The Monorepo Ecosystem

Bruno Guedes

CTO @ Mozantech

bruno.guedes@mozantech.com

What is a monorepo?

Definition:

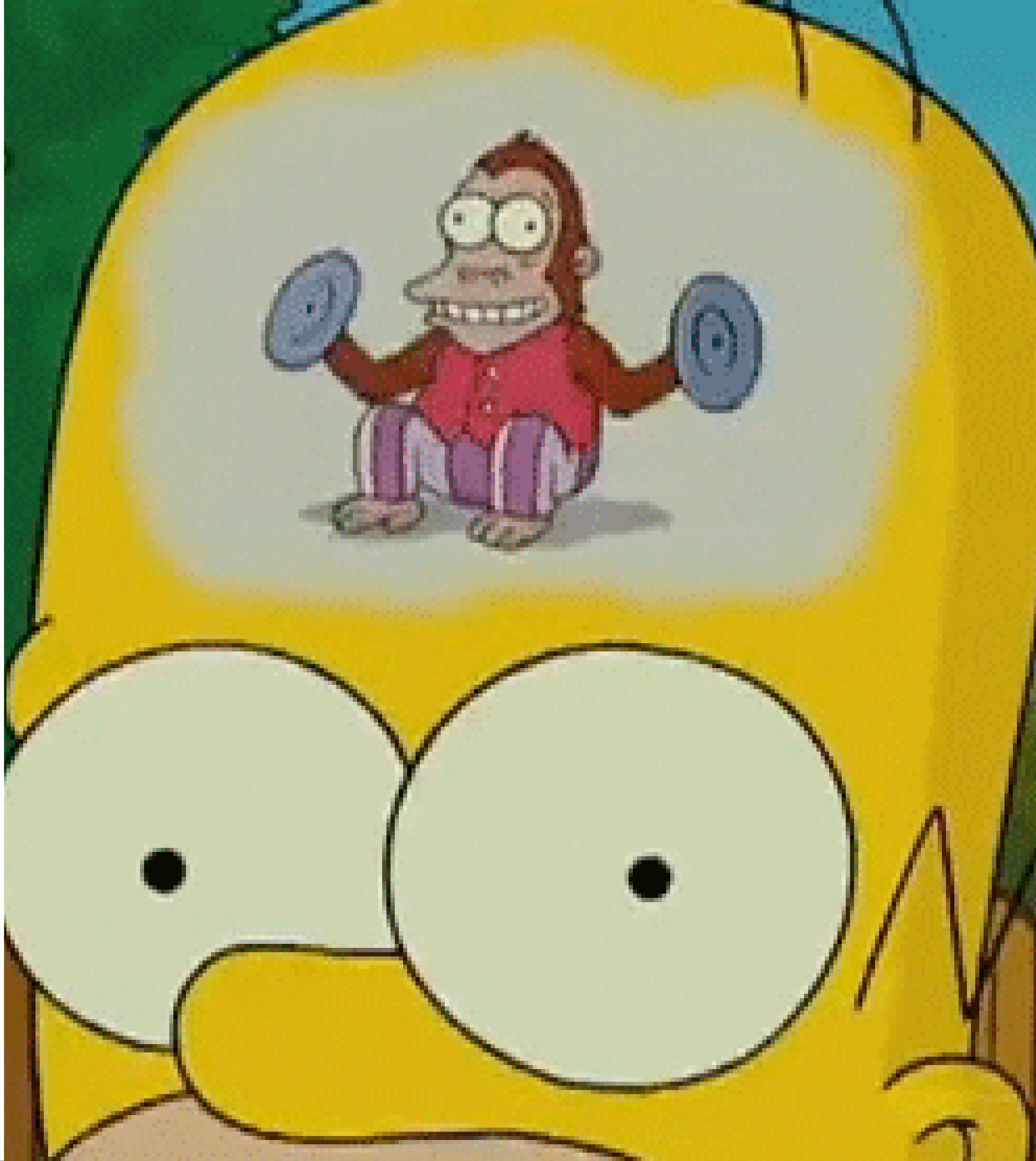
One day someone said:

"Let’s put all of our code inside the same repo!"

...

- A repository contains more than one logical project
- These projects are most likely unrelated, loosely connected or can be connected by other means

from antlassian.com



Monorepo vs Monolith vs PolyRepo?

Is a monorepo really different from a monolith? Aren't we going back?

"It is not hard to see that where you develop your code and what/when you deploy are actually orthogonal concerns".

from nrwl.io

- A *Monolith* is usually a single repo that's difficult to maintain. It's usually the habitat of a single application that has no connection to other internal systems.
- A *PolyRepo* is a standard where teams work across multiple repos that reduce the size and complexity of the codebase.
- A *MonoRepo* is an approach where all the codebase or most part of it is inside a single repo. With this approach, code is managed carefully and some challenges will appear that might slow down the dev process if there is no appropriate tooling in place.

Monorepo Disadvantages

Why is this not a standard?

- Git performance
- Tooling challenges
- Every user has access to all parts of the code (?)

Why should anyone use this?

Monorepo Advantages

It’s not all bad…

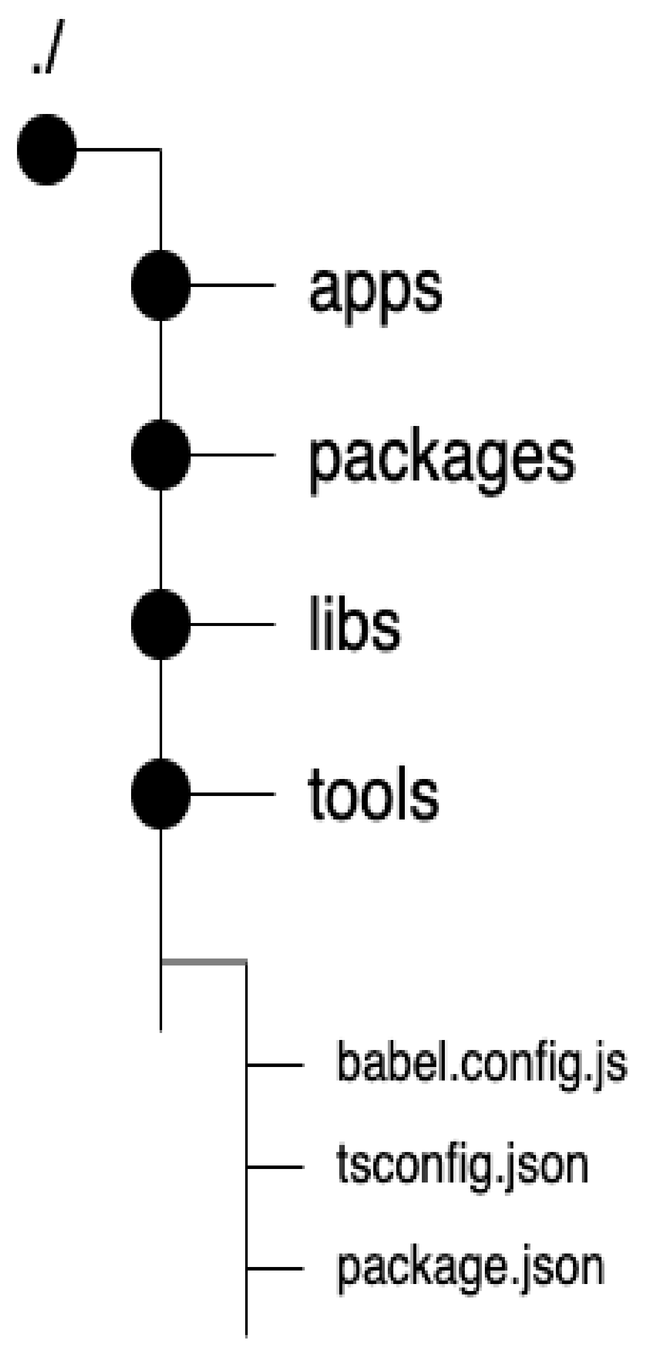
- Direct access to local files (libs / tools / etc…)
- Code ships do not need to be bulked
- Tool it away - Custom tools can improve the dev process
- Centralized revisions
- Release management

Structure - e.g.

Tenants, libs, tools, configs…

The folder structure may vary across the needs of the team and, it should be a strategical point to plan, as it can ease the integration process of new developers and the definition of standars.

- *Apps* - Should contain all the tenants which can or cannot be connected between them
- *Packages* - Standalone libs that need to be shipped outside of the monorepo context
- *Libs* - Internal code pieces that belong in the monorepo context
- *Tools* - This is a subjective part of the repo that should contain all the necessary tools to ease the work on this environment
- *Configs* - Configs are the guidelines center for each tenant, package or lib under this context but, in special cases, certain code pieces may obey to different rules if necessary



If I make a small change, do I need to ship all my code to production?

No...

I mean, if you really want to... 😅

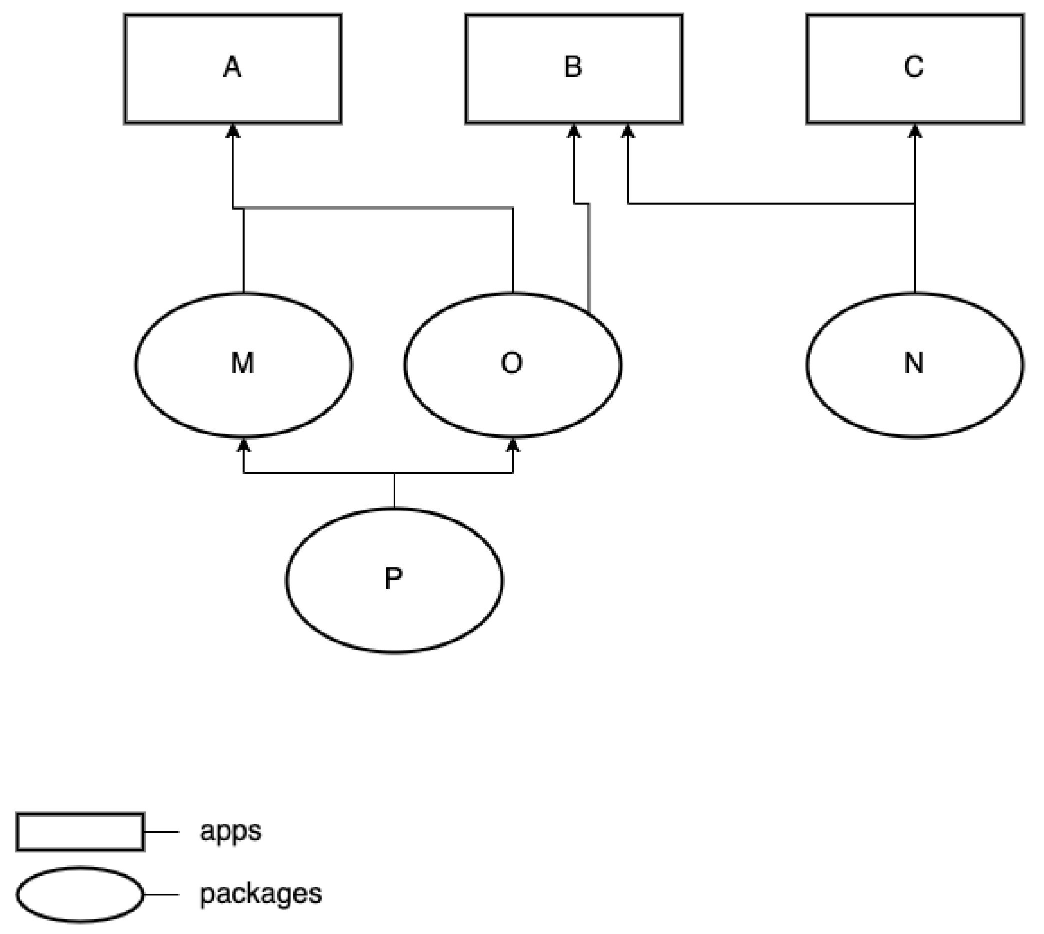
A repo works the way we want. In my opinion we should look at it like a tree

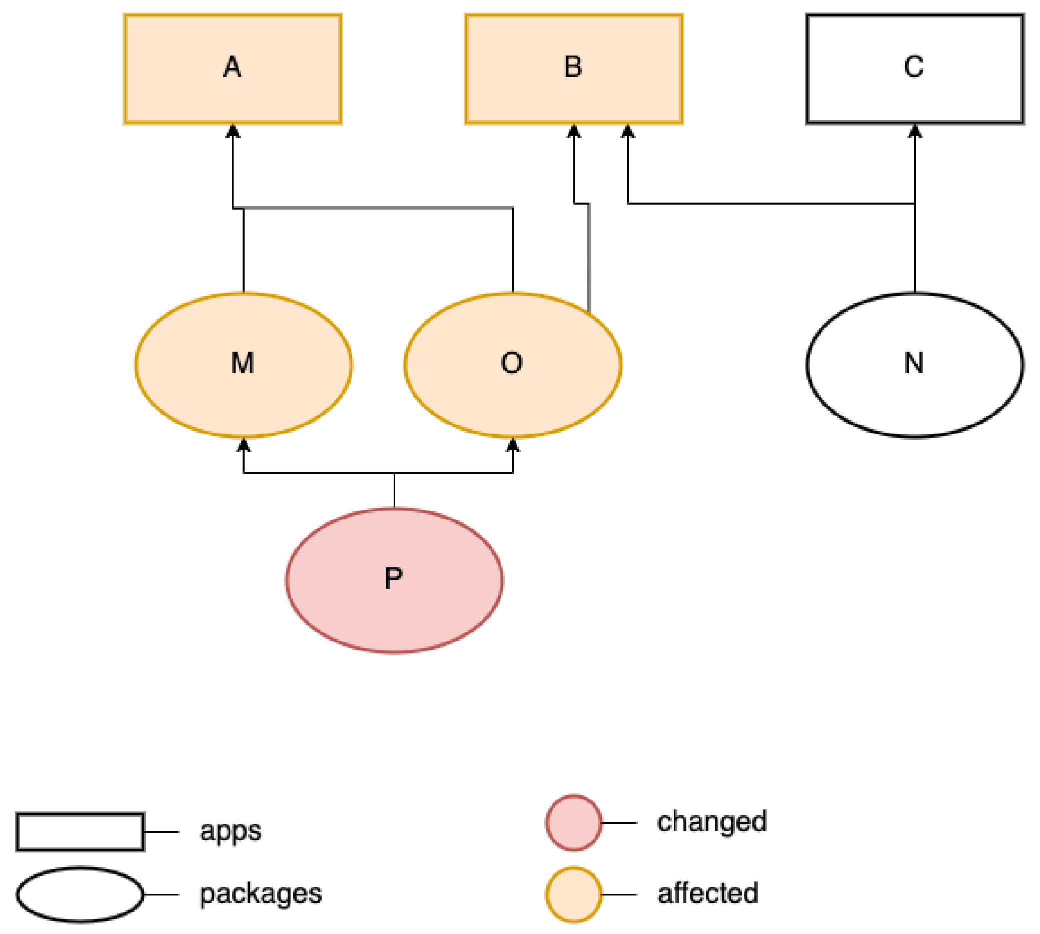
The tree

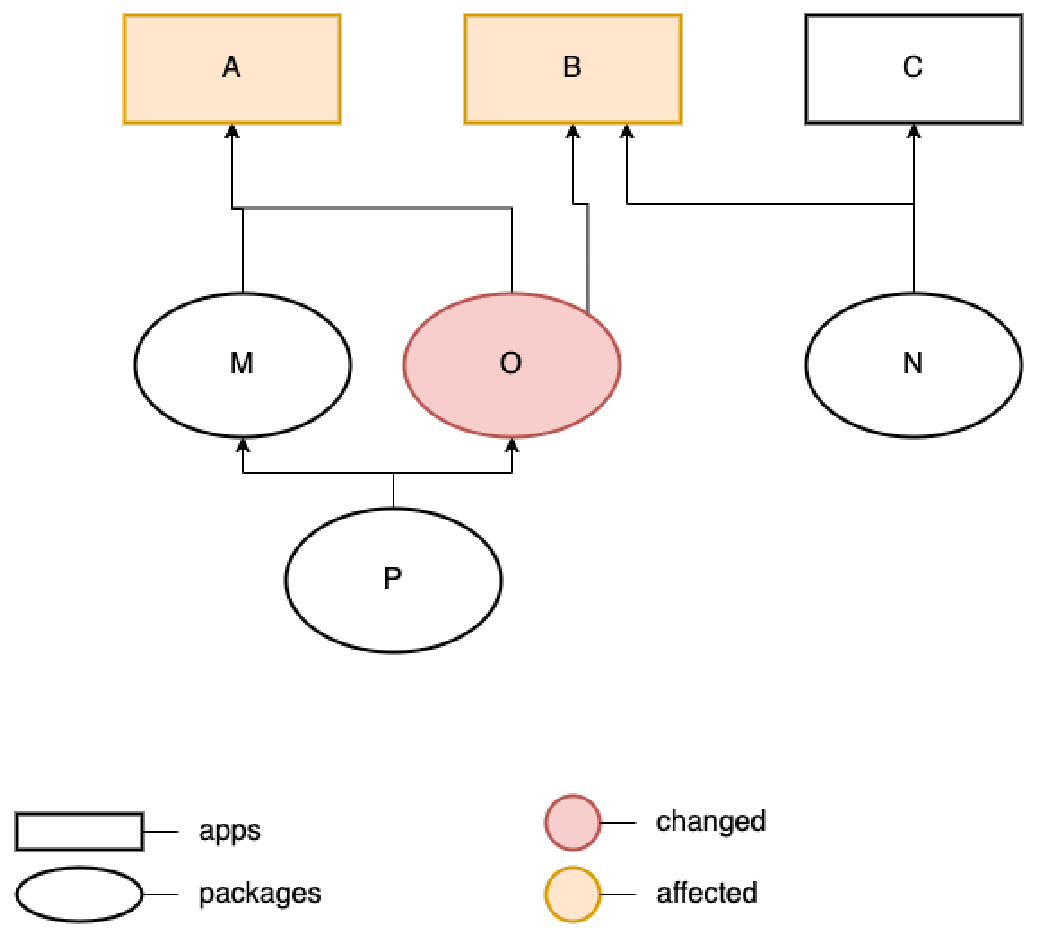
Touching code parts

Changed - code pieces that were effectively changed between two revisions

Affected - code pieces that depend on changed parts between two revisions







Tooling - one example

Being aware of these dependencies and other crucial information is not trivial

Tooling is in fact a big subjective word in this context, but it aims to solve every problem that may result from the monorepo approach.

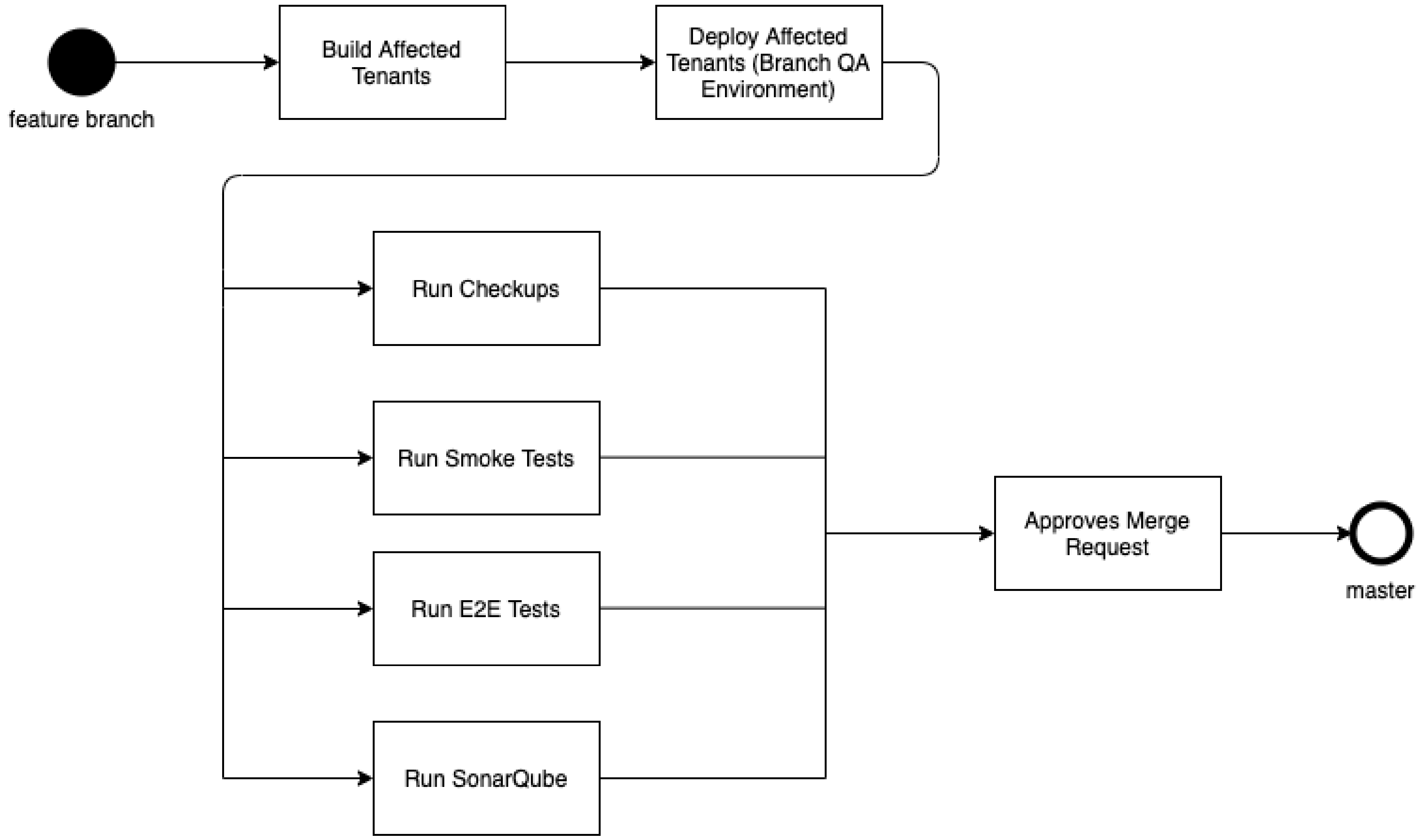
Sadly, there's not much options in the wild and those may not even be a good fit for your solution…

So, how do we know what was changed and affected?

- Stabilize your code
- Map your workspace
- Generate a dependency graph and label your code pieces
- Tag stable revisions on strategical branches
- Check the file differences between the last stable revision and the most recent one
- Find the affected pieces in your dependency graph
- Test and ship them? (create a logic process to fit your team and tech stack needs)

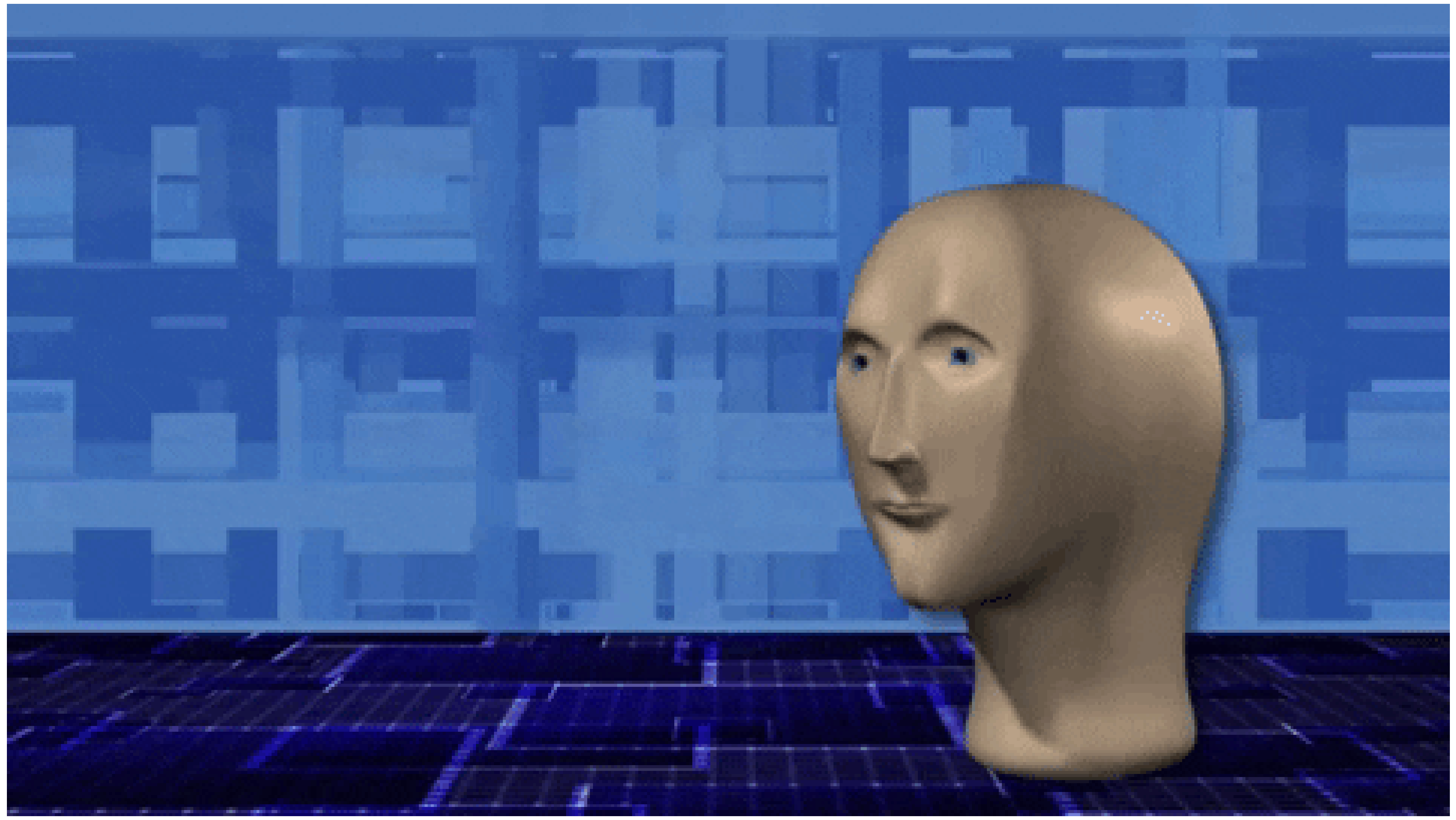
CI Flow - one example

- Keep the main branch stable
- *Always* test before merging
- Have rules
- Keep the environments up to date



Do you need a monorepo?

Probably not…



Thank you