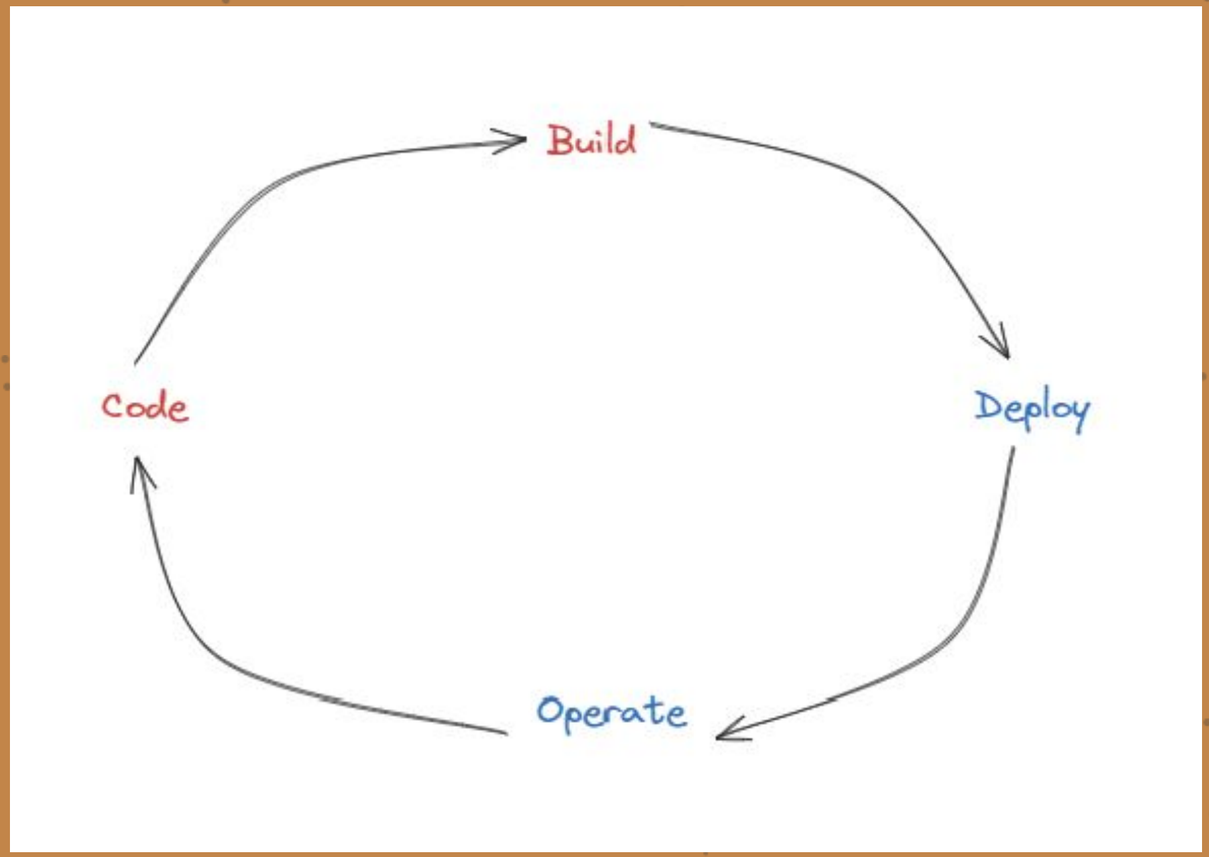


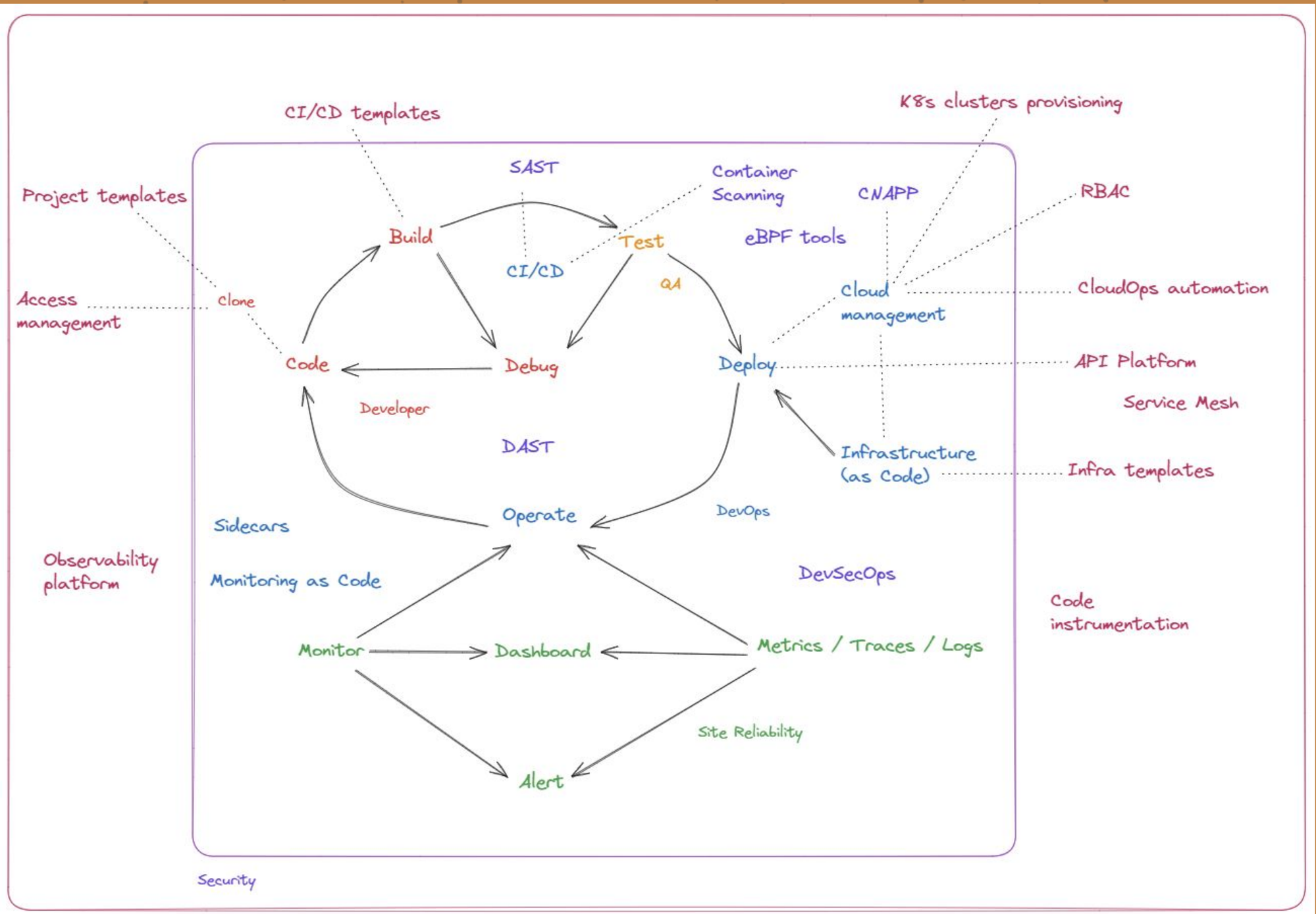
VII - Fraud

Don't we all know this schema?
Don't we use it all the time to define SDLC or even DevOps processes?

Whoever ended up here kept lying about how developing software works and now, after having experienced it, we finally know it.

Let's see what reality looks like.



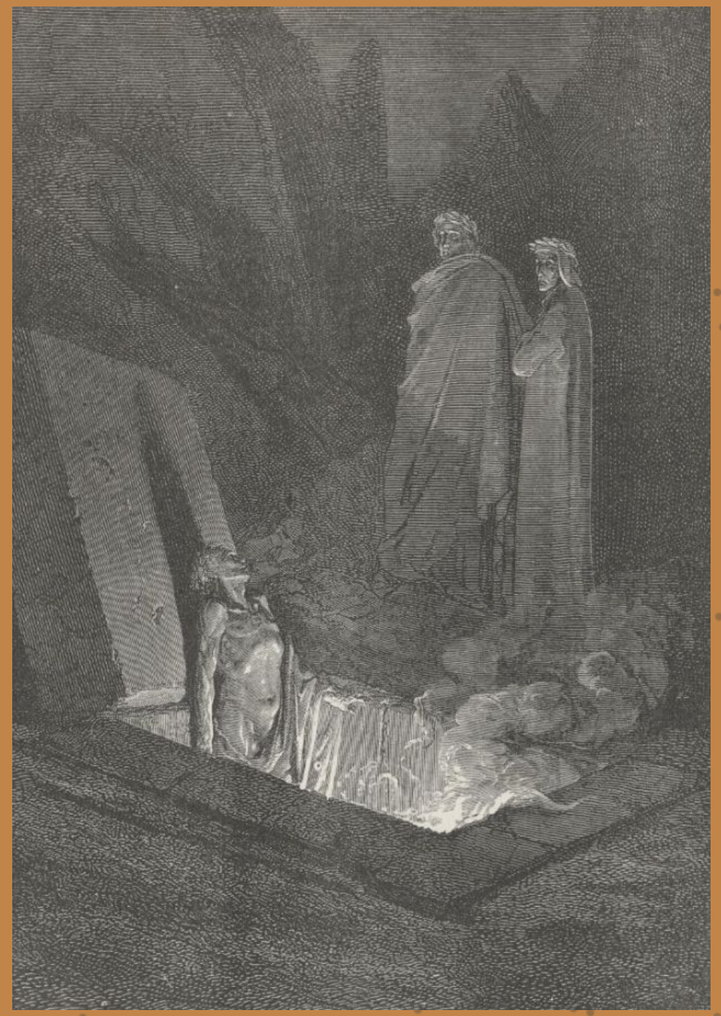

CI/CD templates
K8s clusters provisioning
SAST
Container Scanning
CNAPP
RBAC
Project templates
Build
Test
eBPF tools
CI/CD
QA
Access management
Clone
Cloud management
CloudOps automation
Code
Debug
Deploy
API Platform
Service Mesh
Developer
DAST
Infrastructure (as Code)
Infra templates
Sidecars
Operate
DevOps
Observability platform
Monitoring as Code
DevSecOps
Code instrumentation
Monitor
Dashboard
Metrics / Traces / Logs
Site Reliability
Alert
Security
Platform

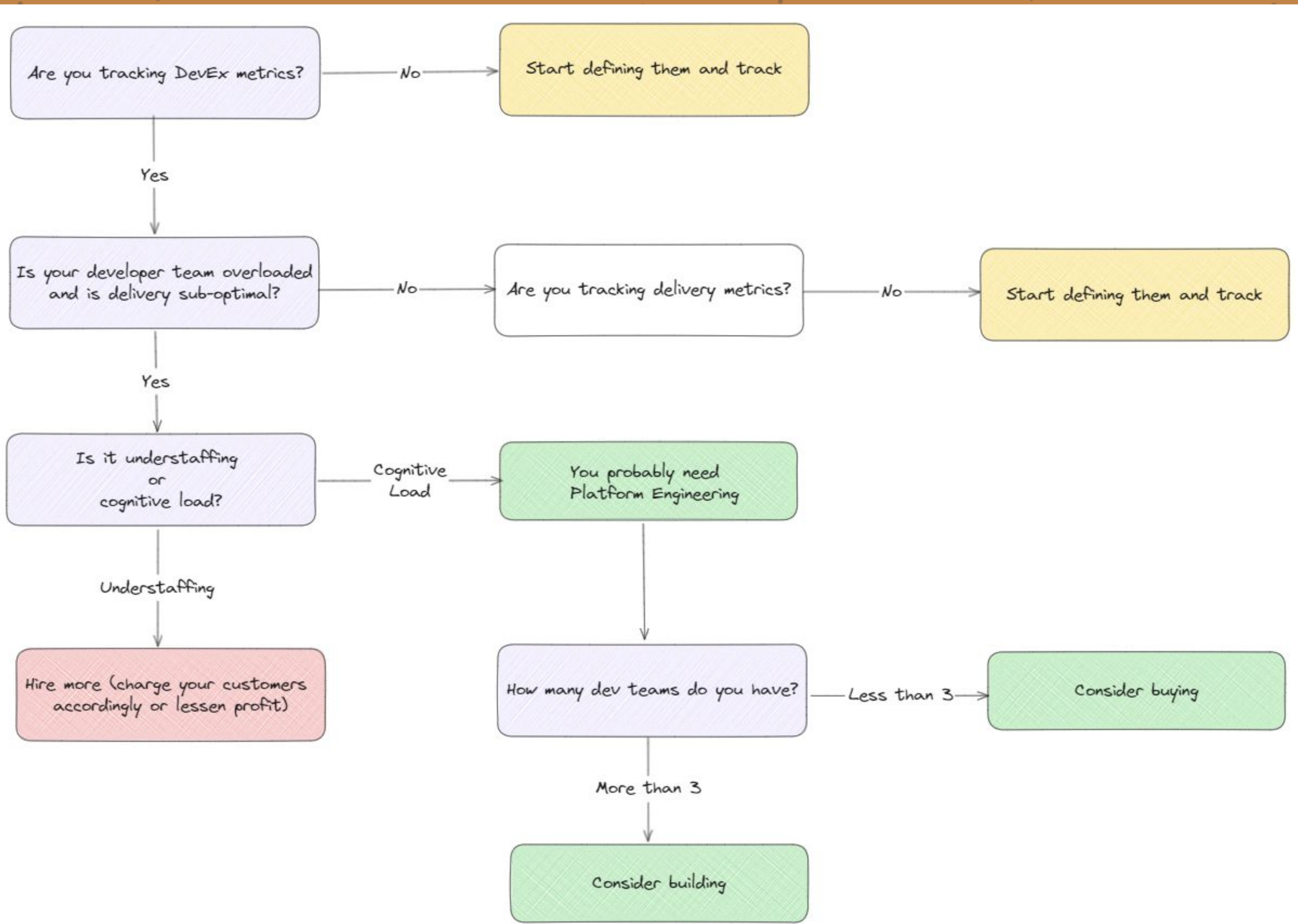
VII - Treachery

Here developers trust has been betrayed, they were promised a working platform but instead they got a labyrinth of tools, not integrated, unused, slowly decaying.

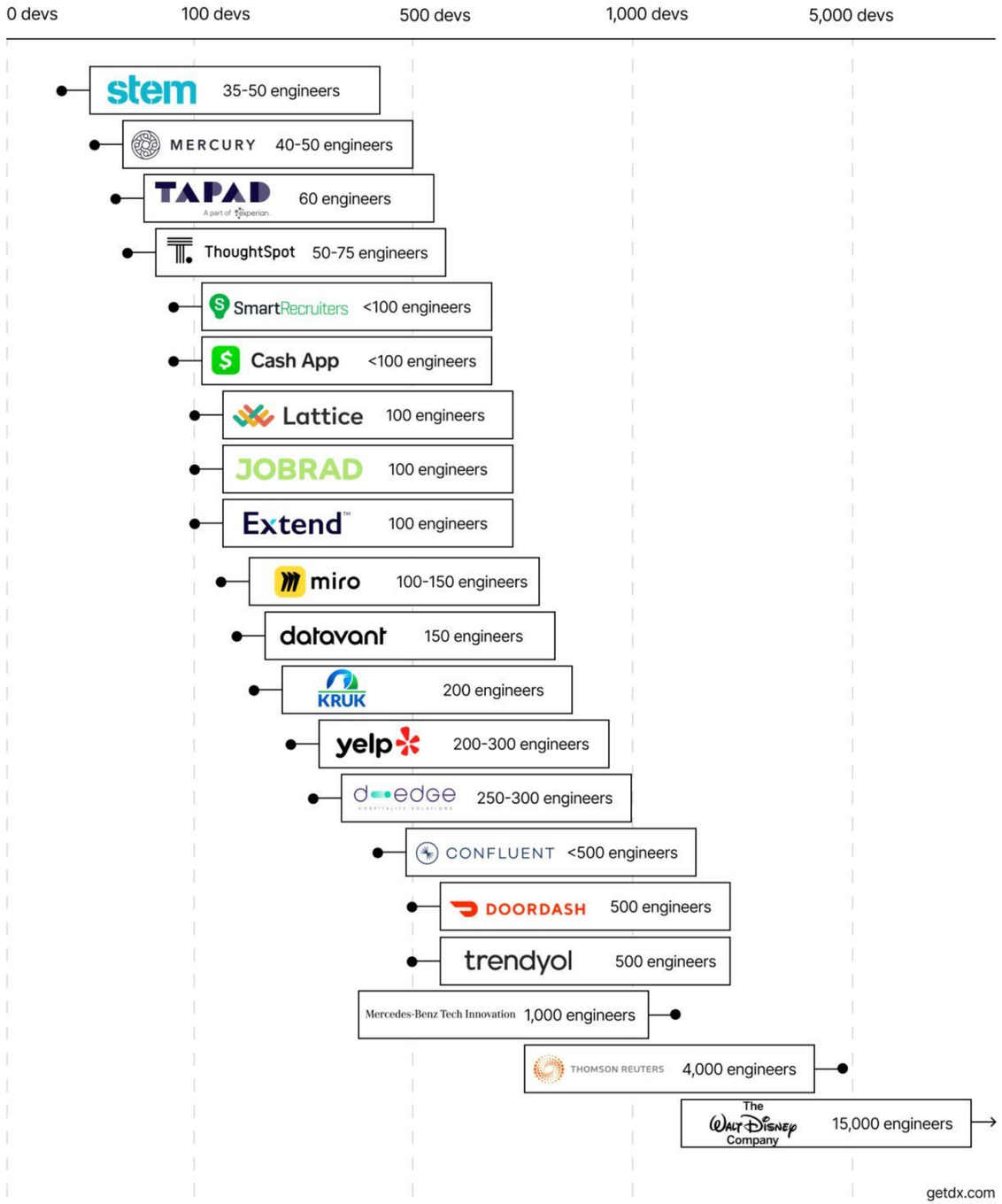
Here lie all the ones who failed to find the Platform Market Fit and their entire platform teams are doomed.

There is a way out of this maze.




Are you tracking DevEx metrics?
No
Start defining them and track
Yes
Is your developer team overloaded and is delivery sub-optimal?
No
Are you tracking delivery metrics?
No
Start defining them and track
Yes
Is it understaffing or cognitive load?
Cognitive Load
You probably need Platform Engineering
Understaffing
Hire more (charge your customers accordingly or lessen profit)
How many dev teams do you have?
Less than 3
Consider buying
More than 3
Consider building

When should you establish a Developer Productivity team?



TLDR;

- It's all about company culture and people, you cannot enforce Platform Engineering top-down;
- Buy if your platform need is already well solved, try different platforms but remember that agnostic and open-source diminishes lock-in;
- Build if you can afford the time, effort and long-term maintenance opportunity cost;
- Always measure your platform effectiveness against metrics tailored on business objectives;
- Start small, solve a problem for your developers that shows the business value of your Platform;
- Platforms have to find their own market fit, in a market (your company) that requires a way different GTM strategy involving users to shape, define and contribute to the product they need. Your development teams are key to the PMF;

Resources

- A Platform Engineering manifesto - https://github.com/mbianchidev/platform-engineering-manifesto
- A work in progress Platform Engineering roadmap - https://github.com/mbianchidev/platform-engineering-roadmap
- An illuminating post by Martin Fowler about platforms - https://martinfowler.com/articles/platform-prerequisites.html
- This slide deck is public at https://speakerdeck.com/mbianchidev/platform-engineerings-inferno

CREDITS

Slides Carnival

for the presentation template

Pexels, Pixabay

for the photos

Big thanks to:

KCD Denmark organizing team for having me on stage

CNCF Tag App Delivery - WG Platforms for the feedback

(https://tag-app-delivery.cncf.io/wgs/platforms/)

You, for attending this talk!

Q&A time!

p.s. yes, I'm open to consult on Platform Engineering and DevEx!

@mbianchide v

RESOURCE PAGE

Use these design resources in your Canva Presentation.

Fonts

This presentation template
uses the following free fonts:

TITLES:
GERMANIA ONE

HEADERS:
GERMANIA ONE

BODY COPY:
GERMANIA ONE

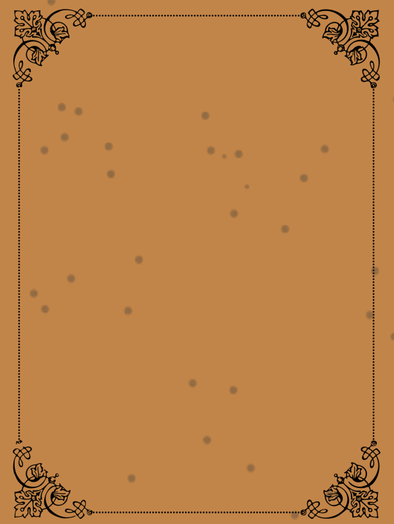
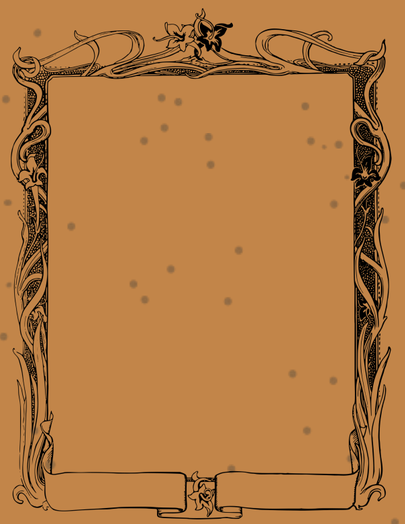
You can find these fonts online too.

Colors

#C28549 #E0CABB #E5D5BC #303023

Design Elements



DON'T FORGET TO DELETE THIS PAGE BEFORE PRESENTING.