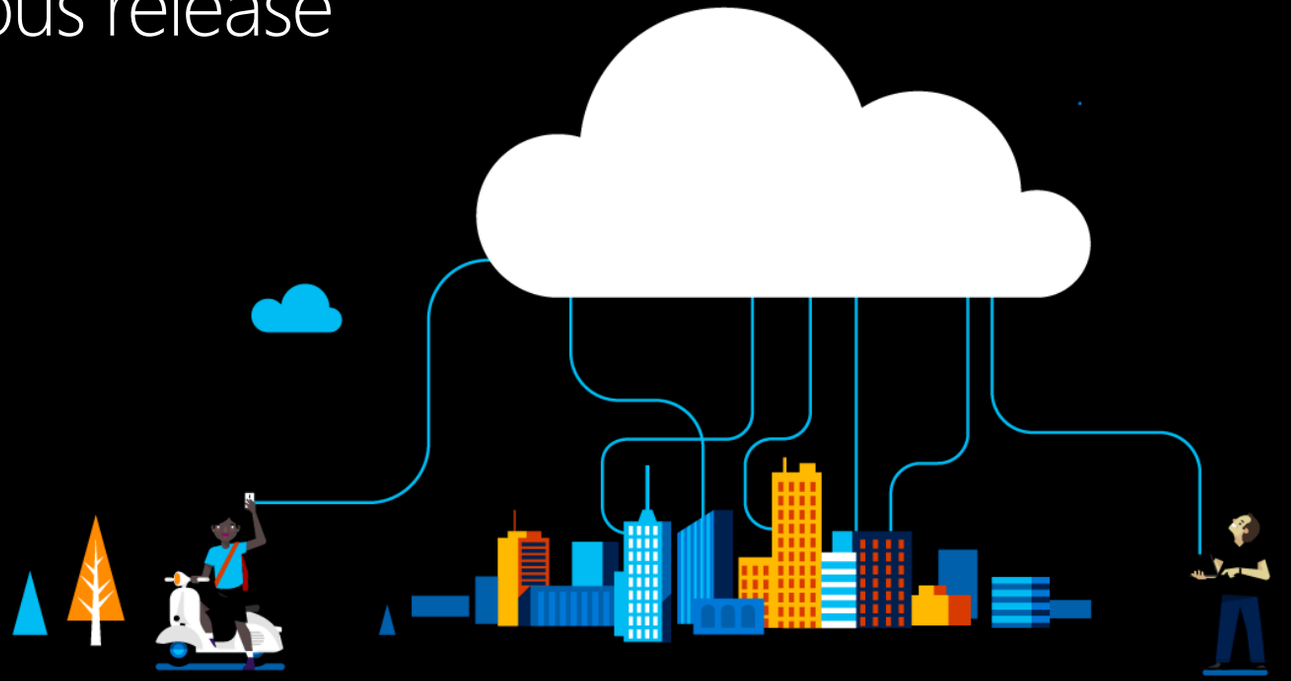
Flagging your features

a DevOps approach to continuous release

Alex Thissen
Cloud architect

@alexthissen



DevOops

Xpirit

Think ahead. Act now.

MVP Microsoft® Most Valuable Professional

Practicing DevOps

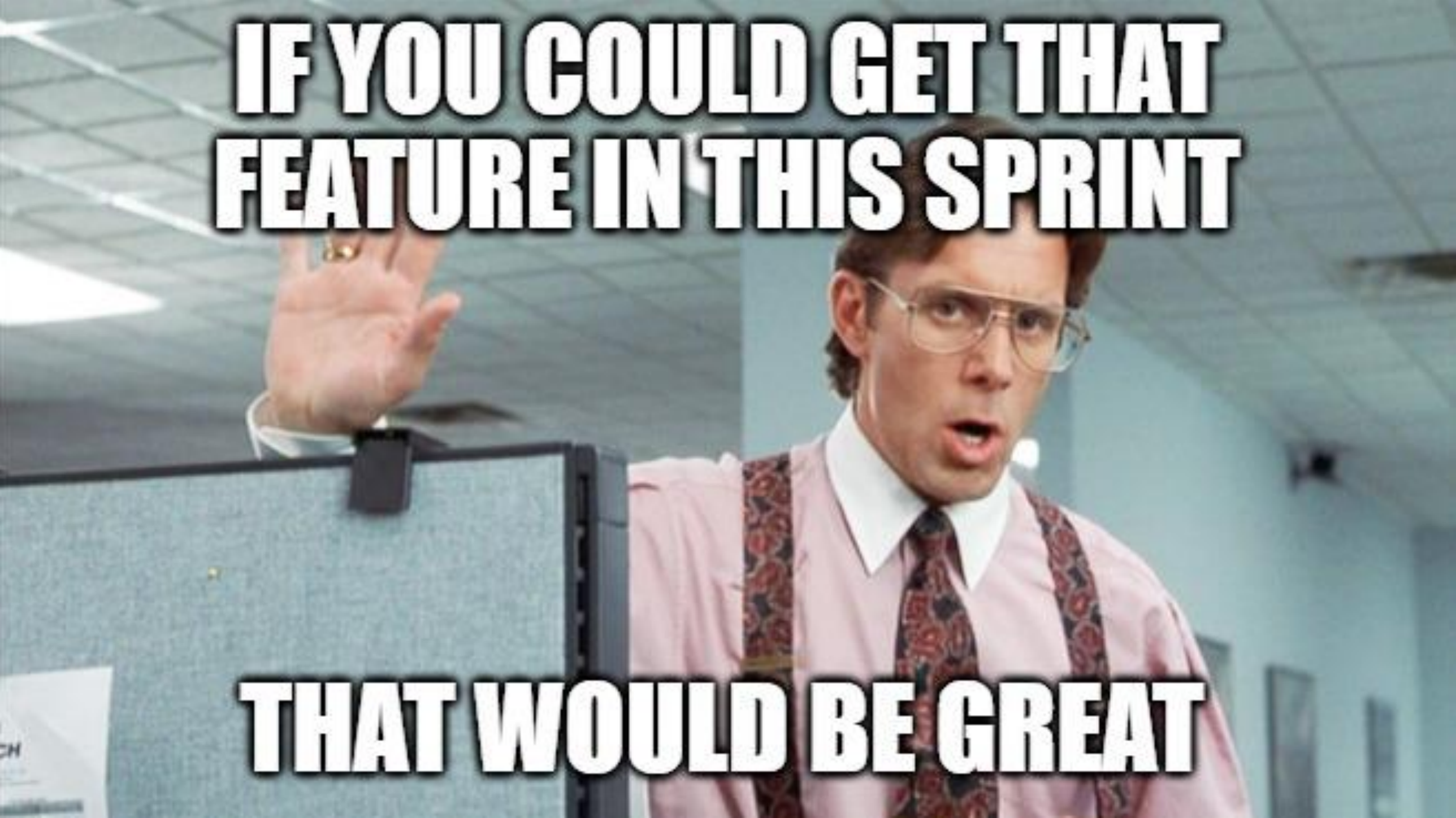
Release multiple times per day

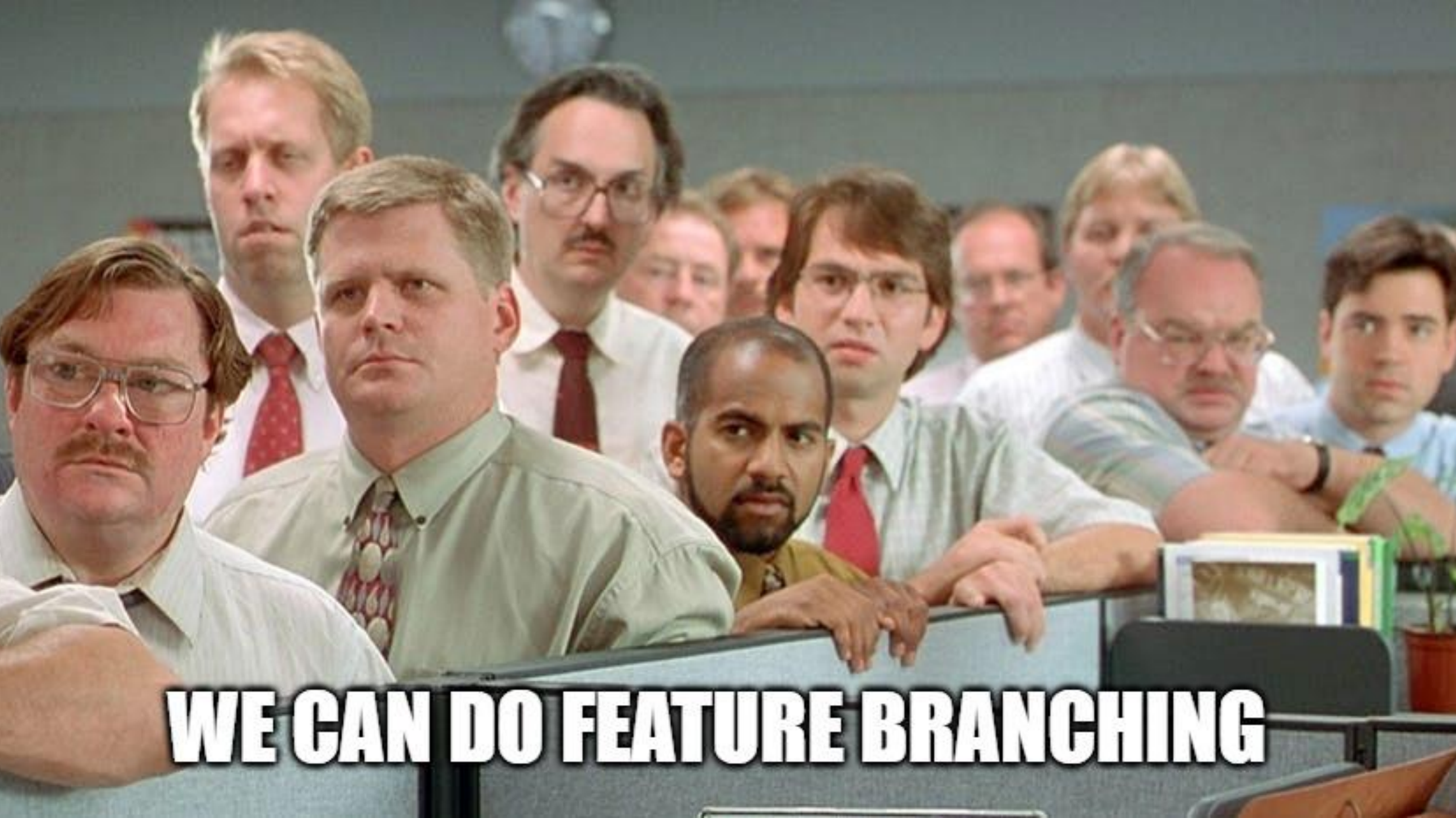
Move fast, safe and forward only

Continuously deploy to production

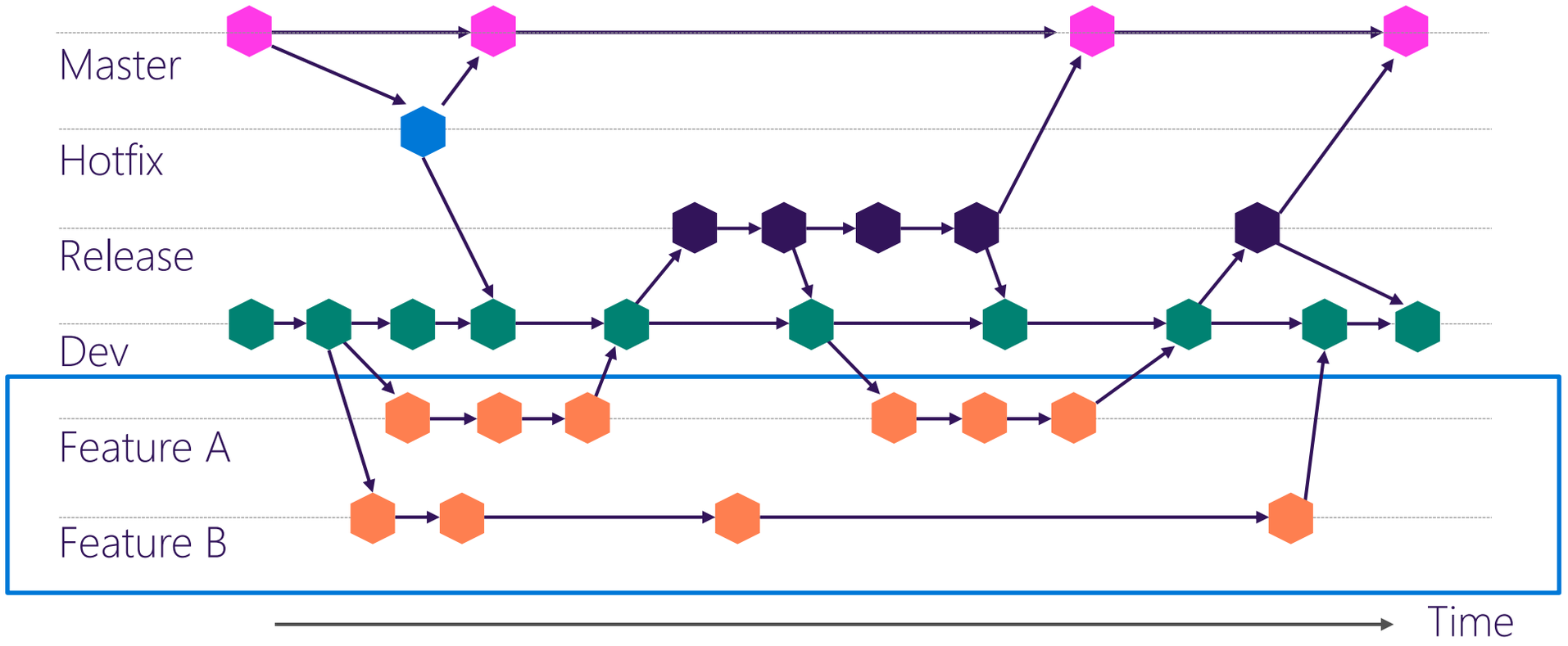
Never roll back in production

Fine-tune the user experience in production


IF YOU COULD GET THAT FEATURE IN THIS SPRINT
THAT WOULD BE GREAT


WE CAN DO FEATURE BRANCHING

Feature branching strategies



New features are created in a separate branch

Reality of feature branches

Team uses long lived branches

Even for feature branches

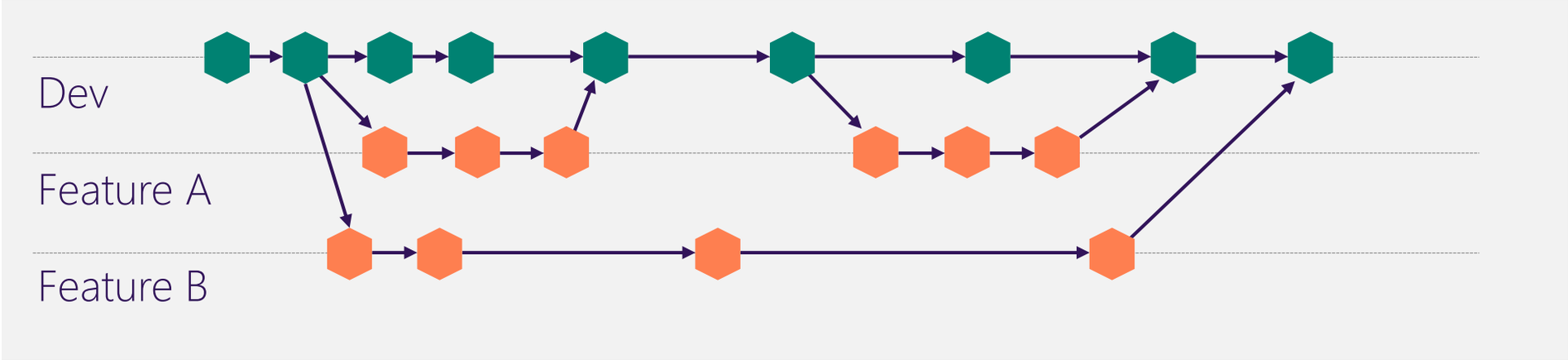
Much like waterfall

Harder to merge

Death marches at end of the sprint or project

Feature can only be released when complete

From feature branches to code branches




```
if (feature.IsEnabled("NewSearchPage"))
{
  // New feature implementation
}
```

Features in a different way

Back to “trunk based development”

Push functionality when and to whom you want

Decouple release from deployment

Expand or “rollback“ without redeployment

Why should you want feature flags?

Code branch management

Instead of repository branches

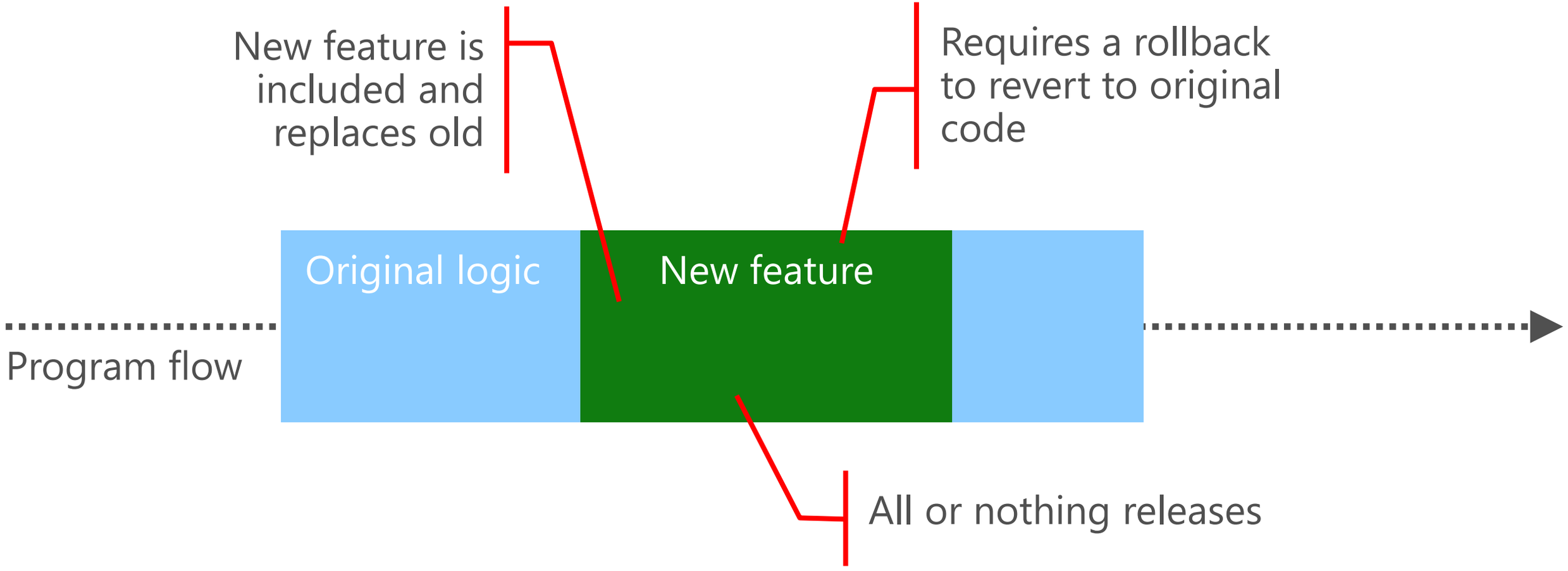
Simple form of switching functionality on and off

Other reasons include

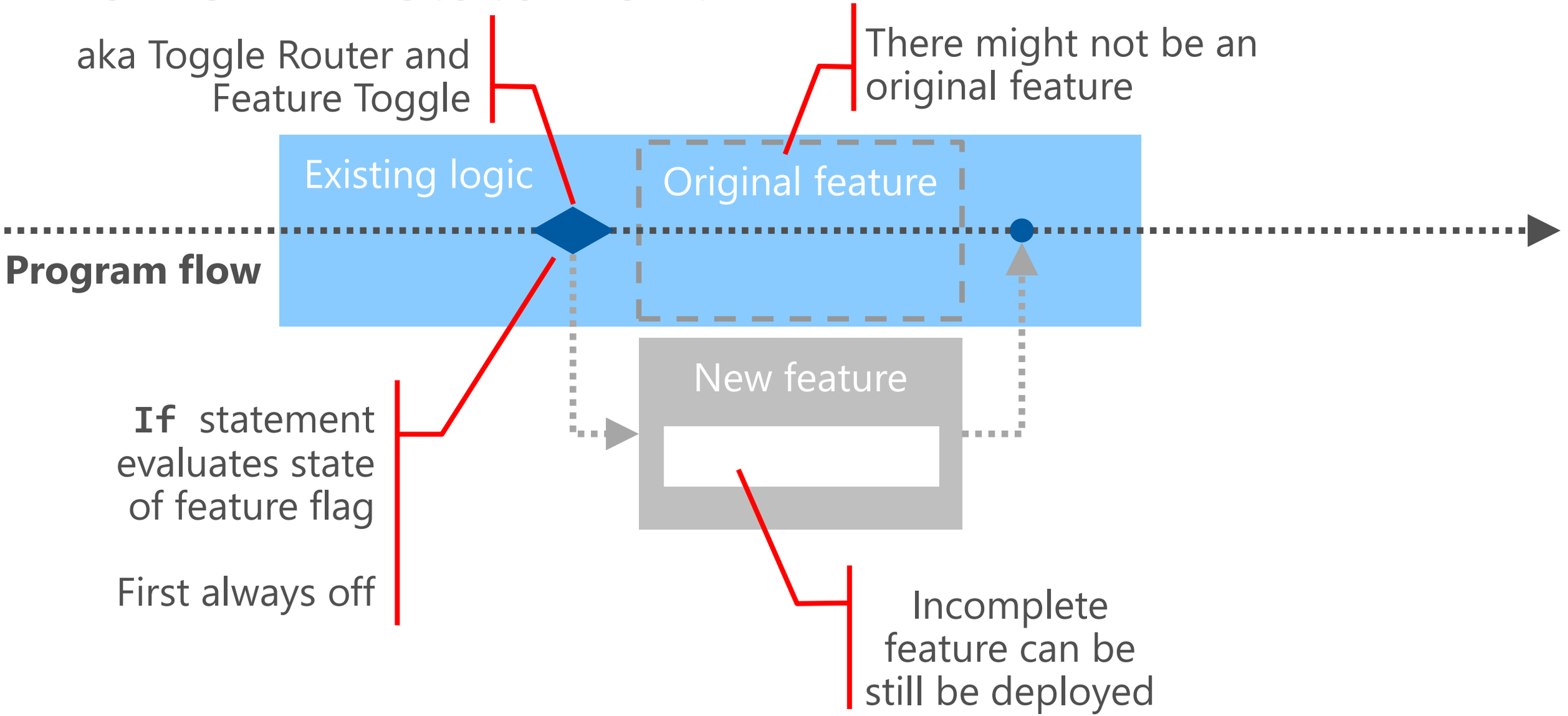
- Testing in production
- Flighting
- Instant kill switch
- Selective activation for user bases

Decoupling deployment from release

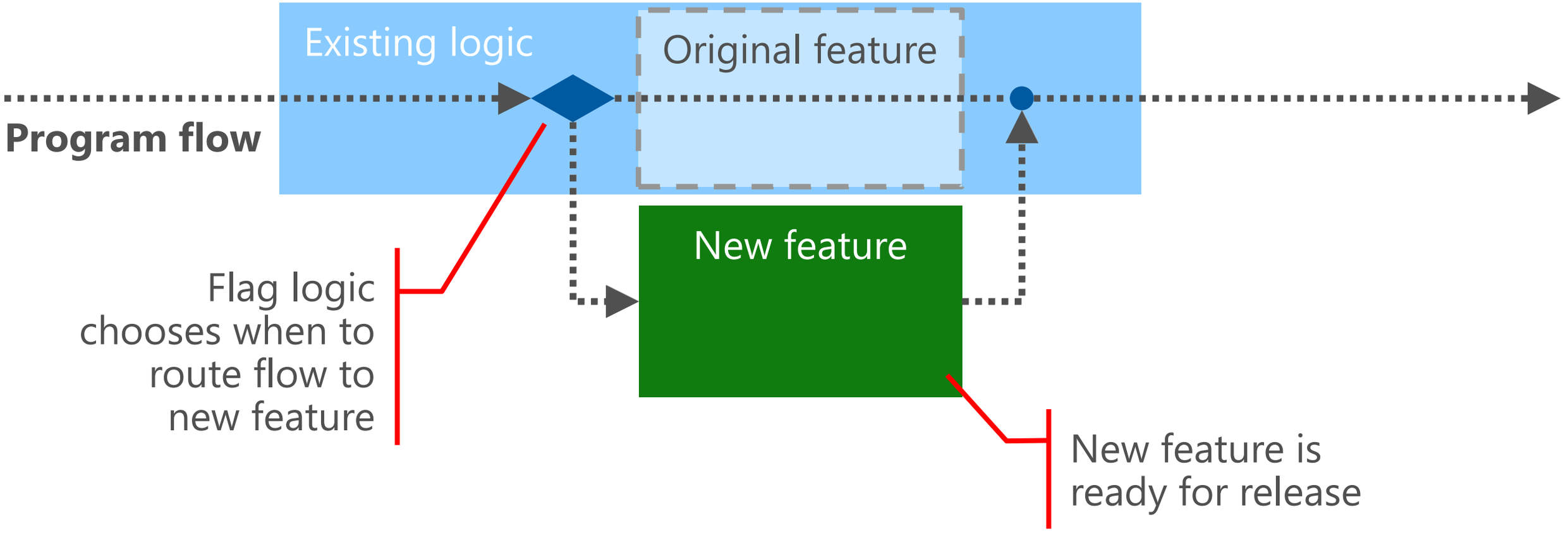
Original approach for a new feature



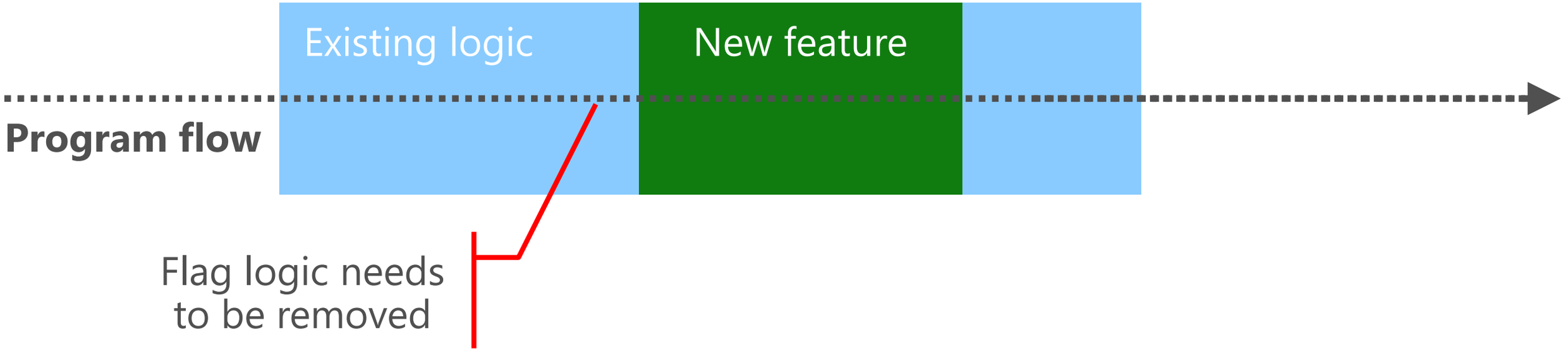
The new ‘if’ statement



The new ‘if’ statement



The new ‘if’ statement



Flags are released three times

1. Once as code (deploy)
2. Once as configuration (release)
3. Once to remove

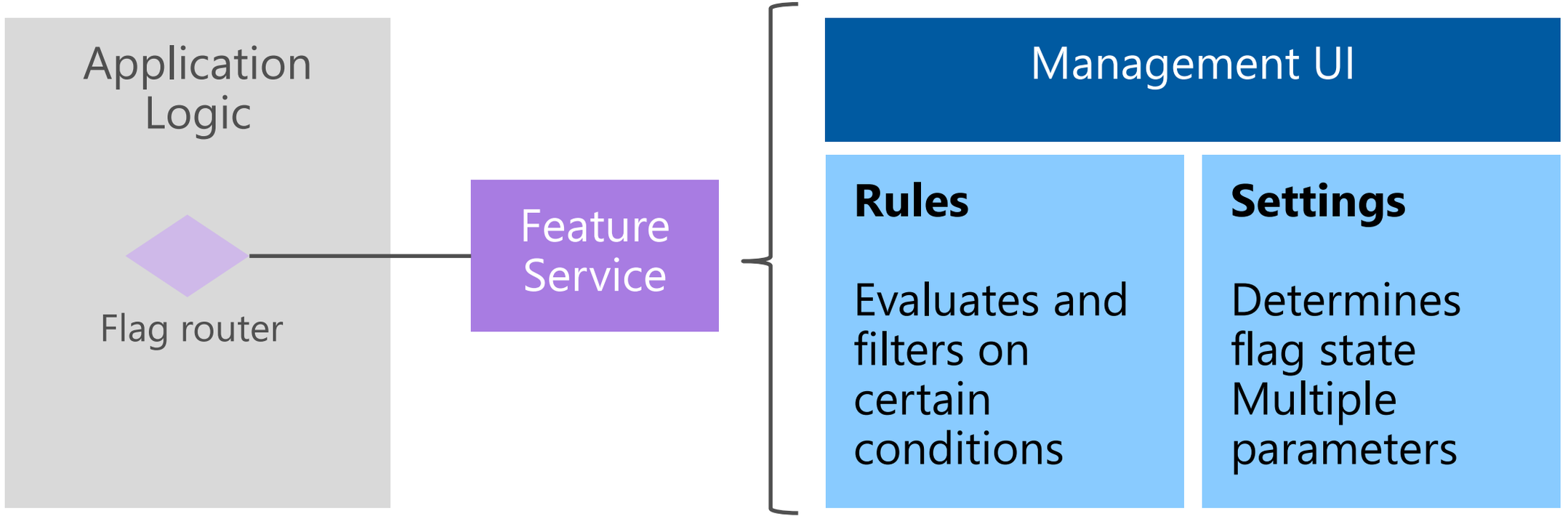
Feature flag categories

Release	Operations	Experiment	Permission
• Deploy incomplete code • Time release	• Performance related • Manual circuit breakers	• Testing scenarios • Dynamic	• Targeted users • Defining cohorts
Short-lived	Short to long-lived	Short lived	User or request based
• On • Off • Percentage	• On/Off Kill-switch	• Percentage • Time Window	• Claims • Cookies • Tenant

Feature management

Rules and settings for evaluation in flag router

Might be more intricate than binary state



Feature context

Named key-value pairs

Value might be set of parameters for complex evaluation

Different per environment

Feature	Value
LeaderboardListLimit	On
APIv2	Parameters
BetaWebsite	Opt-in

Canary

Feature	Value
LeaderboardListLimit	On
APIv2	Off
BetaWebsite	Opt-in

Early Adopters

Feature	Value
LeaderboardListLimit	Off
APIv2	Off
BetaWebsite	Opt-in

General Availability

Ingredients for feature management

In process service for evaluation

Dynamic change during running

Filters for evaluation (on/off, custom)

Externalized storage of key/value pairs

Cache and refresh plus ability to vary per environment

Management plane

Tools and platforms for feature management

Software as a Service

LaunchDarkly

Bullet Train

CloudBees Rollout

Platform as a Service

Azure App Configuration

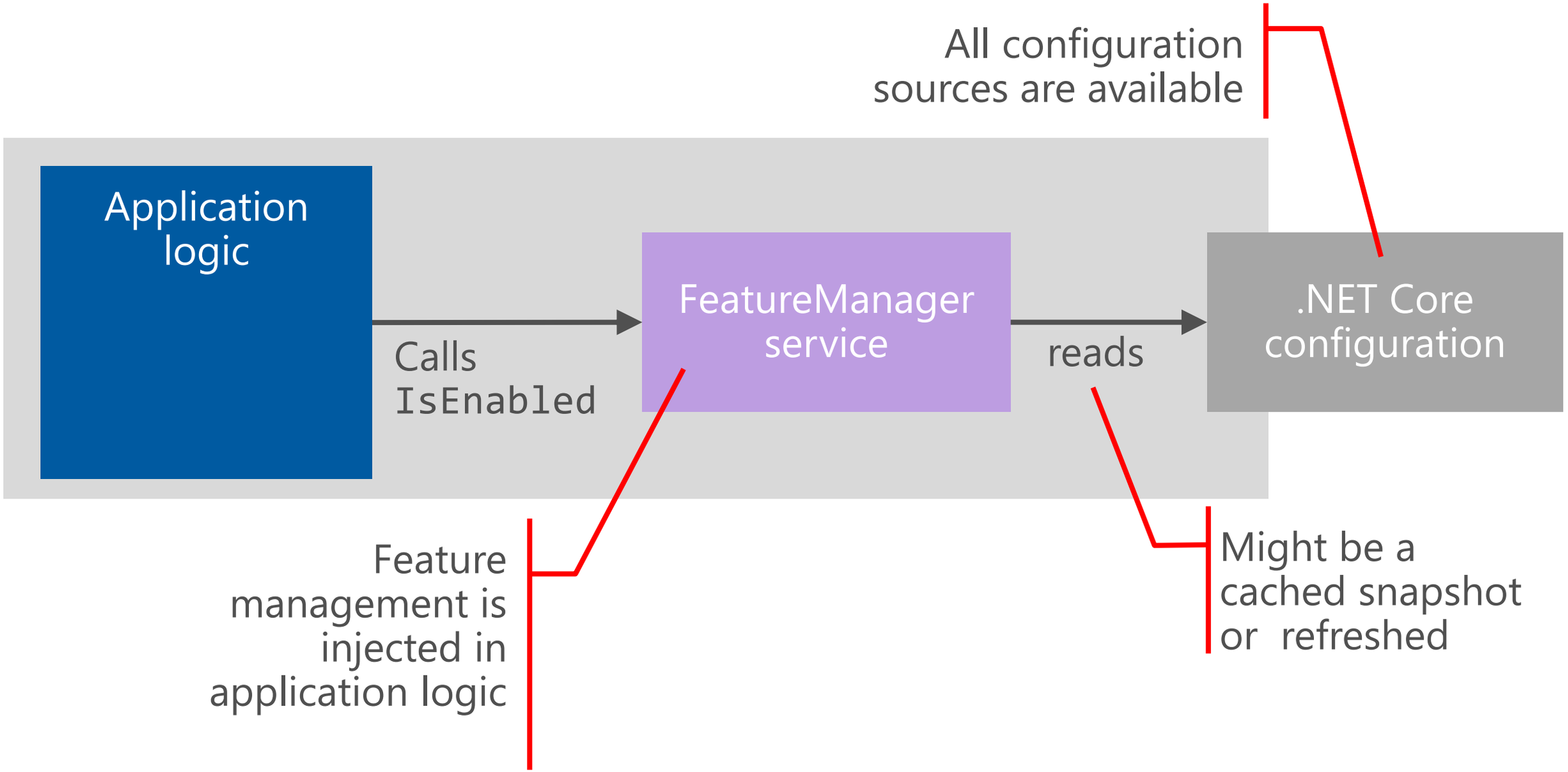
Application Frameworks

MOGGLES

Esquio

Microsoft .NET Core Feature Management
RimDev.AspNetCore.FeatureFlags
NFeature (GPL)
Feature Toggle (Apache 2.0)
Feature Switcher (Apache 2.0)
FlipIt (Apache 2.0)

.NET Core Feature Management architecture



Feature flags: .NET Core

Register service

```
services.AddFeatureManagement();
```

Inject where needed in code

Uses configuration values

Dynamic and snapshot evaluation

```
"FeatureManagement": {
   "Feature1": false,
   "Feature2": true,
   "Feature3": {
     "EnabledFor": [{
         "Name": "AlwaysOn"
       }
     ]
   },
   "Feature4": {
     "EnabledFor": [{
         "Name": "Microsoft.Percentage",
         "Parameters": {
           "Value": 50
         }
       }
     ]
   }
 }
```

Custom filters

Implementations of IFeatureFilter

```
public interface IFeatureFilter
{
  bool Evaluate(FeatureFilterEvaluationContext context);
}
```

Provided out of the box

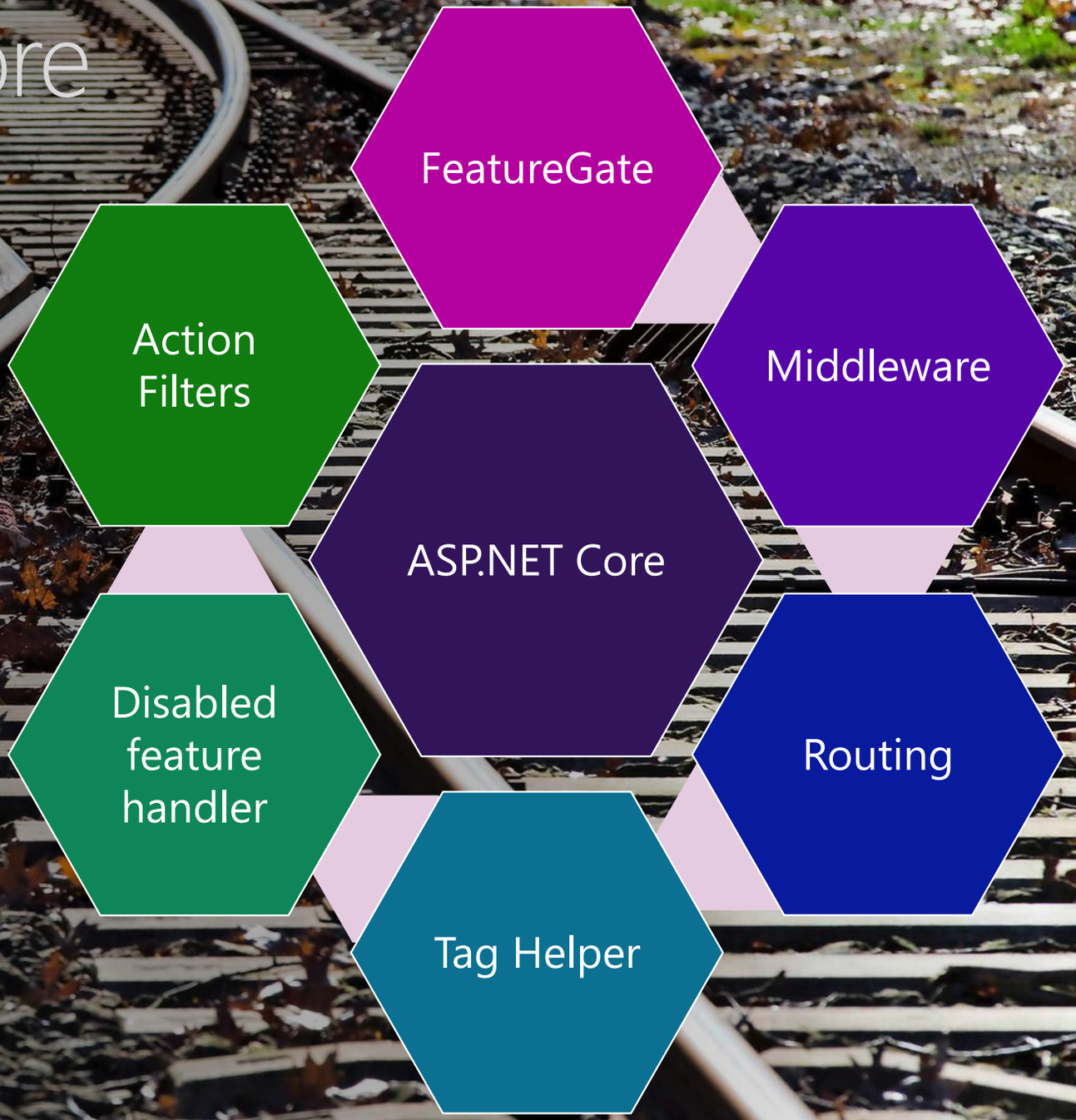
PercentageFilter

TimeWindow

Other filters:

User targeted, e.g. ClaimsFilter, CookieFilter, Browser

Features in ASP.NET Core



Deep integration into application framework

NuGet package

Microsoft.FeatureManagement.AspNetCore

Feature management in Azure

Azure App Configuration

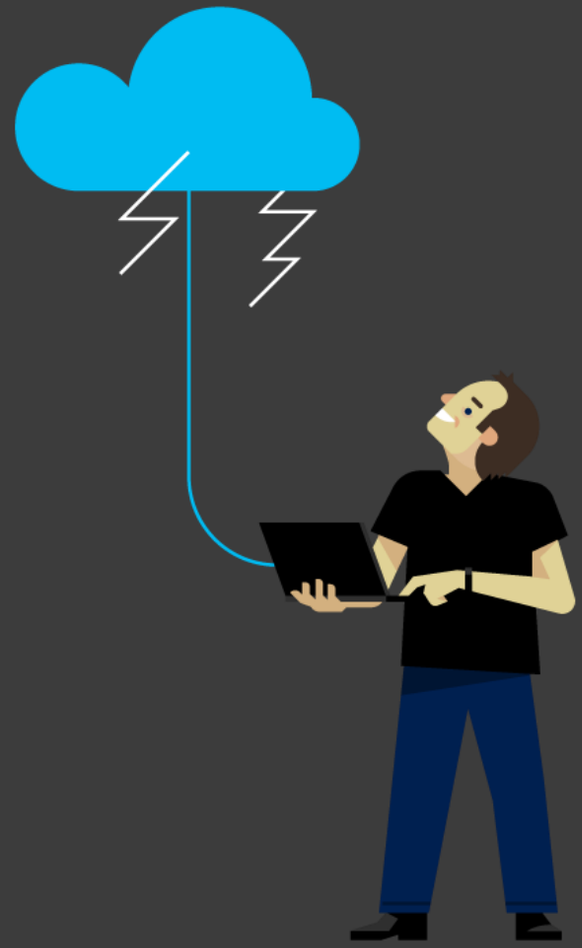
.NET Core configuration

Total 4 feature flags loaded

Key	Label	State	Description	Last modified	
LeaderboardListLimit	(No label)	Off On		10/20/2019, 1:14:42 PM	•••
LeaderboardListLimit	Canary	Off On		10/25/2019, 11:44:24 PM	•••
LeaderboardListLimit	Development	Off On	Development only!	10/26/2019, 3:58:42 PM	•••
LeaderboardListLimit	EarlyAdopters	Off On		10/25/2019, 11:44:52 PM	•••

Demo
Feature flags on Microsoft platform

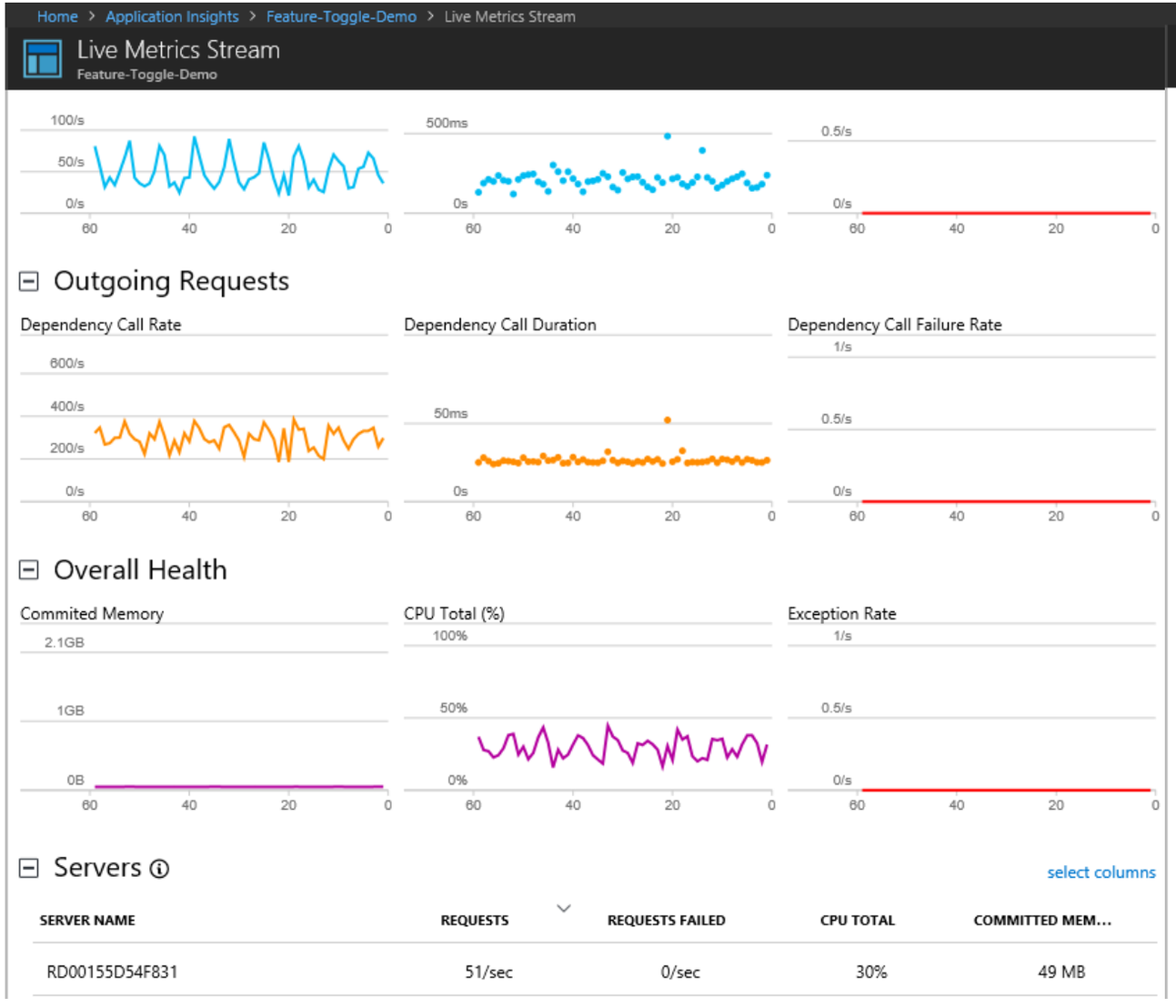
ASP.NET Core Feature Management
Azure App Configuration



Health and metrics

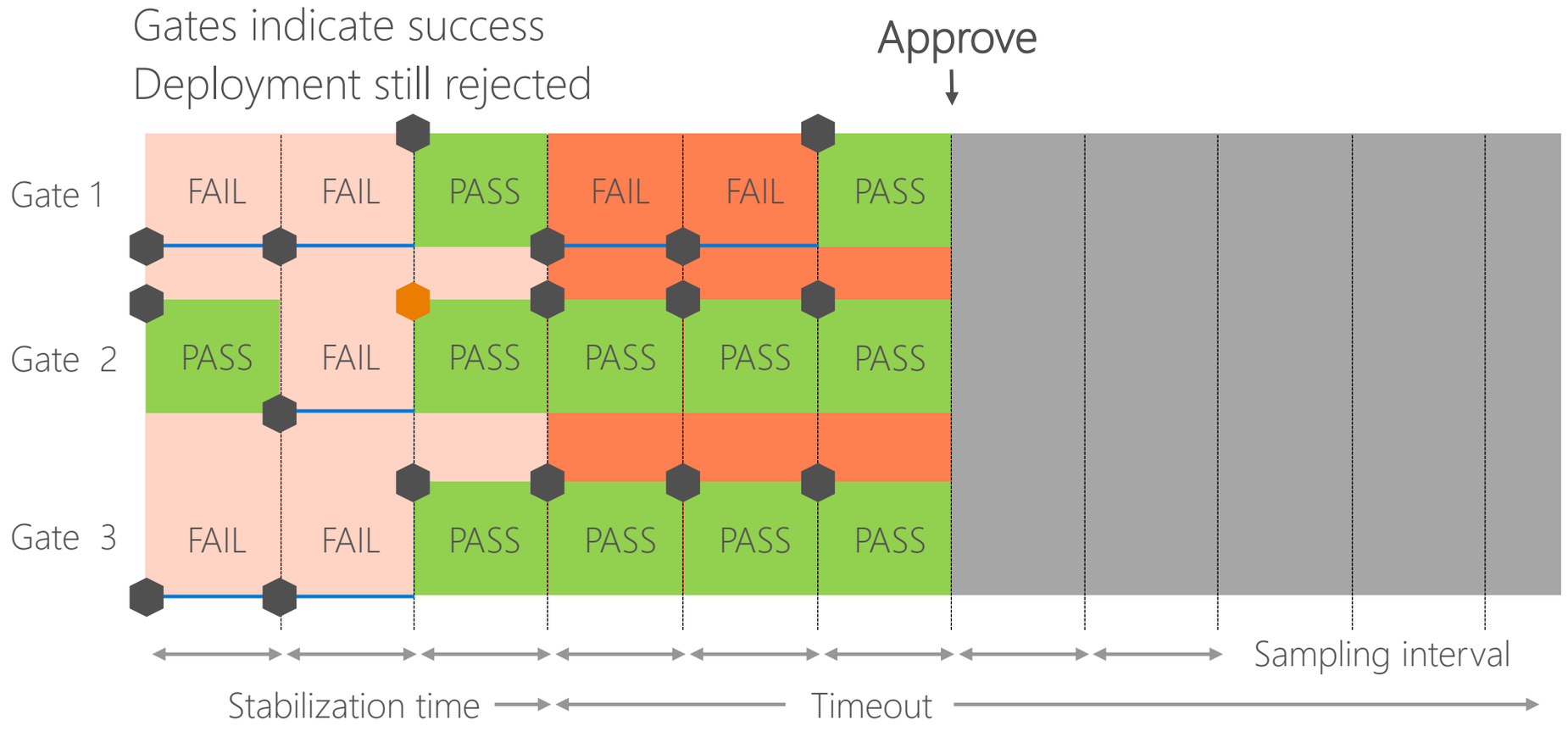
Telemetry is crucial to know about system stability

Absence of errors is not sufficient



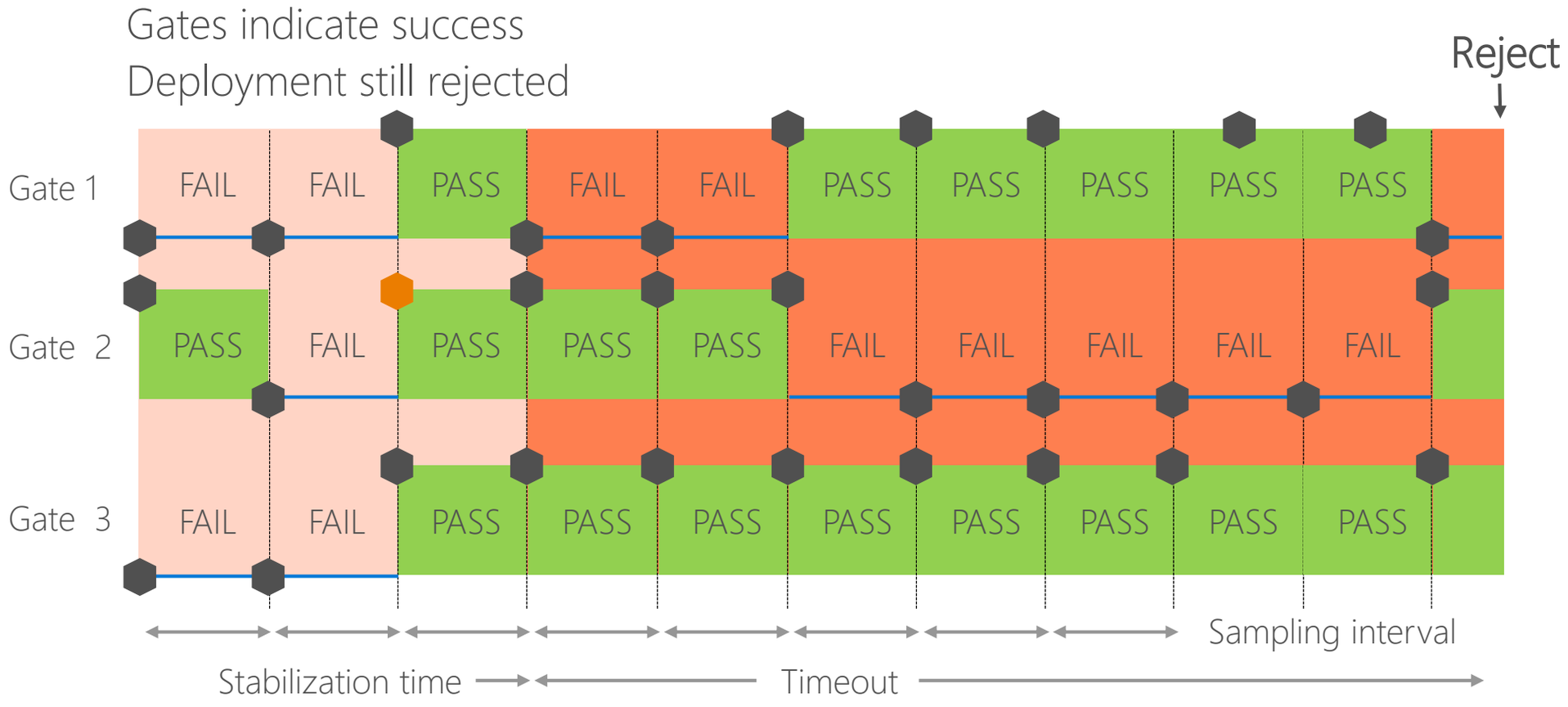
Telemetry

Essential to monitor your solution when toggling



Telemetry

Essential to monitor your solution when toggling



Development cycle with feature flags

1 Deploy

2 Release

Gives more autonomy

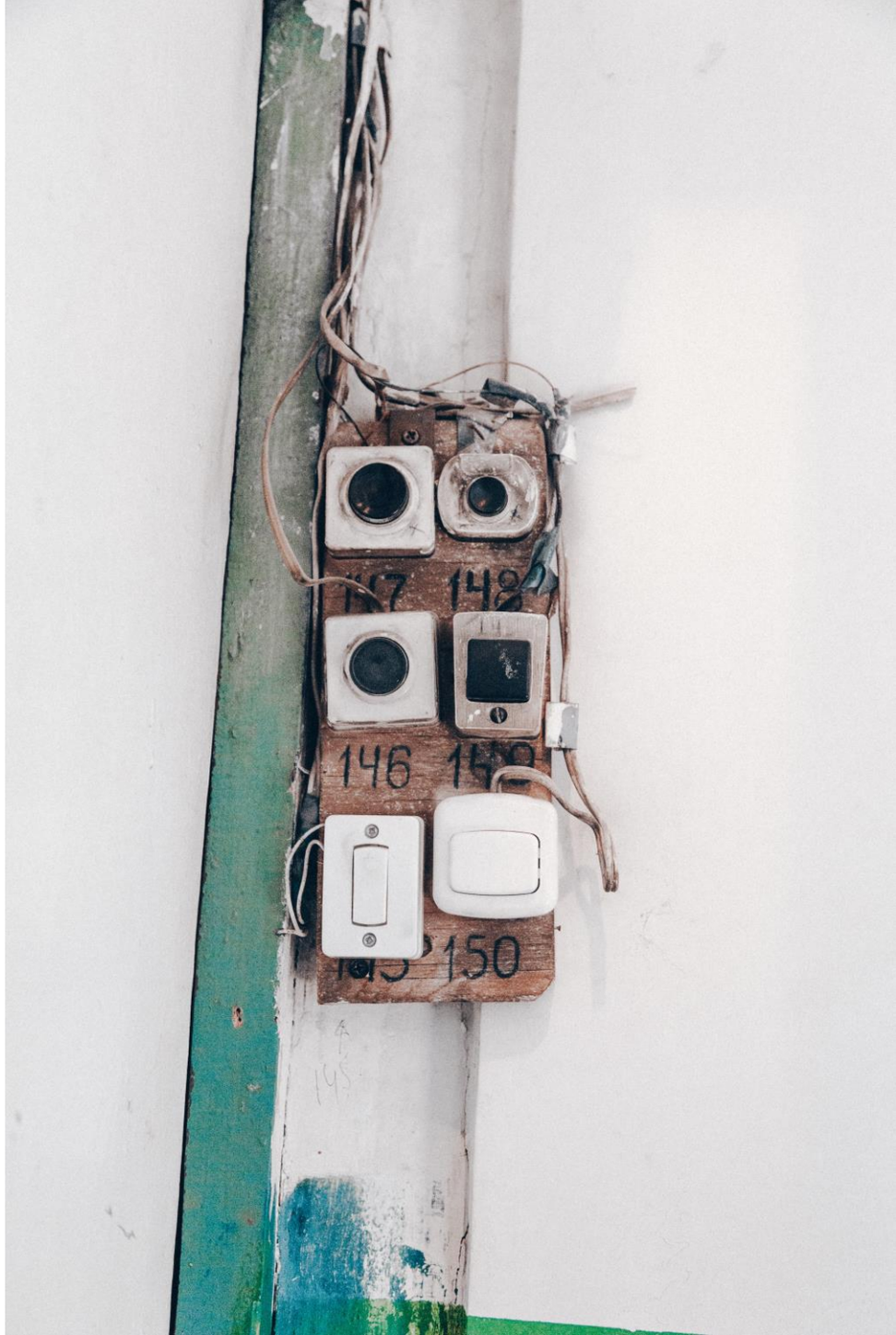
Full control over deployment and release

Deployment should include creation of flags in external environment

Flags introduce technical debt

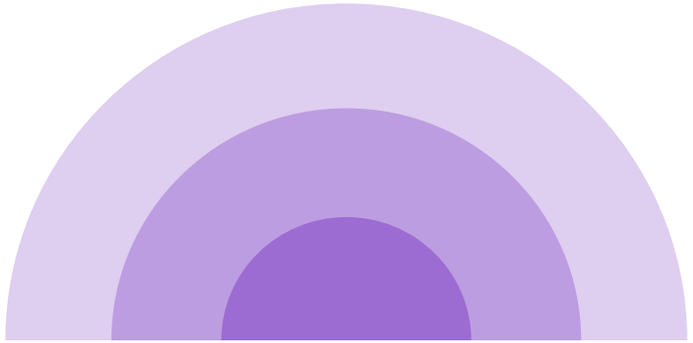
Have to clean up afterwards

Plan for removing flags in a sprint



Gradually exposing and releasing

Rings



Gradually increase blast radius

Deployment:

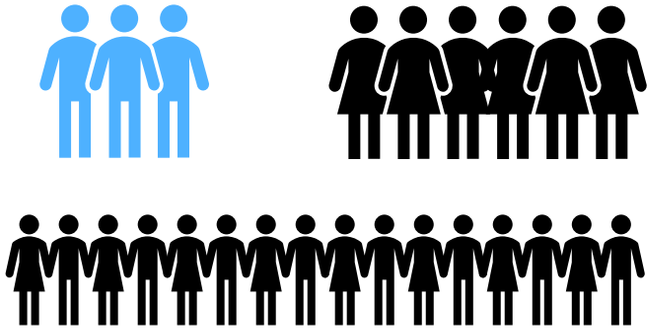
Different environments

Feature:

For a period of time

Increasing percentage

Cohorts



Different size groups

Canary

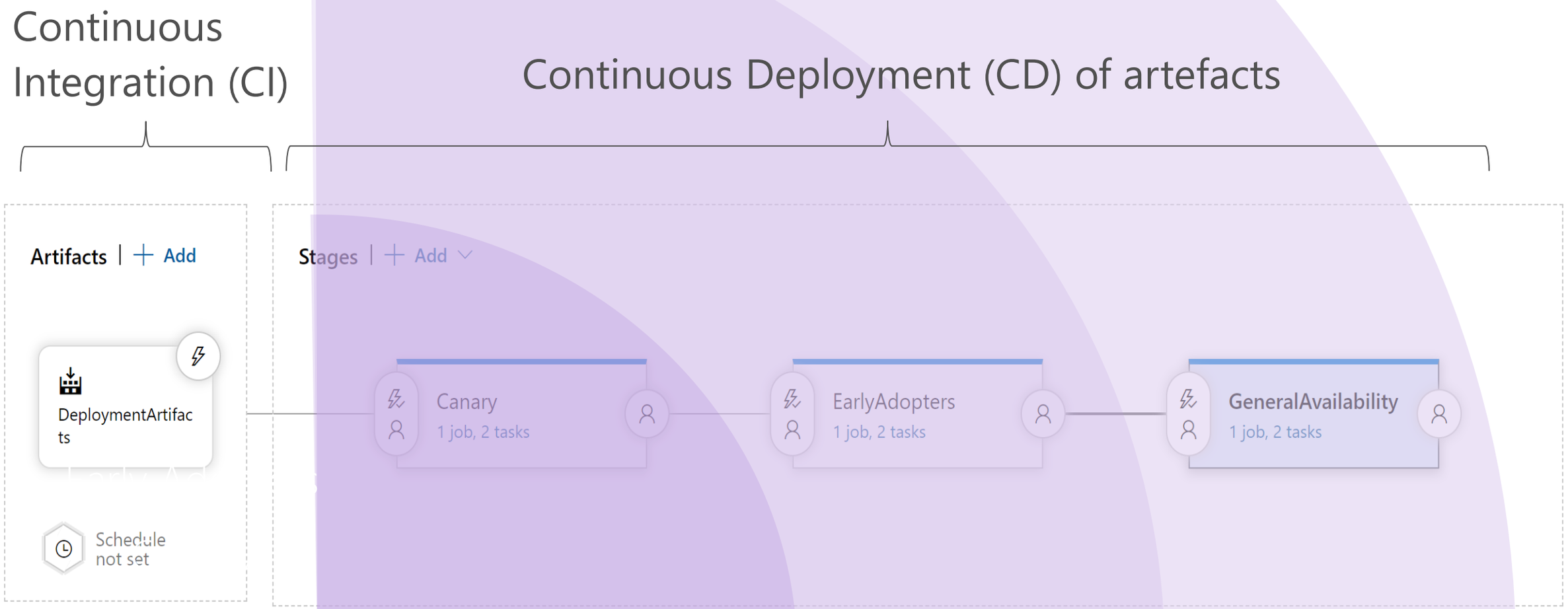
Early adopters

General public

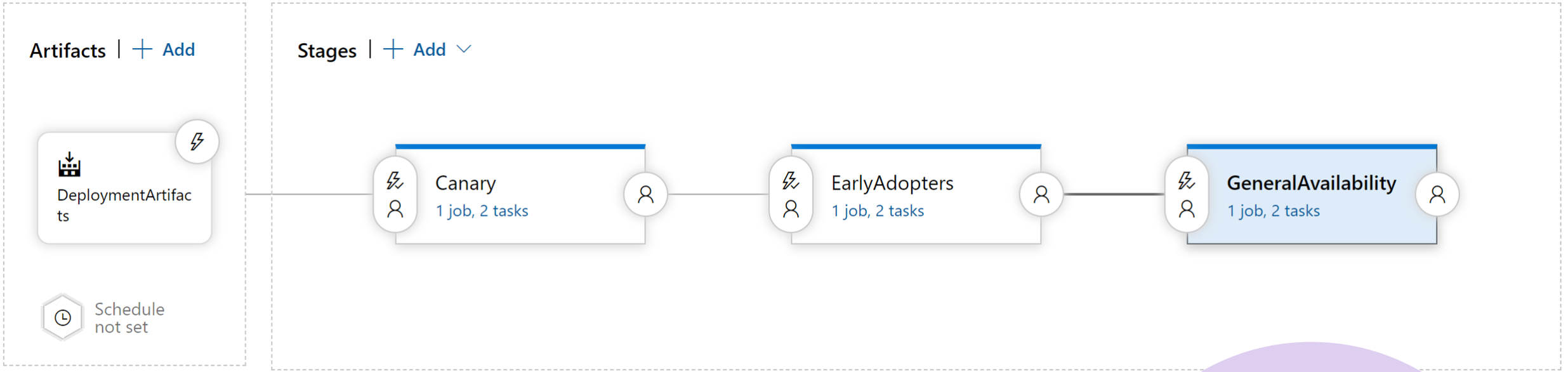
Can offer opt-in model

Combining deployment rings and release flags

Progressively reveal and expose features using flags



Blast radius of feature flags state



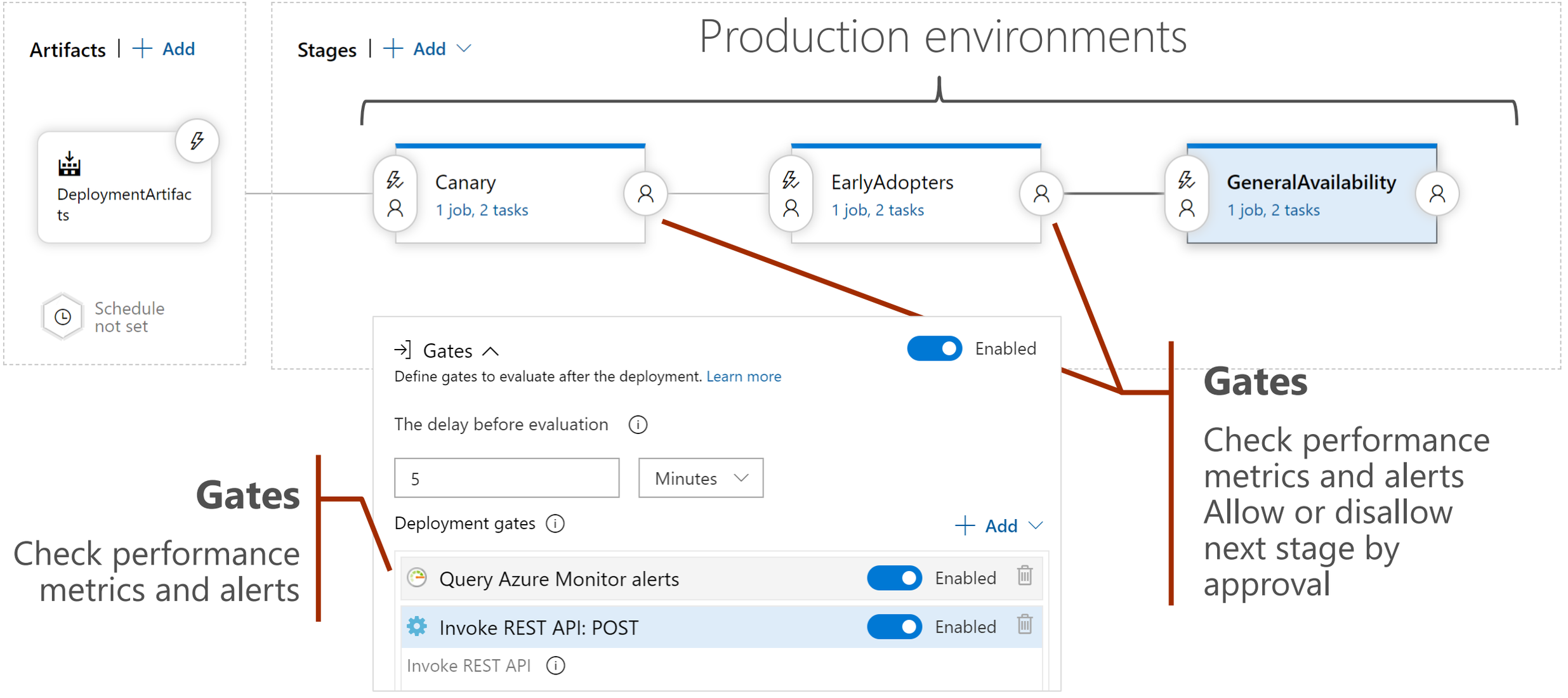
Same principles as deployment

Automated pipelines

Gradual exposure per flag and environment

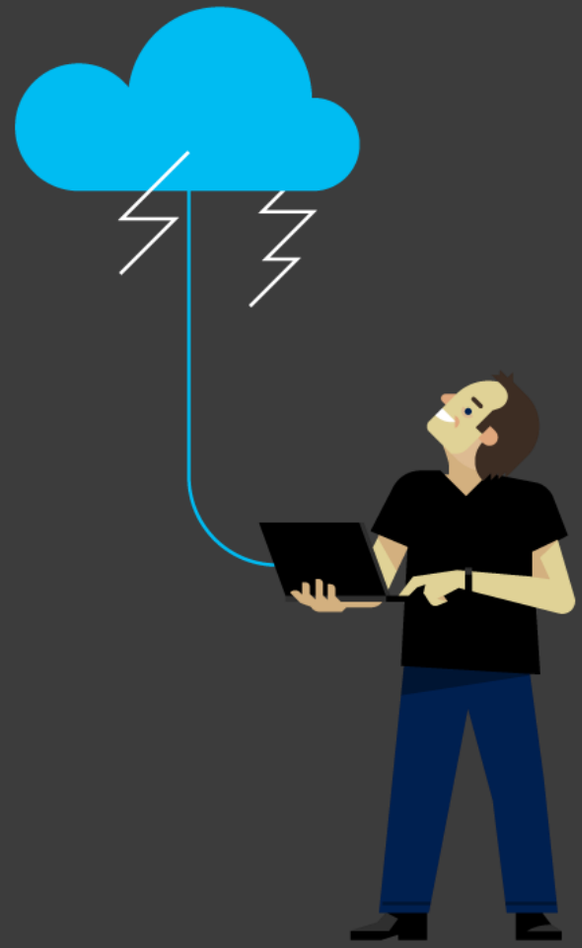
Deploy across environments after approval and quality gate

Pipelines with approval and gates for flags



Demo
Continuous release

Azure Pipelines



Testing

Unit testing as usual: test both implementations

Integration testing permutations doesn't work

New strategy:

- Offline service for configuration management (disaster)
- Mimic production flag states
- Members of cohorts

Test strategies

A|B testing

Dark launching

Canary releases

Testing in production

Making it real

Real features
Real data
Real environment
Real users

QA and UAT where it matters

A|B testing (aka split testing)

Combine feature flag with experiment led by hypothesis

Increase percentage that has feature enabled

Measure results, outcome and gather feedback

Previously Routing to two different implementations

Now Percentage filter in single implementation

Dealing with technical debt

Lifecycle of flag

1. Always off
2. Toggle flag based on rules
3. Always on

Remove

Create clean-up branch before merge to master

Smells

Too many flags

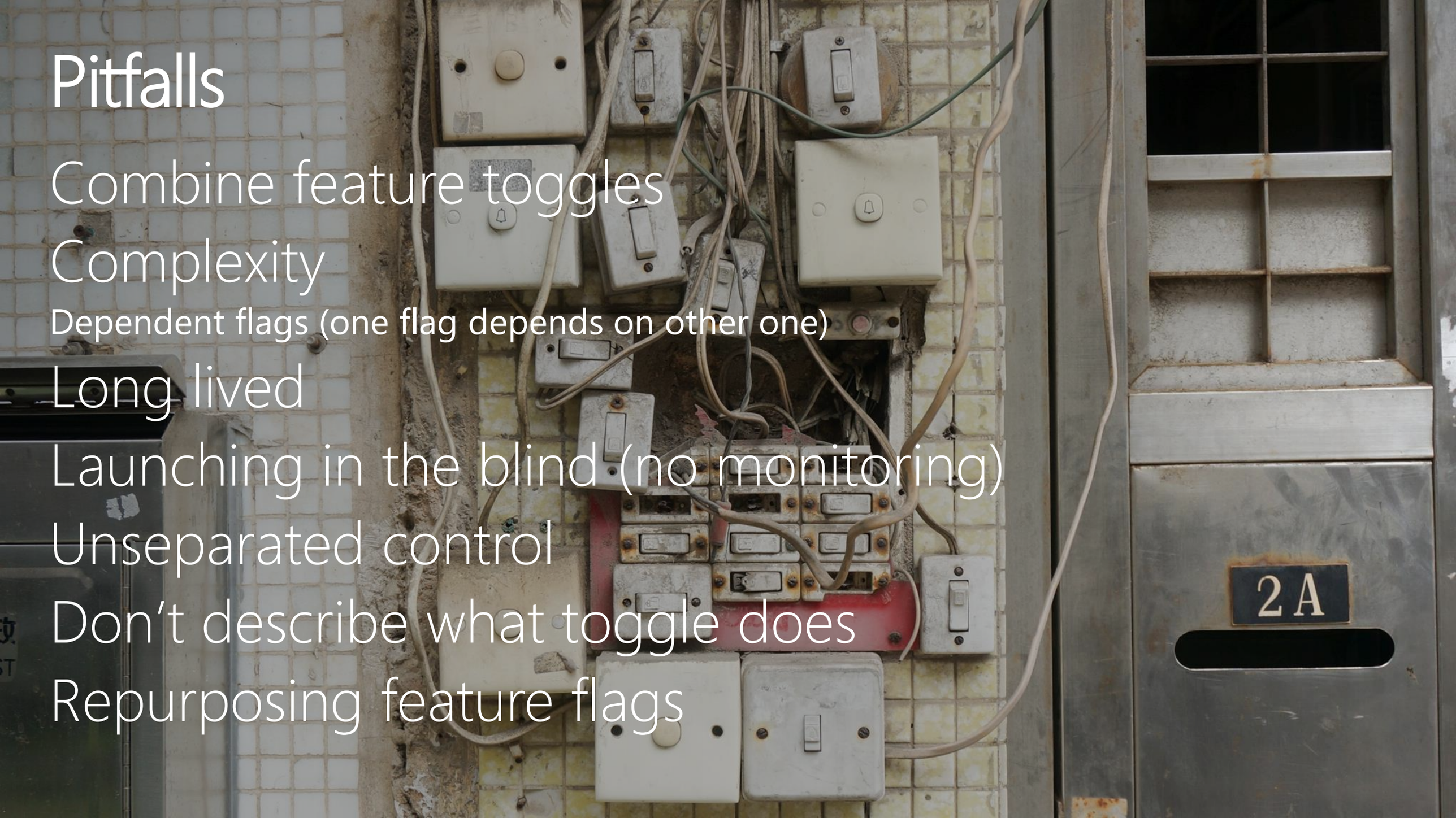
Flags used in multiple places

Strategy pattern needed?

Duplicate code vs if statements

Fine grained toggles

Toggling business value vs technical capabilities



Pitfalls

Combine feature toggles

Complexity

Dependent flags (one flag depends on other one)

Long lived

Launching in the blind (no monitoring)

Unseparated control

Don’t describe what toggle does

Repurposing feature flags

Best practices

Avoid flags as long-lived functionality

Application management code is not same as a feature under a toggle

Separate flag management from app

In case app becomes unresponsive for managing flags

Monitor usage of flags

New, not used yet

Not used (remove from config)

Being used

Everyone uses one variation (remove from code

Enable just one toggle at a time

Measure outcome and give time to stabilize

Tips and recommendations

Add hardcoded fallback

Make it a business decision

Circuit breaker as a safeguard feature toggle

Start with master flags and add smaller flags later

Define naming convention for flags

Questions and Answers

Maybe later?

@alexthissen
athissen@xpirit.com

Resources

Feature toggles

https://martinfowler.com/articles/feature-toggles.html
https://opensource.com/article/18/2/feature-flags-ring-deployment-model
https://docs.microsoft.com/en-us/azure/devops/migrate/phase-features-with-feature-flags
https://dougseven.com/2014/04/17/knightmare-a-devops-cautionary-tale/
https://featureflags.io/

.NET Core Feature Management

https://github.com/microsoft/FeatureManagement-Dotnet

Demo source code

https://github.com/alexthissen/featuremanagement

Azure App Configuration

https://github.com/Azure/AppConfiguration
https://docs.microsoft.com/en-us/azure/azure-app-configuration/use-feature-flags-dotnet-core