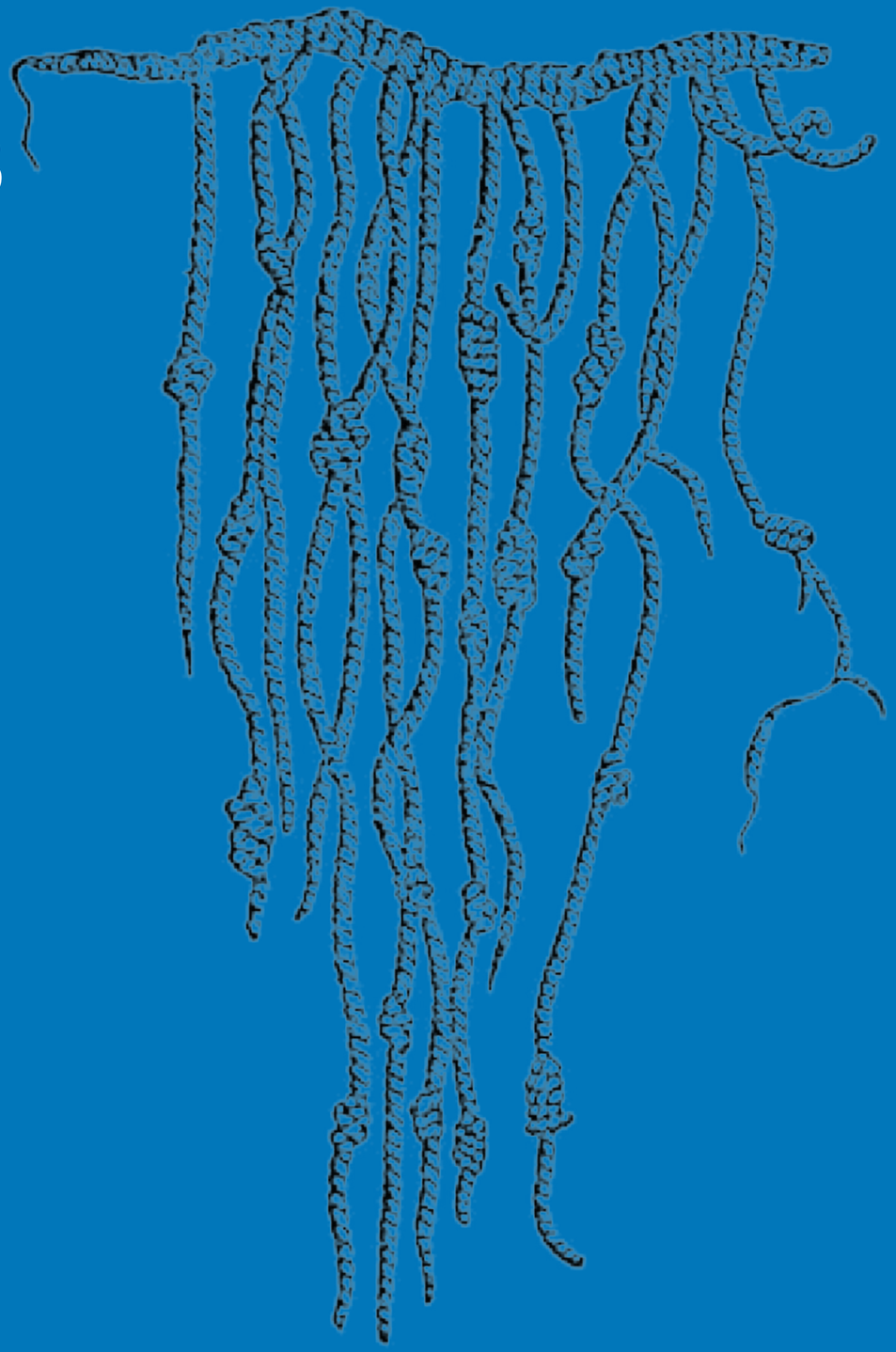
Untangling Deployments

From branching scheme to free-standing artefact layer



Mark Meyer - DevOps & CloudNative
MaibornWolff GmbH - October 29, 2024

What to expect

Answers? More questions!

This talks topics:

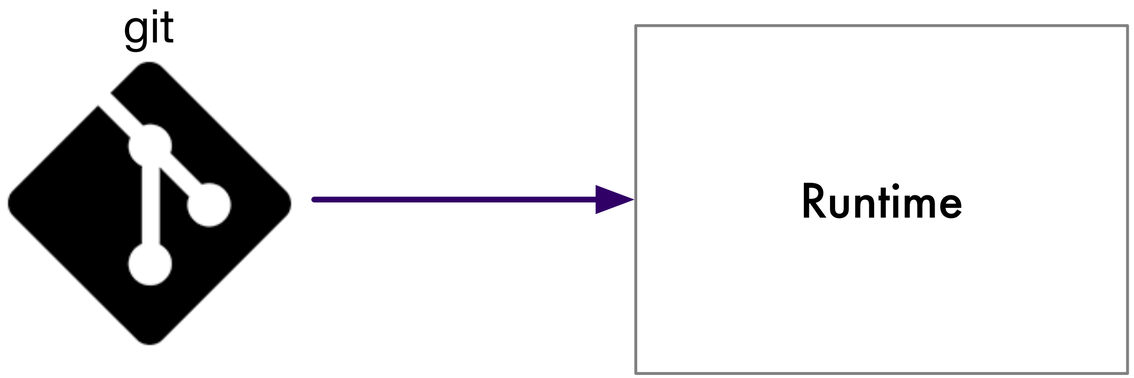
- Canto I: How everyone deployed 7 years ago
- Canto II: How to decouple deployment policy from mechanism
- Canto III: The free-standing artefact layer
- Relationship to team structure

I'll try to keep the focus on a conceptual level. I'll share ideas and show how it can be implemented.

Canto I:
The branching scheme

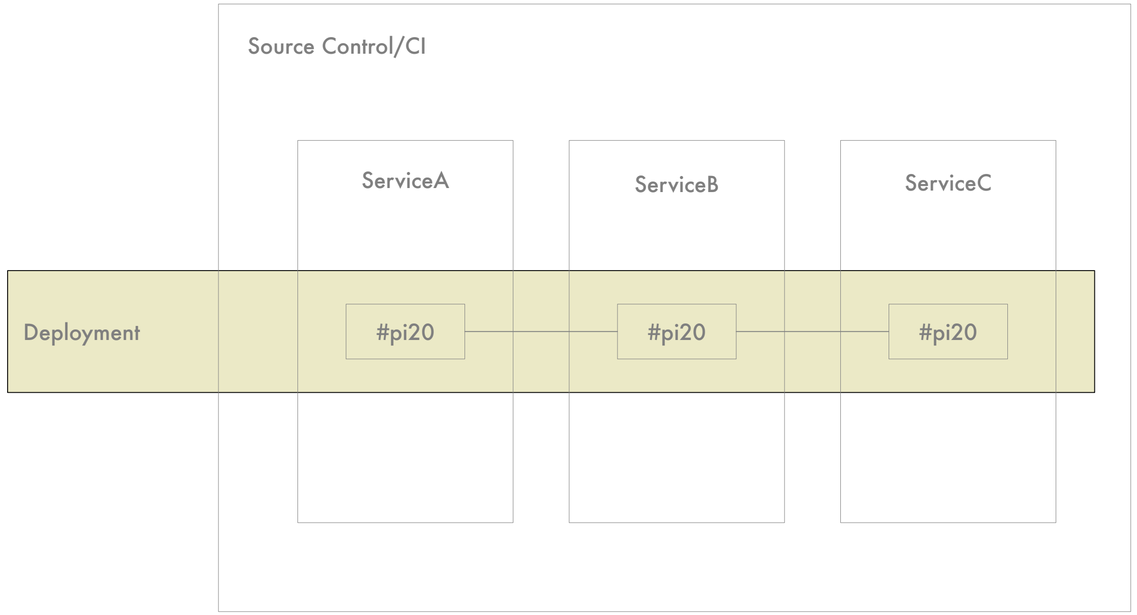
Deployment “The classic”

Deployment

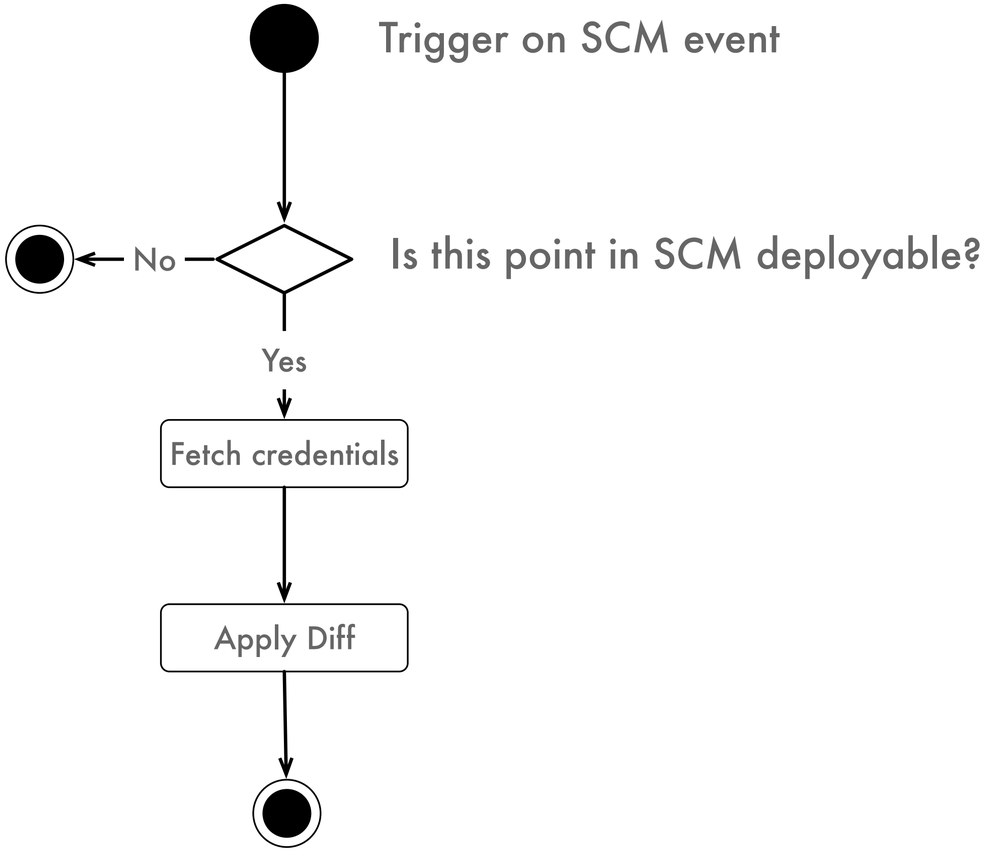


Classic “DevOps” deployment

State of the art ~7 years go



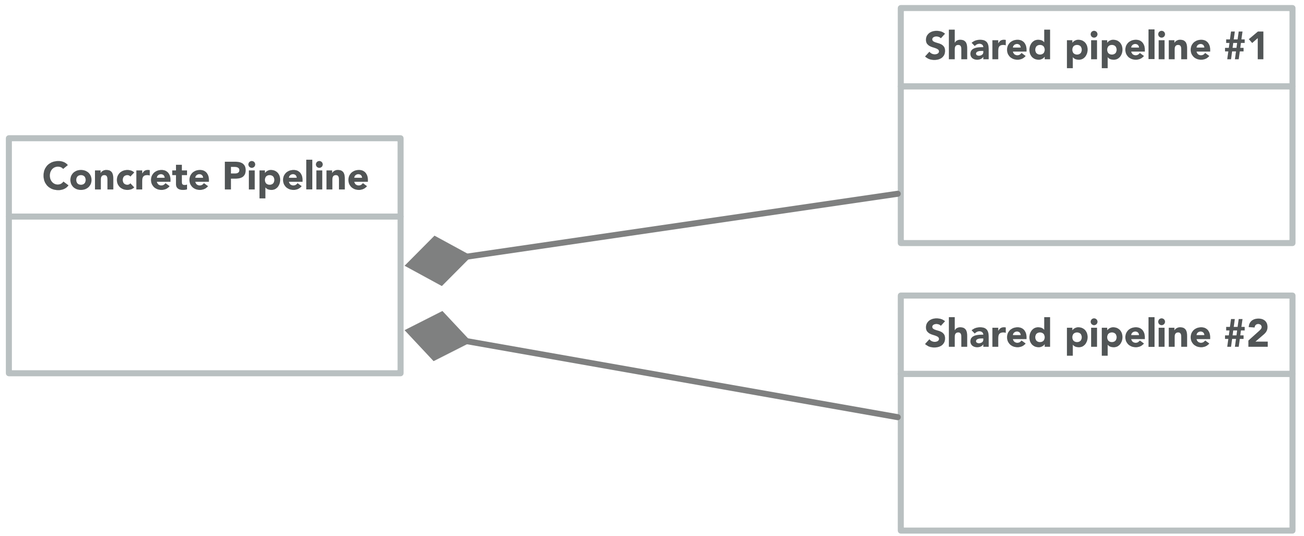
Global View



Repo View

Classic “DevOps” deployment

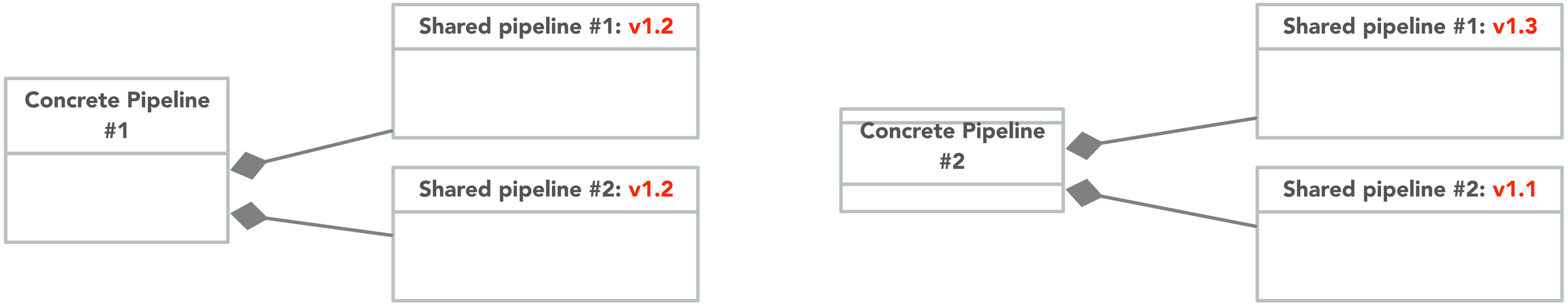
Versioning the deployment mechanism - Gitlab (1)



- Implemented with shared pipelines
- Sharing is good in theory, “include” creates problems though

Classic “DevOps” deployment

Versioning the deployment mechanism - Gitlab (2)



- Explicit control over step sequencing
- Implicit information passing/no fixed interface contract between includes
- Promotes unnecessary complexity

Classic “DevOps” deployment

Why people move away

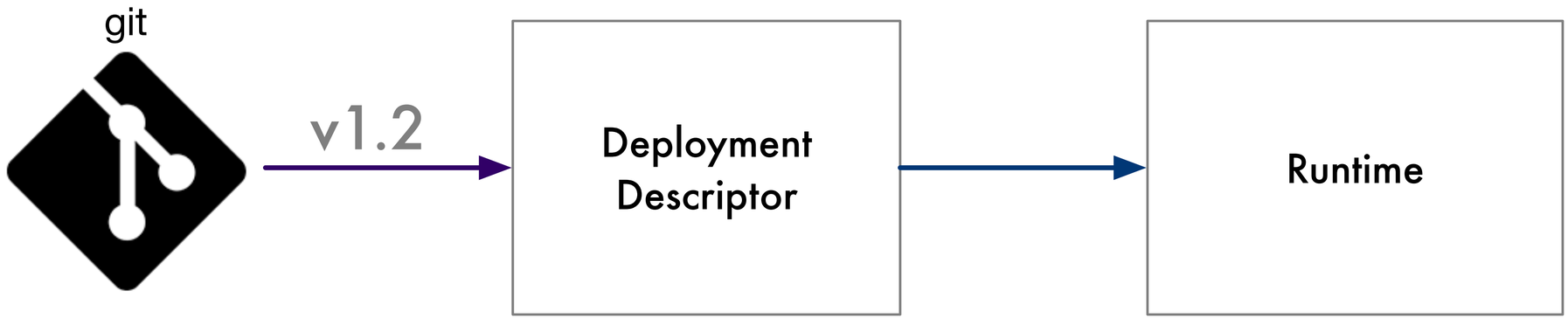
- Deployment decision encoded in the SCM identifier (branch, tag)
- High-level SCM identifier, usually related to product increment or release
- Usually this SCM identifier is global across all deployment units

- Result: you choice of software lifecycle ends up hardcoded in some bash script inside every artefact repository, possibly in multiple different ways

Canto II:
Decoupling deployment policy

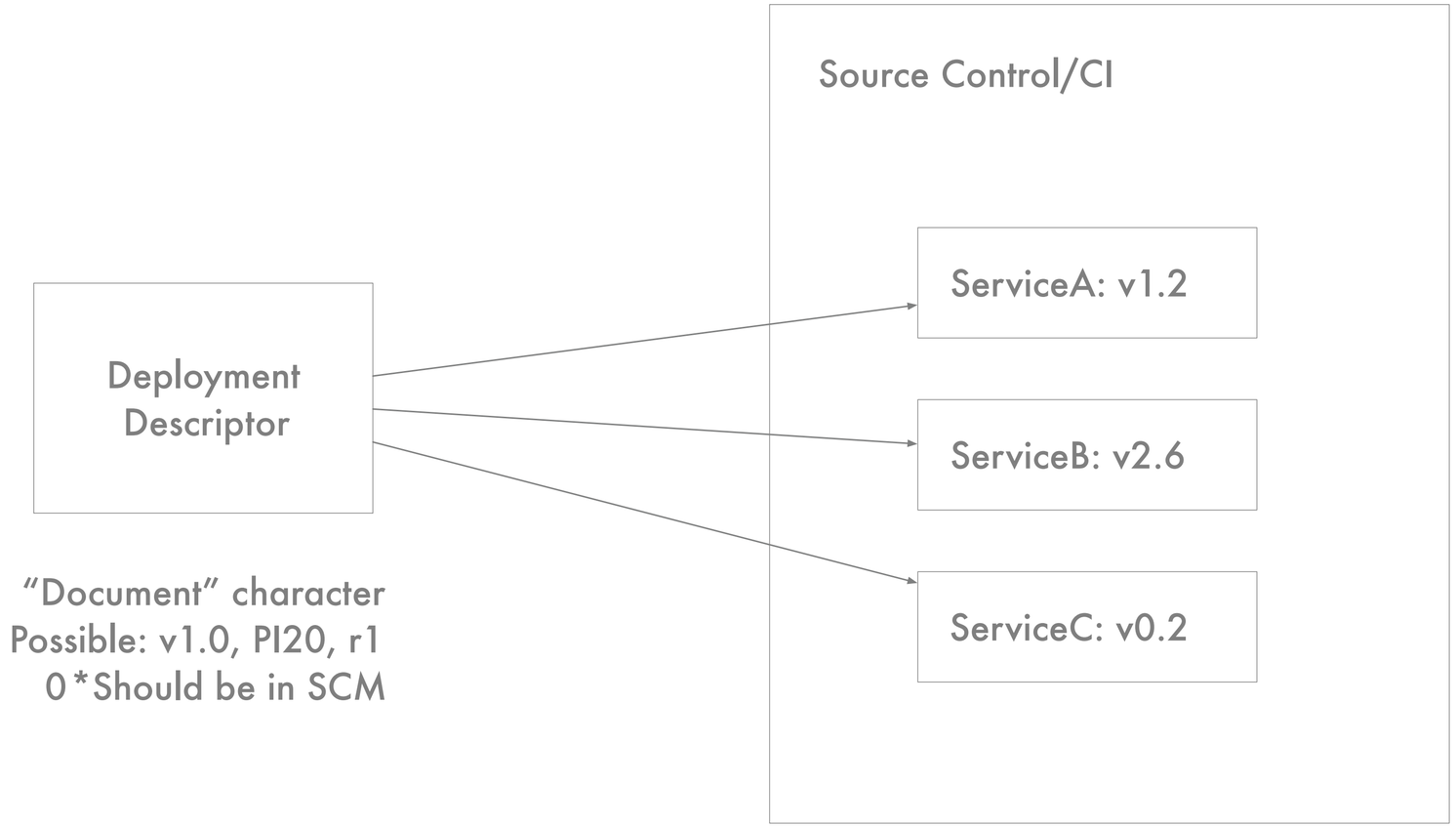
Deployment “The Standard”

Deployment



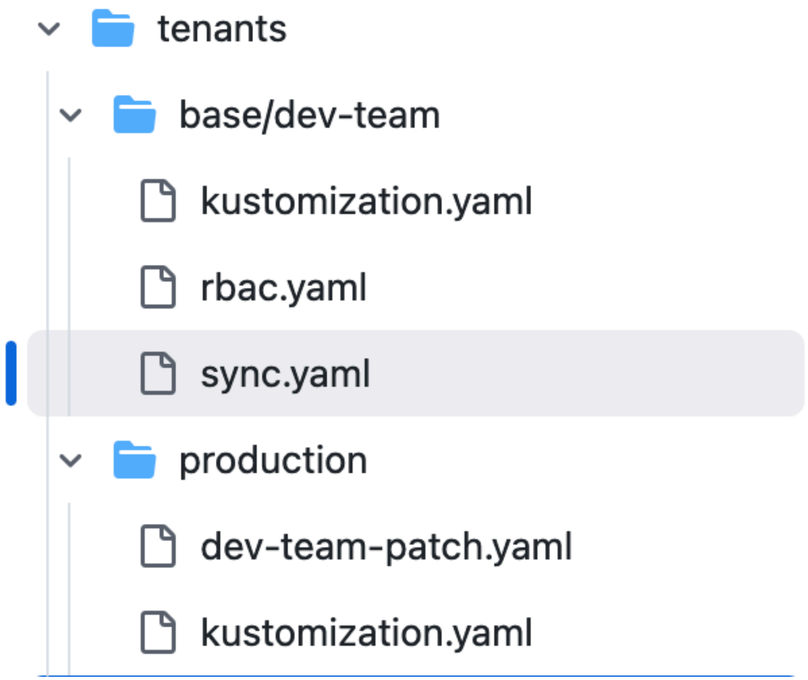
The deployment descriptor

Decoupling policy from mechanism



The deployment descriptor

How Flux does it - The platform admin repo




```yaml
apiVersion: source.toolkit.fluxcd.io/v1
kind: GitRepository
metadata:
  name: dev-team
  namespace: apps
spec:
  interval: 1m
  url: https://github.com/fluxcd/flux2-multi-tenancy
  ref:
    branch: dev-team
---
apiVersion: kustomize.toolkit.fluxcd.io/v1
kind: Kustomization
metadata:
  name: dev-team
  namespace: apps
spec:
  serviceAccountName: dev-team
  interval: 5m
  sourceRef:
    kind: GitRepository
    name: dev-team
  prune: true
```

Taken from: gh.com/fluxcd/flux2-multi-tenancy

The deployment descriptor

Can you do this with the tool du jour?

- Yes, trust me.
- Two implementation alternatives:
  - Yaml descriptor that defines revisions to trigger pipelines on
  - Pull source from SCM and run tooling like Terragrunt
- It totally can be done, independent of the tooling/stack you use
- It’s about concepts: In this case separating policy from mechanism

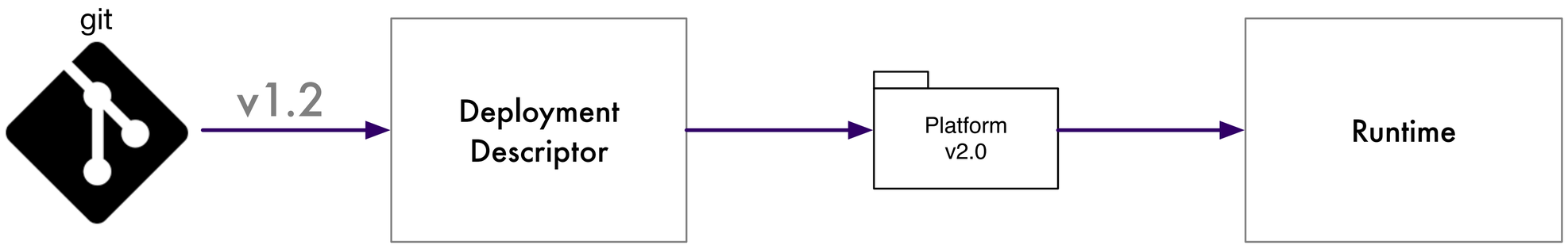
Result:

Deployments are in CI.
Development planning is in Jira.

Canto III:
The free-standing artefact layer

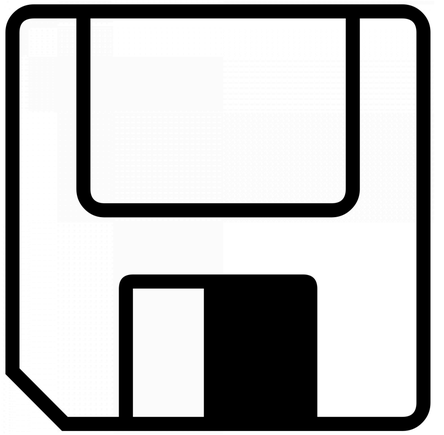
Deployment “The Decoupled”

Deployment



The artefact layer

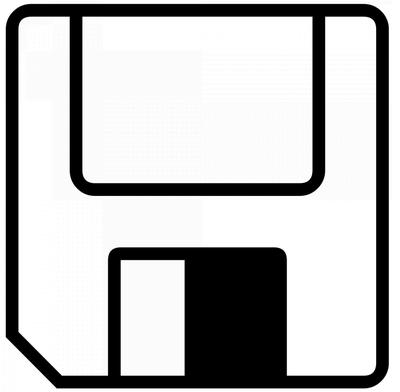
What does “free standing” mean?



- After factoring out the policy, we’re left with:
  - The infra code needs to be online to deploy
  - The infra code can only be deployed from a central CI system
- My proposition:
  - The infra code needs to be packaged like an application

The artefact layer

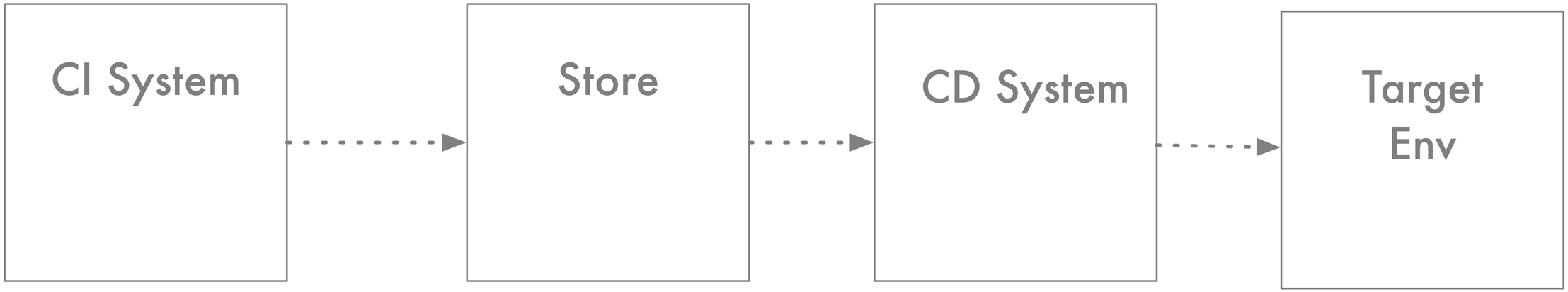
Bundling infra code



- Create an infra bundle:
  - It’s named for the deployment unit with a version, e.g. “hello_1.2.3.zip”
  - It contains the deployment unit (e.g. the “hello" service)
  - It contains all the dependent infra modules
  - Tools: Maven, Bazel
- Place the infra bundle into an artefact store
- At deploy time: pull the artefact and execute/apply the code

The artefact layer

Bundling infra code - system structure



Examples:
- Gtilab
- Github
- Codeberg

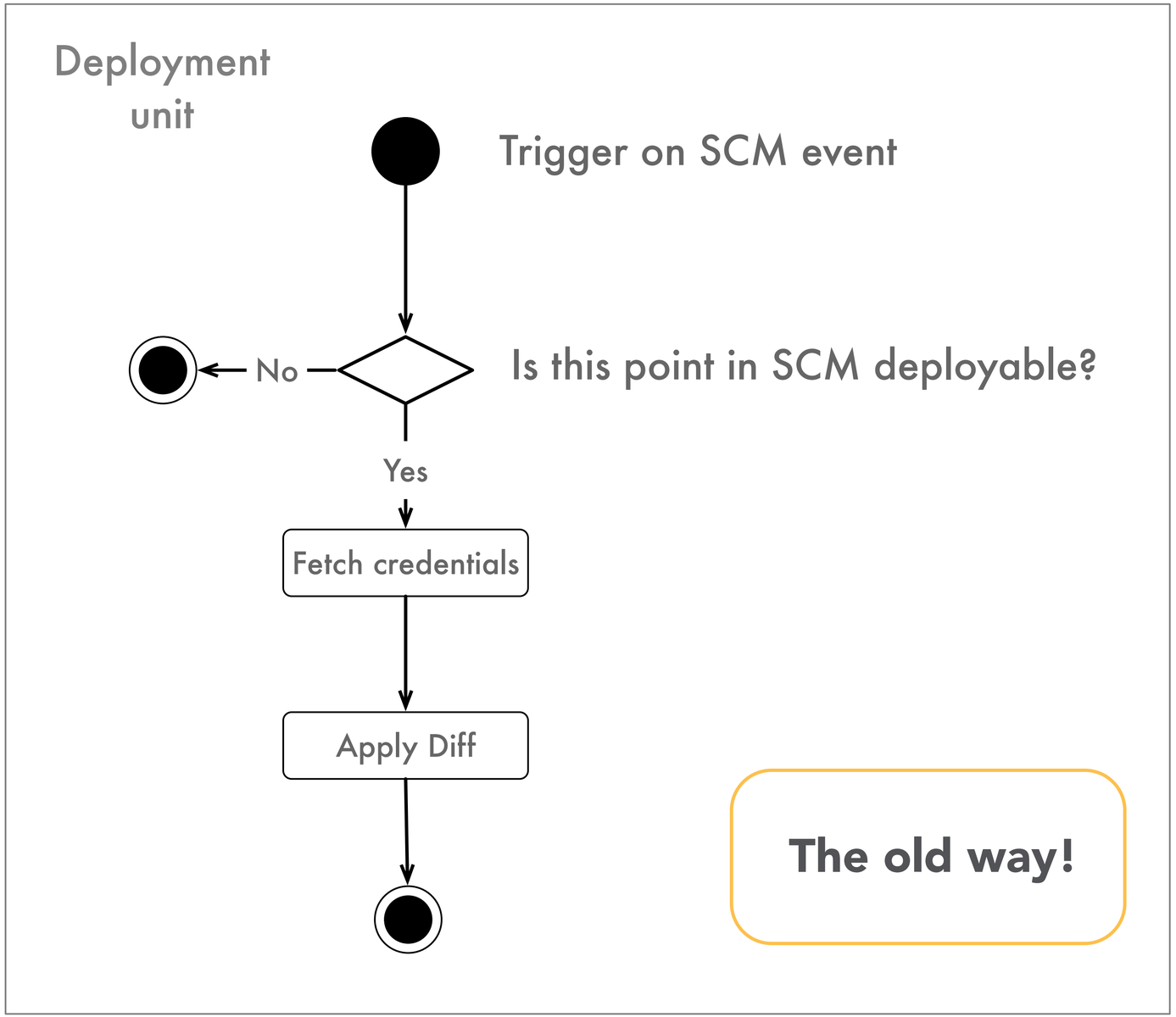
Examples:
- S3
- OCI Artefact
- An SD card

Examples:
- ArgoCD
- Flux
- CodeCommit
- Raspberry Pi

Examples:
- VMware
- AWS
- Clay Tablets

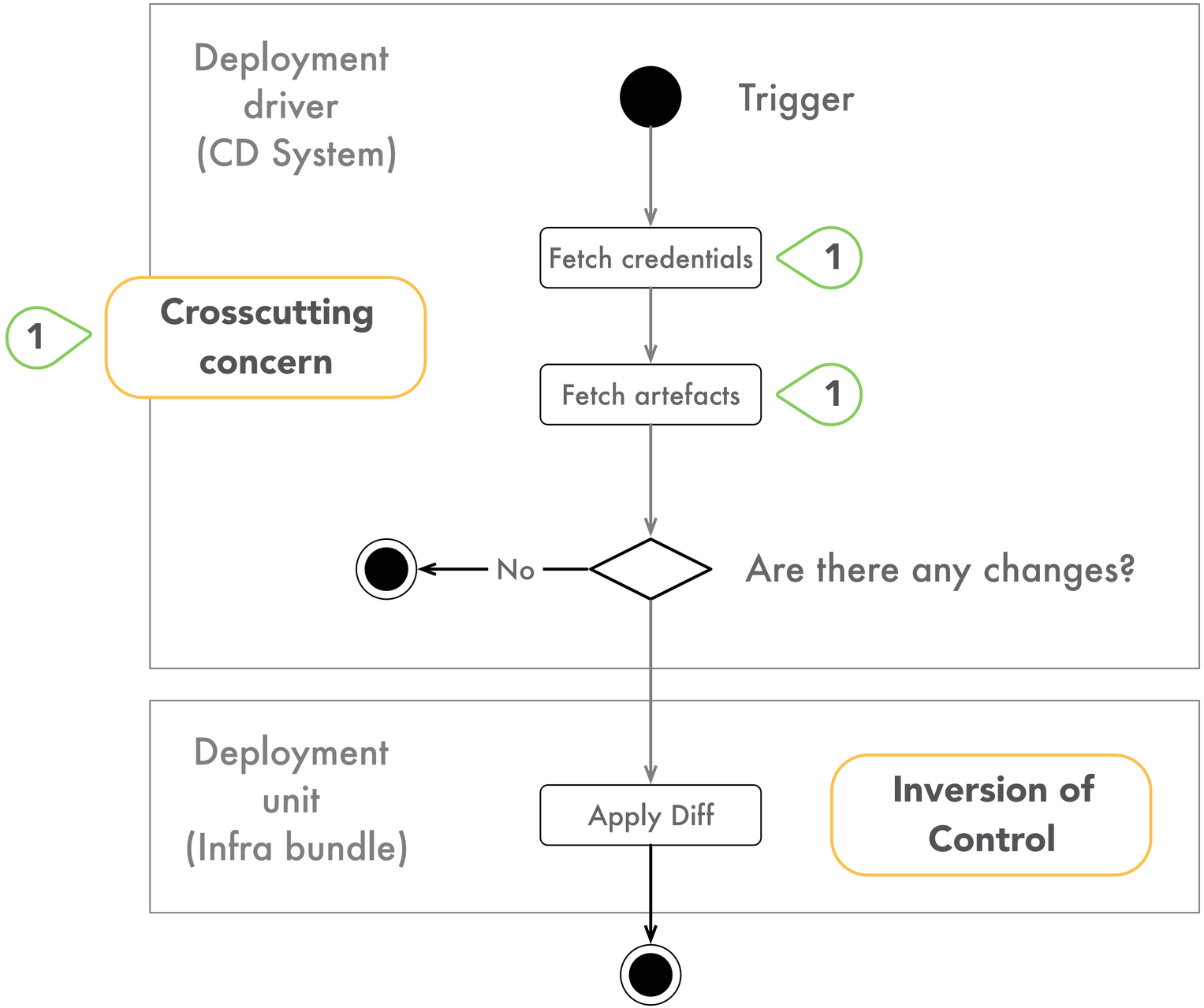
The artefact layer

Small reminder from before



The artefact layer

Deployment control flow



Result:

Artefacts can be deployed independently from the time and the place they have been developed and tested.

The human factor

The organisation (1)

How does this interact with team topologies?

- Scheme (1) may be for the “founding team” (i.e. just a single stream-aligned team)
- Setup (2) seems to be the new standard
- Setup (3) is for a diversified team:
  - does not presuppose e.g. a platform boundary
  - works equally well whether monorepo or not
  - High degree of flexibility for growing the organisation around it and refining structures inside

The organisation (2)

What does this mean for the “complex subsystem team”?

- Not everyone can wear every hat at the same time with the same quality
- You can (should!) make service code version an open attribute of infra code
- This does not require any non-infra roles to wade through layers of terraform

- Infra Sandboxes are quickly built and torn down, since no central configuration is required to make them happen, making for happy specialists (and generalists too)

Thank you!

Mark Meyer - DevOps & CloudNative
MaibornWolff GmbH - October 29, 2024