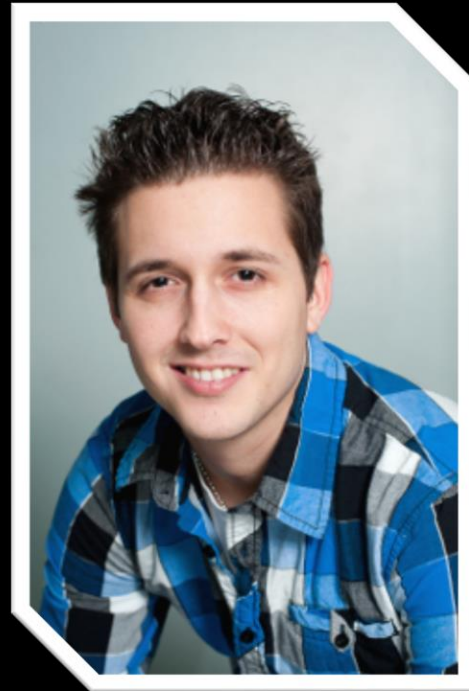
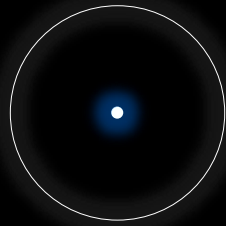




SPS COMMERCE

UNLEASHING DEPLOY VELOCITY WITH FEATURE FLAGS

A DEVOPS JOURNEY



TRAVIS GOSSELIN

PRINCIPAL SOFTWARE ENGINEER

DEVELOPER EXPERIENCE 

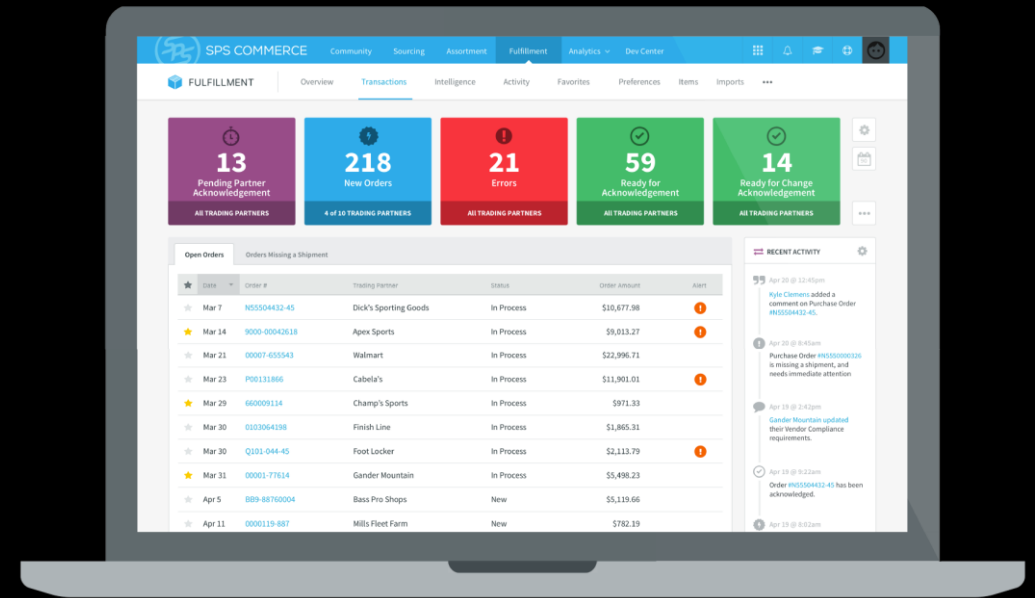
@TRAVISJGOSSELIN 

www.travisgosselein.com 



SPS COMMERCE

INFINITE RETAIL POWER™



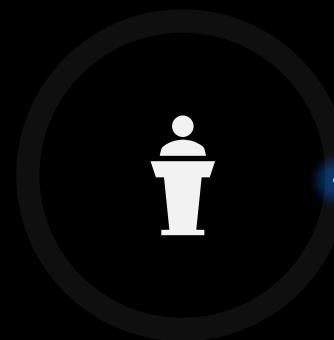


DEVOPS

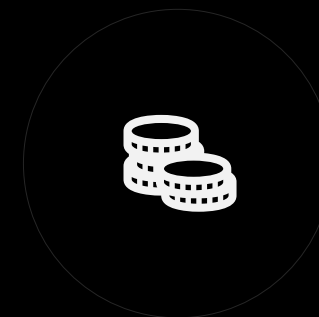
A CULTURE & STATE OF MIND



Continuous

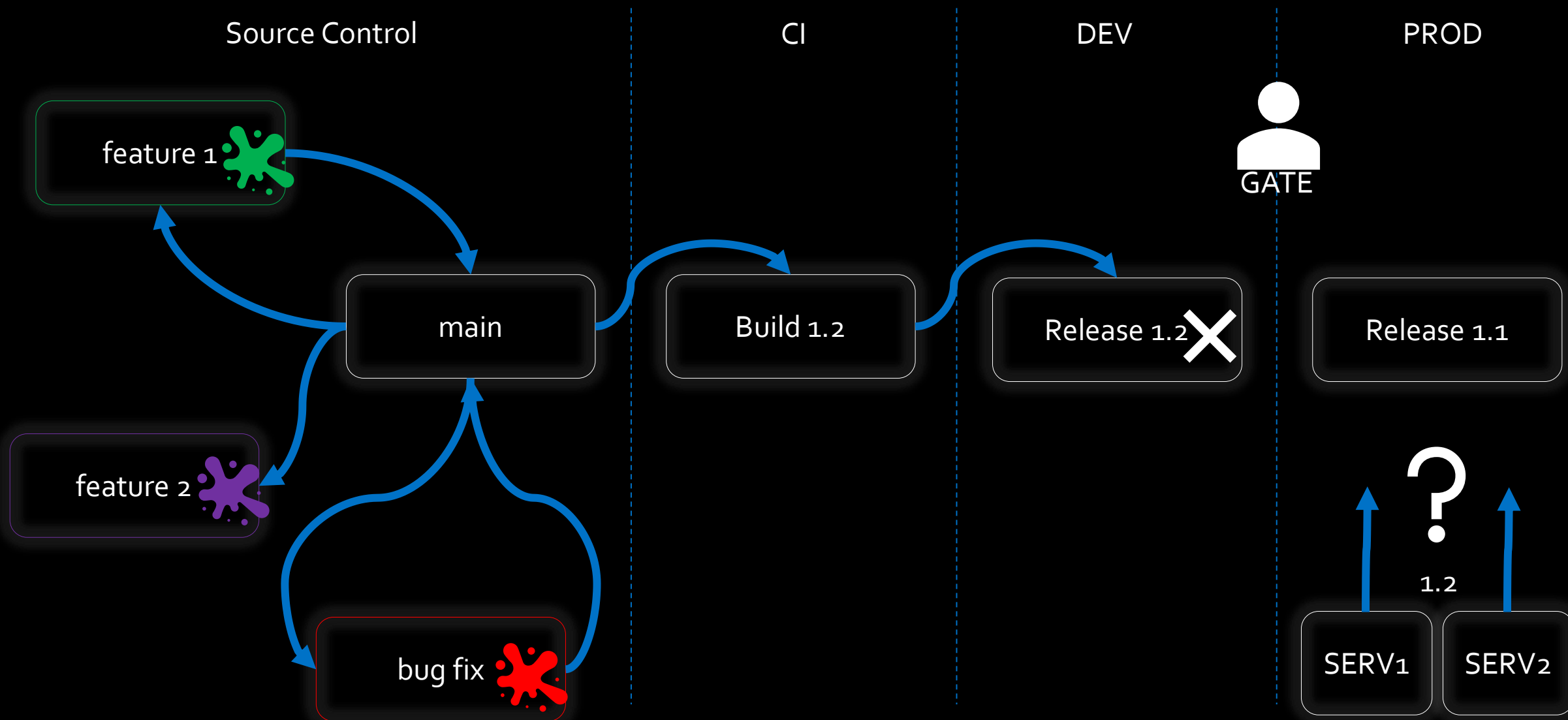


Automation

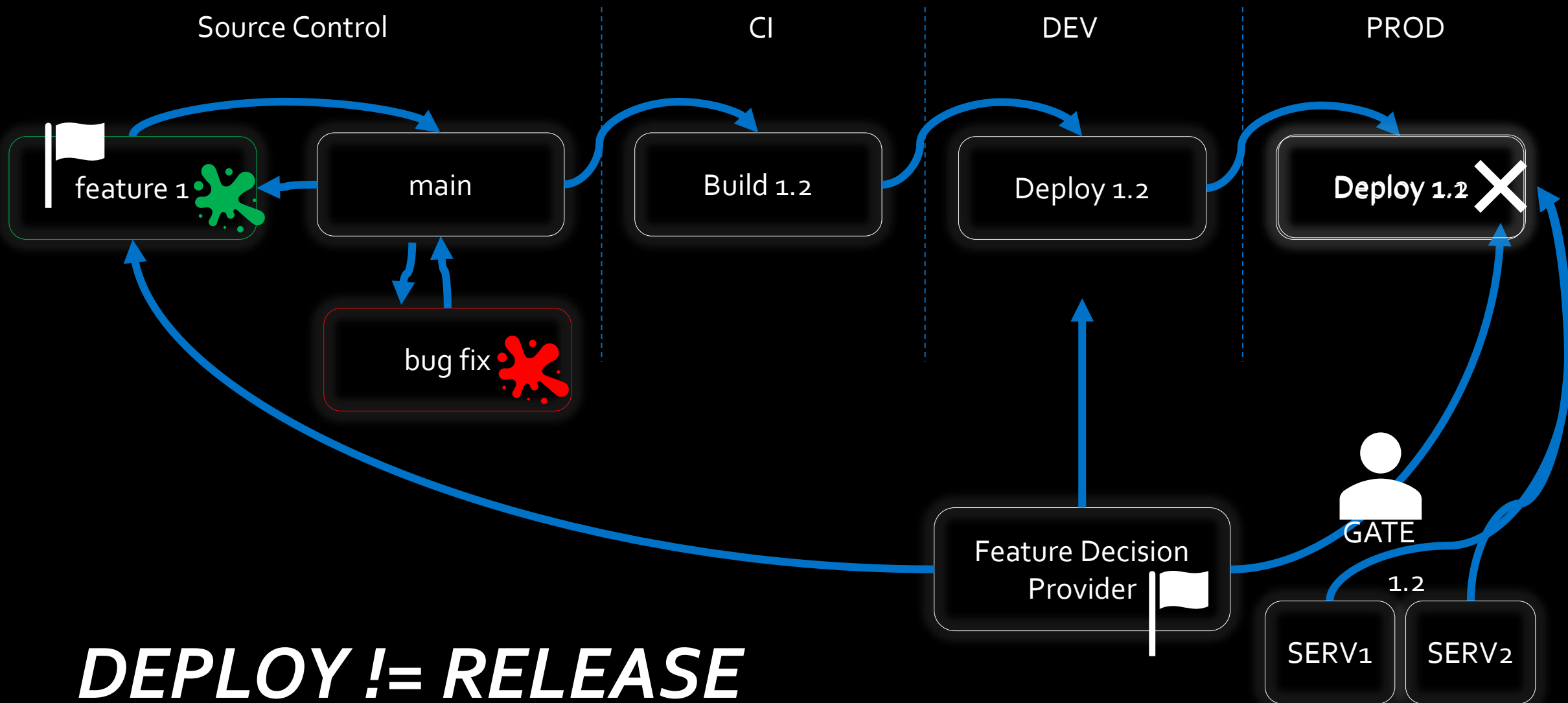


Sharing

The Problem



The Solution



*"...a powerful technique,
allowing teams to modify
system behavior without
changing code"
(MartinFowler.com)*

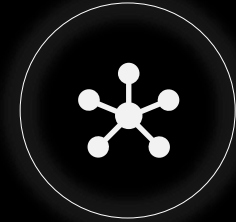


FEATURE FLAGS

AKA "CONFIG FLAGS", "FEATURE TOGGLES", "FLIPPERS"



Release



Operational



Experiment



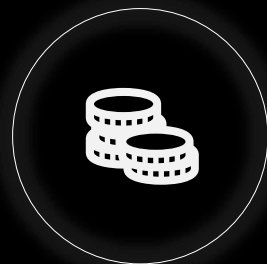
Permission

Flexibility with Feature Flags



Branching Strategy

Shorter, Multiple
Active Releases



True CI

Continuous
Integration the way it
should be!



Ship Faster

Confidently Deploy
Rollback Features



Testing in PROD

A/B Testing
Canary Releases
Less Environments



Culture

Product Owners
Manage Features

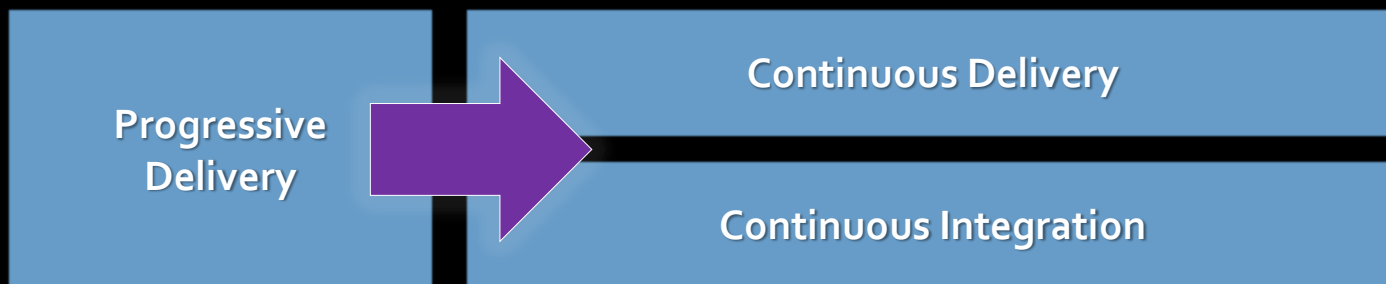
Progressive Delivery

//

Progressive Delivery is a modern software development lifecycle that builds upon the core tenets of Continuous Integration and Continuous Delivery (CI/CD). Organizations that employ Progressive Delivery ship code faster, reduce risk, and continuously improve the customer experience.

//

Launchdarkly.com



Azure DevOps





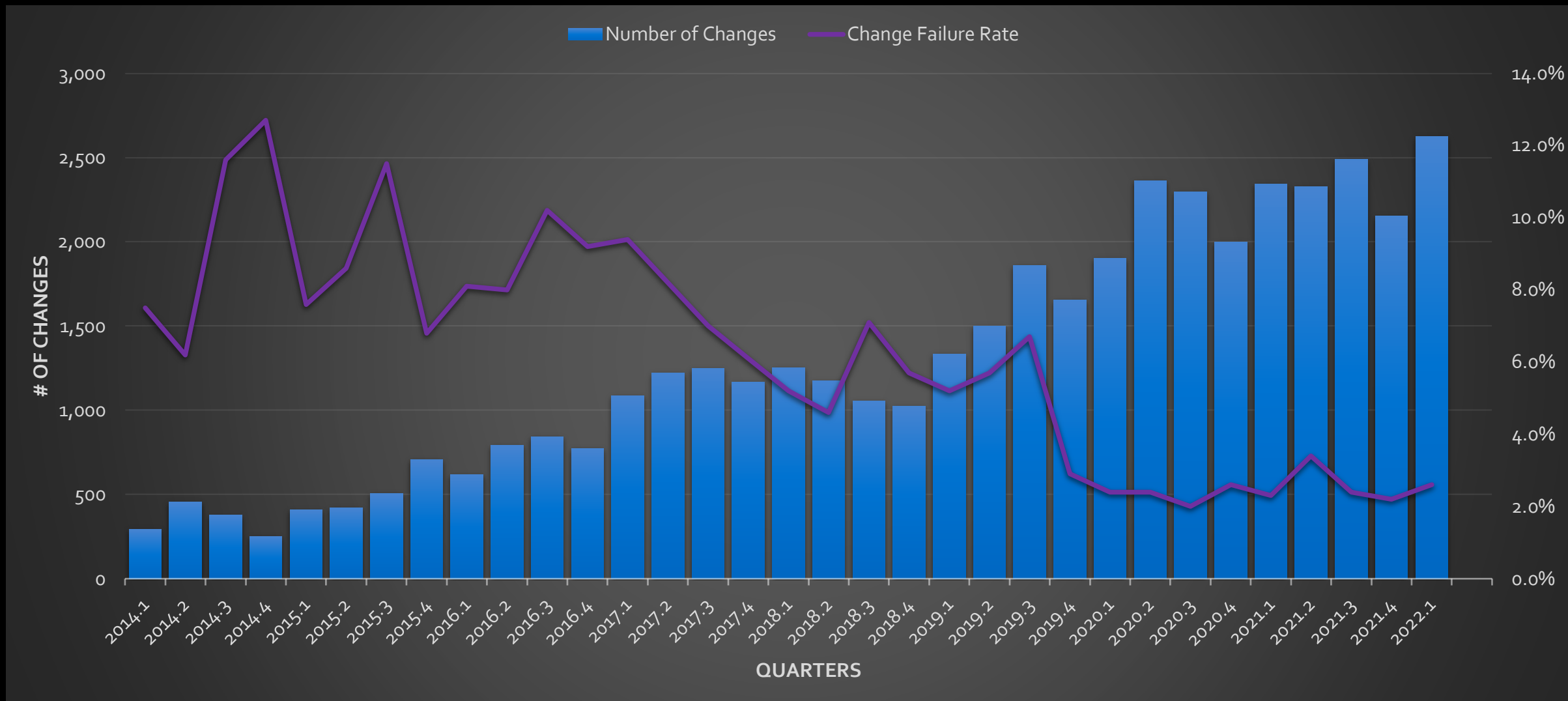
SOFTWARE DELIVERY AND OPERATIONAL PERFORMANCE



Performance Metrics		Higher Performers
✓	Deployment Frequency	On-demand (whenever we want)
✓	Lead Time for Changes	Less than an hour
✓	Mean Time to Restore	Less than one hour
✓	Change Failure Rate	Less than 5%

circleci.com/2021-puppet-state-of-devops.html

SPS Commerce Change Performance



Simple Feature Flag

```

public void SetupUser(UserContext user)
{
    CreateUser(user);

    if (IsSendgridEmailEnabled(user))
    {
        SendSendgridEmail(user);
    }
    else
    {
        SendLocalSmtplibEmail(user);
    }
}
  
```

← Feature Flag

← *NEW* Code

← *OLD* Code

```

public bool IsSendgridEmailEnabled(user)
{
    return service
        .UseFeature(user, "Sendgrid");
}
  
```



IMPORTANT REALIZATIONS

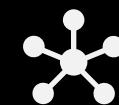
YOUR FEATURE FLAG HONEYMOON IS OVER!



Shifting Left



Maintainability



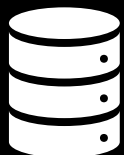
Changes to
Existing Code



Feature
Management
Provider

Feature Management Providers

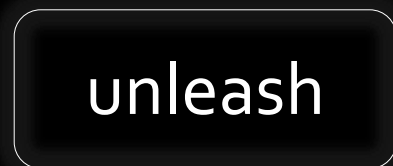
General Infrastructure



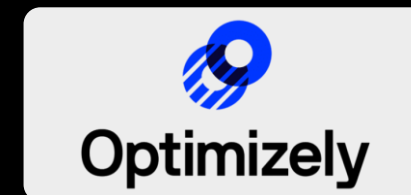
Microsoft
Azure



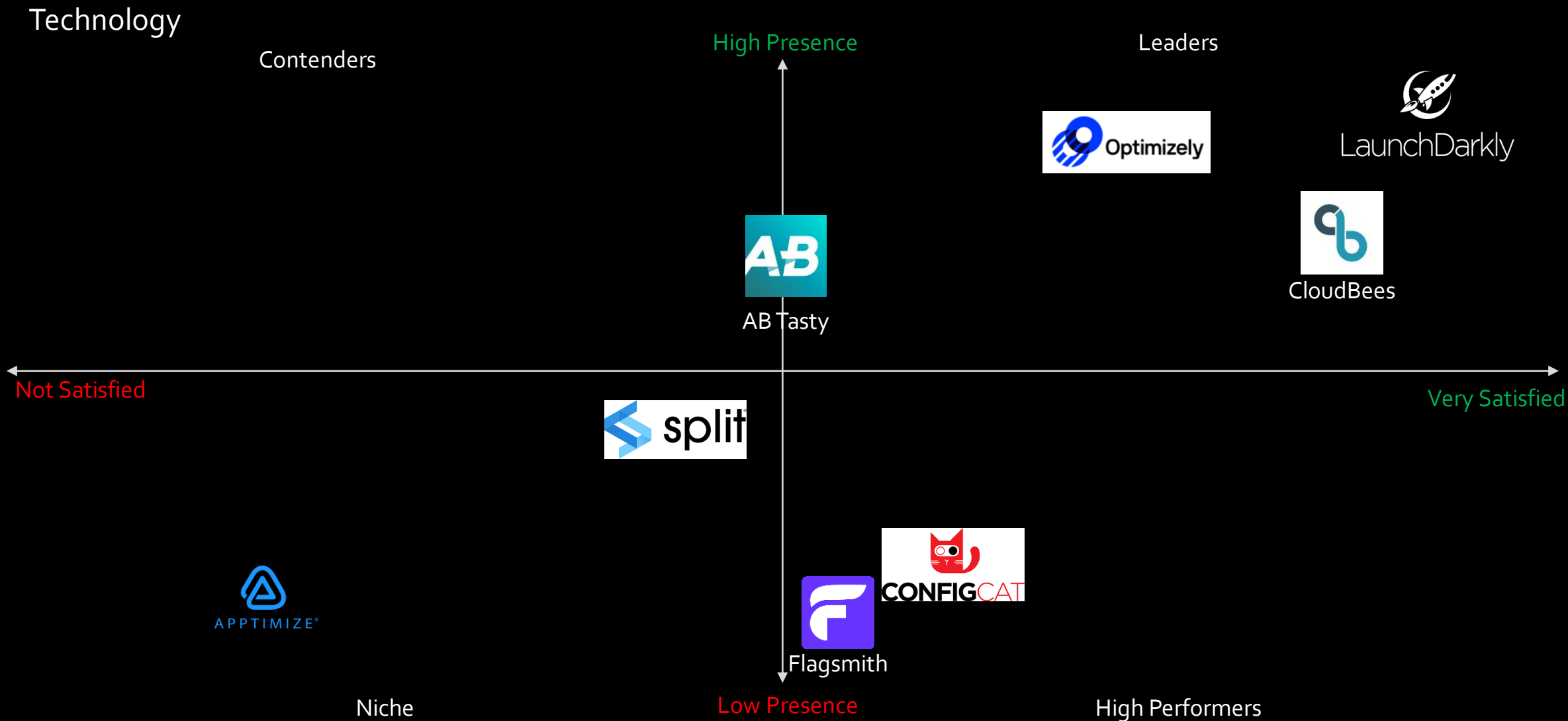
Targeted & Open Source



Full Service



Feature Management Providers





THOUGHTWORKS TECH RADAR

Techniques

ADOPT ?

1. Applying product management to internal platforms
2. Infrastructure as code
3. Micro frontends
4. Pipelines as code
5. Pragmatic remote pairing
6. Simplest possible feature toggle

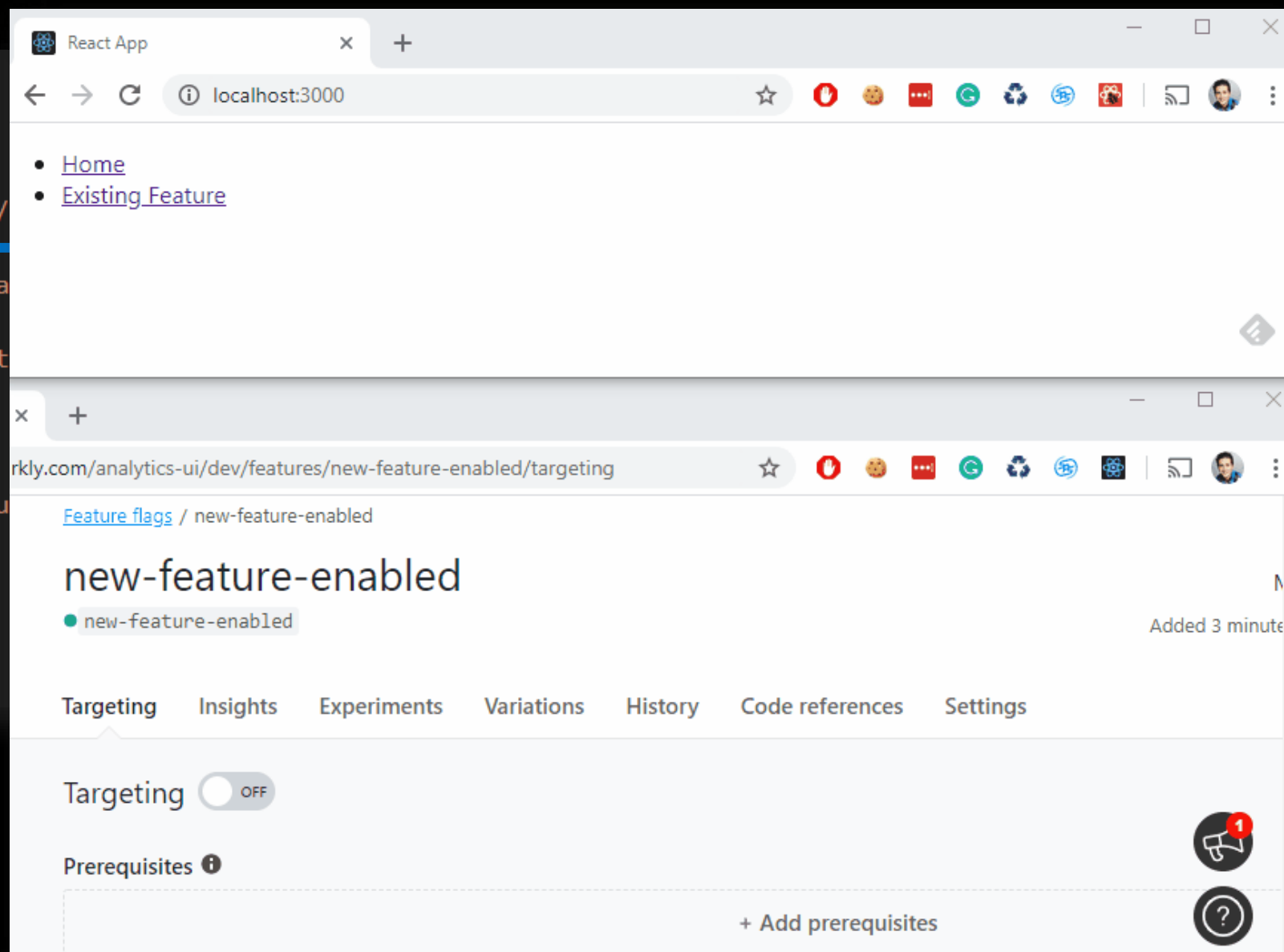
Unfortunately, feature toggles are less common than we'd like, and quite often we see people mixing up its types and use cases. It's quite common to come across teams that use heavyweight platforms such as LaunchDarkly to implement feature toggles, including release toggles, to benefit from Continuous Integration, when all you need are if/else conditionals. Therefore, unless you need A/B testing or canary release or hand over feature release responsibility to business folks, we encourage you to use the **simplest possible feature toggle** instead of unnecessarily complex feature toggle frameworks.



UI Routing

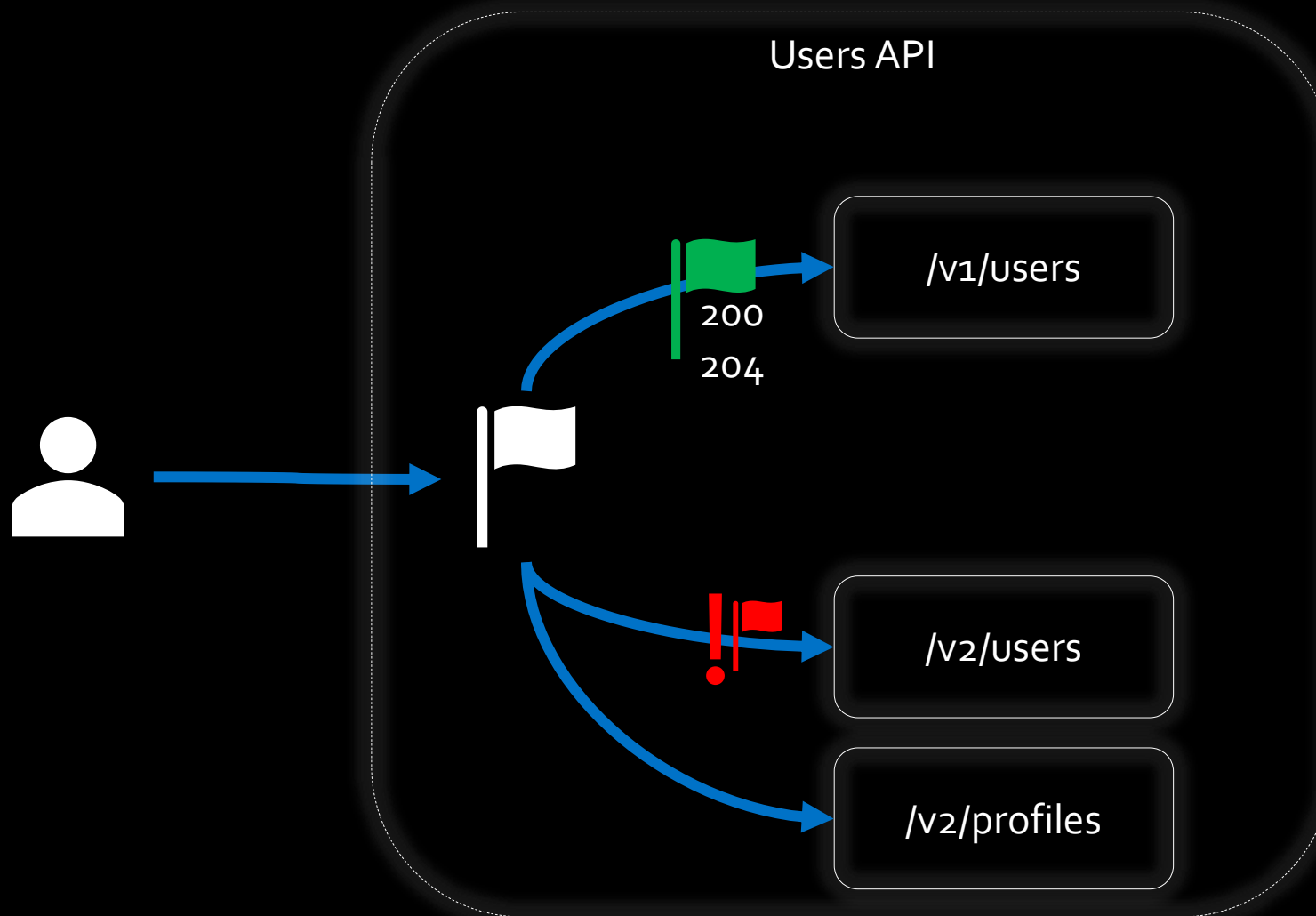
```

10 export default () => {
11   return (
12     <nav>
13       <ul>
14         <li><Nav label="Home" link="/" />
15         <FeatureFlagRouter ←
16           featureKey="newFeatureEnabled"
17           fallback= {
18             <li><Nav label="Existing Feature" link="/existing-feature" />
19           }
20           key="existing-feature"
21         >
22         <li><Nav label="New Feature" link="/new-feature-enabled" />
23       </FeatureFlagRouter>
24     </ul>
25   </nav>
26 )
27 }
  
```

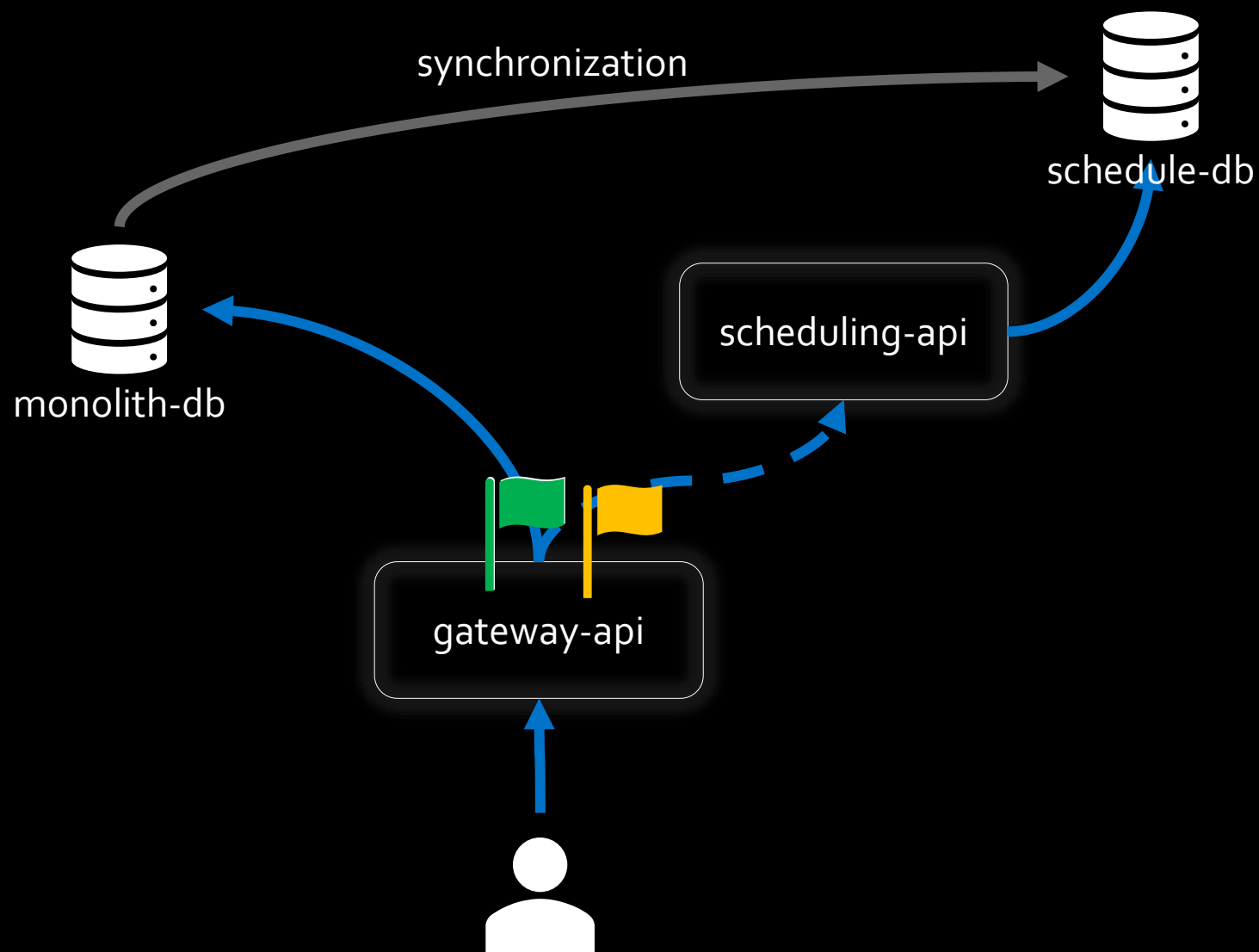


The screenshot shows a web browser with two tabs. The first tab, titled "React App", is at `localhost:3000` and displays a navigation menu with two items: "Home" and "Existing Feature". A blue arrow points from the `<FeatureFlagRouter>` component in the code on the left to the "Existing Feature" link in the browser. The second tab is at `arkly.com/analytics-ui/dev/features/new-feature-enabled/targeting` and displays the configuration page for the "new-feature-enabled" feature flag. The page title is "new-feature-enabled" and it shows a "Targeting" toggle set to "OFF". Below the toggle is a "Prerequisites" section with a "+ Add prerequisites" button. The browser's address bar and various extension icons are visible at the top of the browser window.

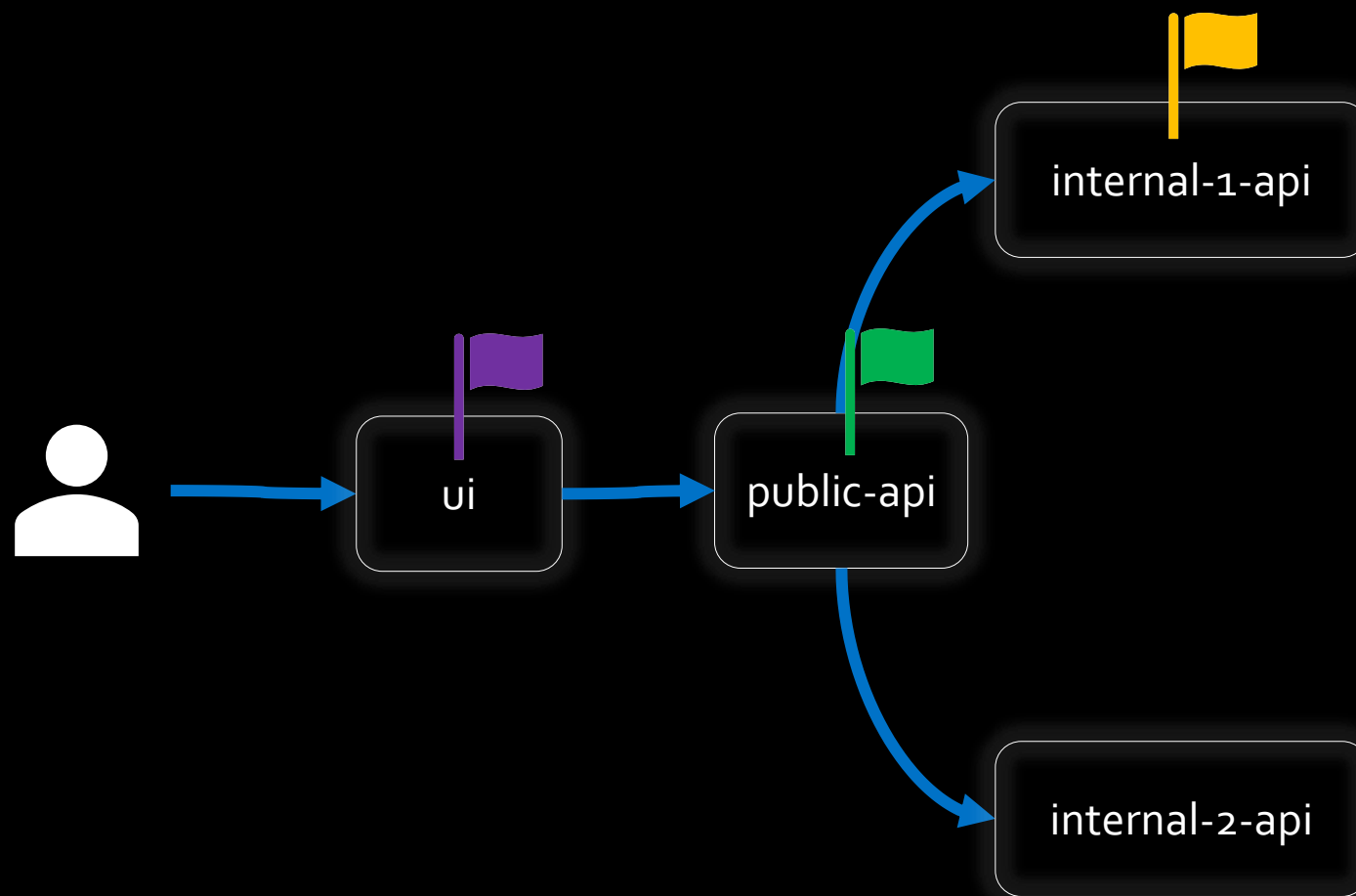
APIs



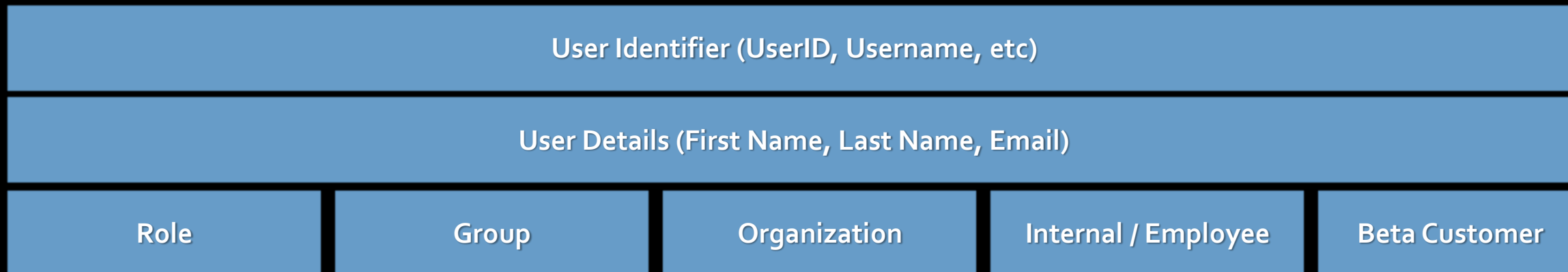
Migrations



Coordination



User Context



```
const user = {
  userId: '123456'
  name: 'Travis Gosselin'
};
const client = featureProvider.initialize(user);
```

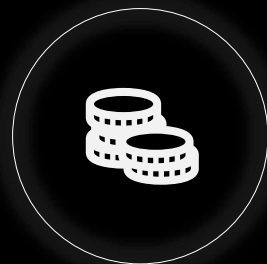


```
const user = {
  userId: '123456'
  firstName: 'Travis',
  lastName: 'Gosselin'
};
const client = featureProvider.initialize(user);
```

Other Scenarios



Log Level
Verbosity



Dynamic
Configuration



Kill Switches

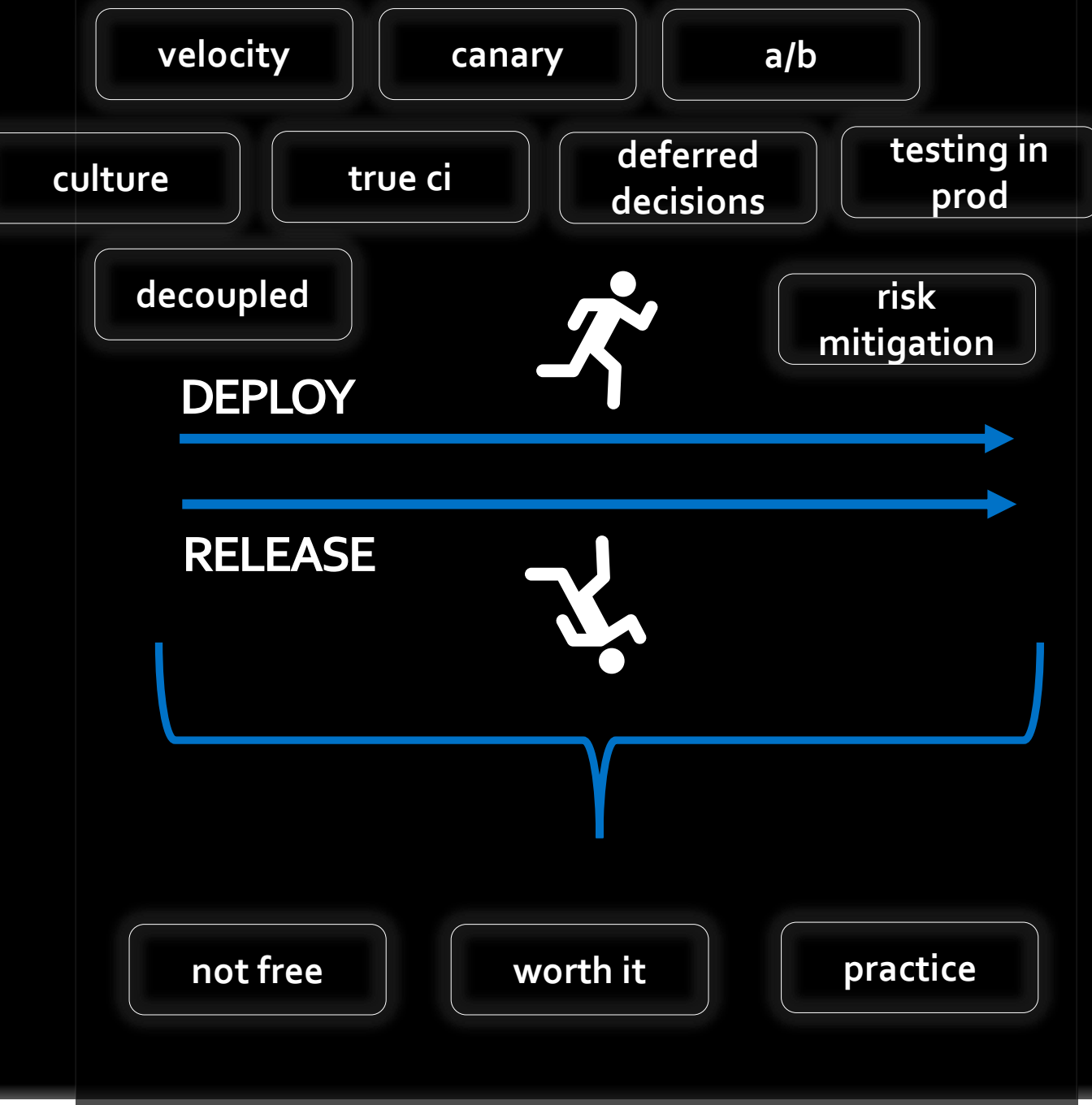


Sunset Features



Timed Features

UNLEASHING DEPLOY VELOCITY WITH FEATURE FLAGS



TRAVIS GOSSELIN 

@travisjgosselin 

www.travisgosselin.com 