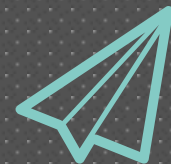


Continuous Delivery und Deployment

Dipl-Ing. (FH) Michael Johann



Warum Continuous Delivery?

Vor dem Wie klären wir erstmal das Warum

Warum CD?

Zuverlässigkeit
Zuverlässiges Ausliefern von Software

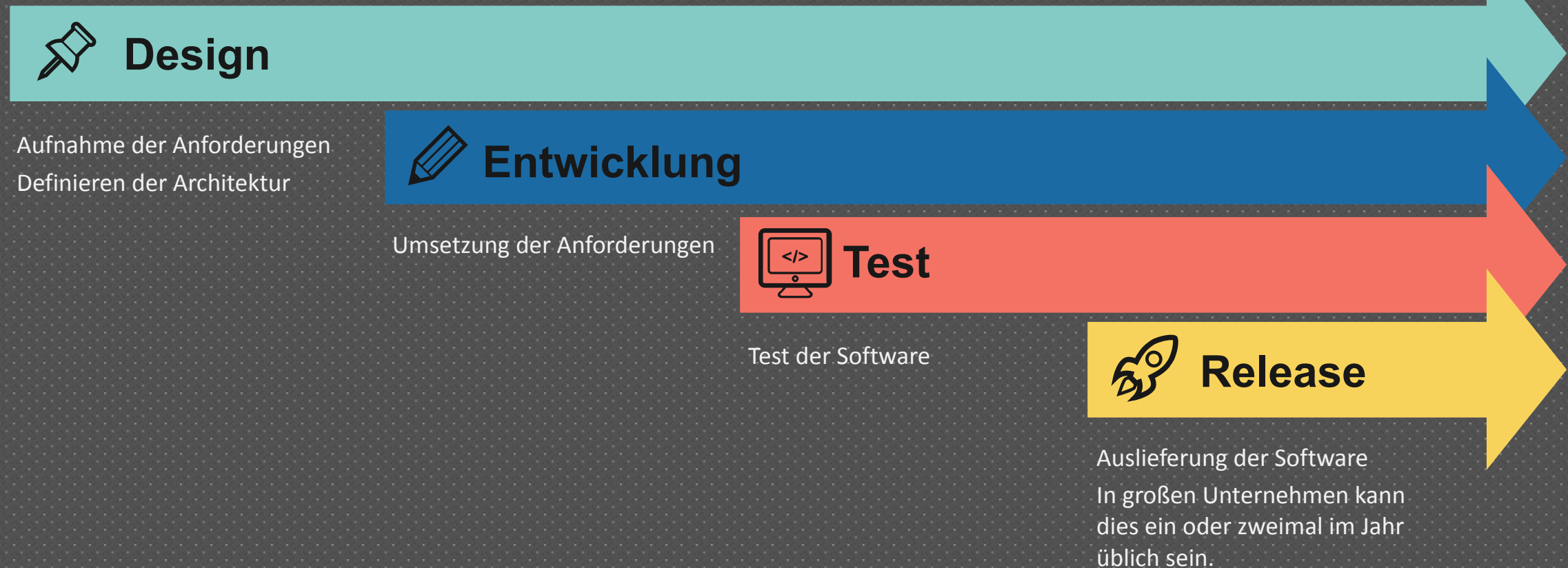
Schnelligkeit
Schnellere Auslieferung von Software.
Stichwort: „Time To Market“

Ständige Innovation
Neue Features schneller realisieren

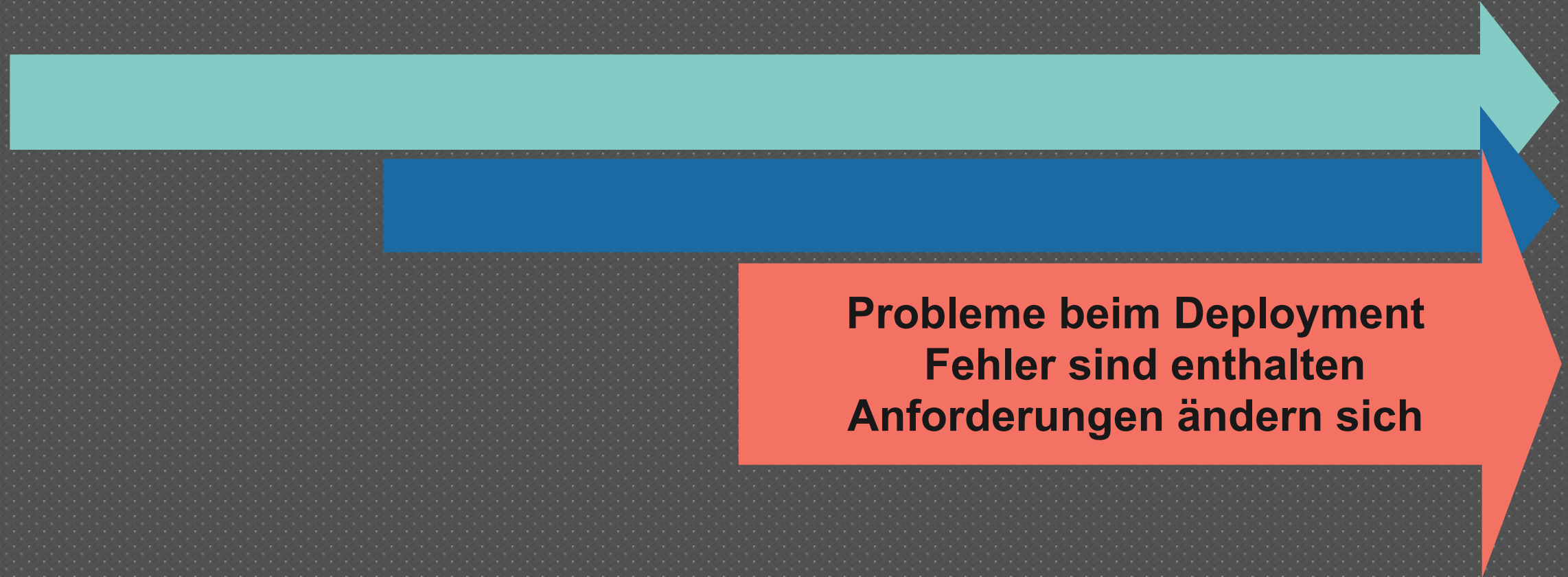
Kundennähe
Kunden erhalten in kurzen Abständen neue Features



Der klassische Ablauf



— Die Realität



— Die Realität

6

Feature falsch

Es kann sein, dass ein Feature falsch implementiert wurde. Gründe hierfür gibt es viele. Die Tatsache bleibt.

Aufbauten

Weitere Features sind auf falsch implementierten Features aufgebaut worden.

Bugs, Bugs

Fehler häufen sich und können nur schwer behoben werden.

Probleme

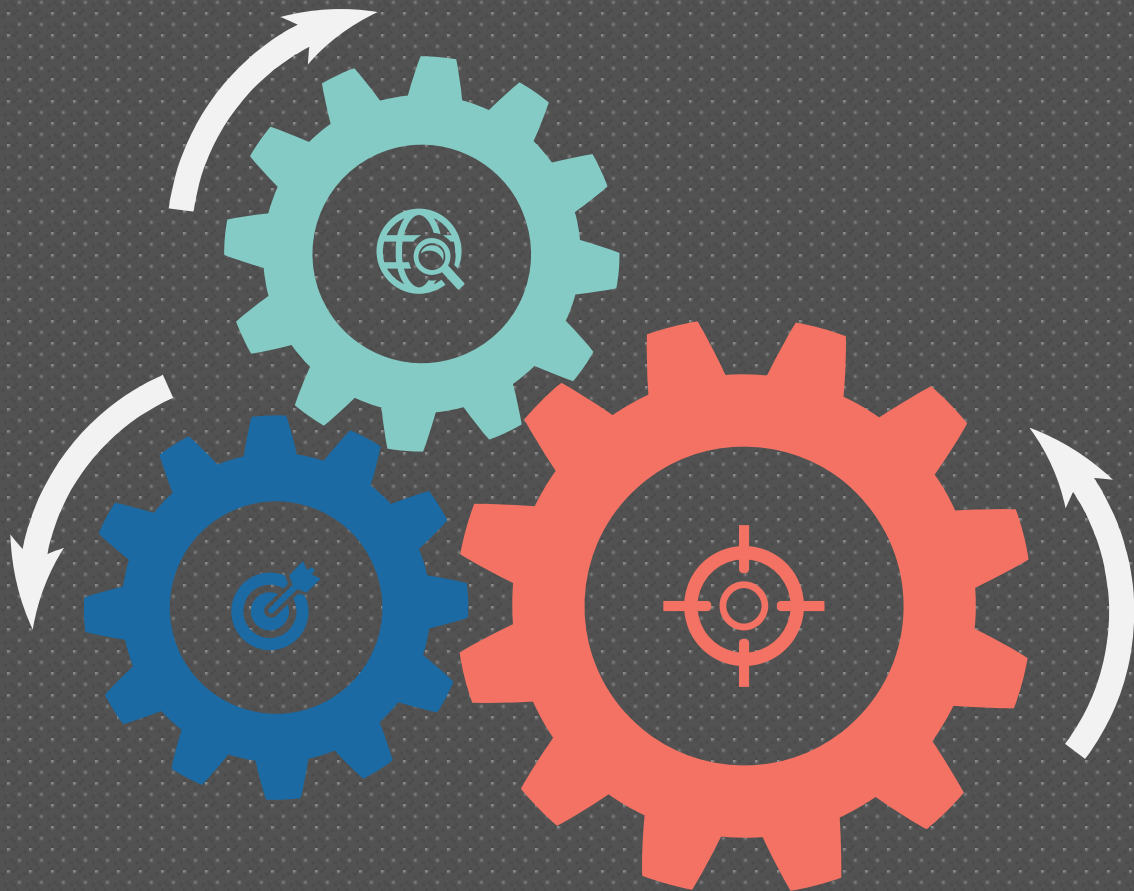
Alles zusammen führt zu enormen Problemen bei der Auslieferung. Hier scheitern Unternehmen immer wieder. Vor allem bei monolithischen Deployments.



Wie kann Continuous Delivery helfen?

Wir wollen alle ruhiger schlafen

1. Das Richtige bauen



Iteratives Vorgehen

Continuous Delivery hilft uns, bessere Software zu entwickeln und auszuliefern.



Kleine Auslieferungseinheiten

Häufige Auslieferung kleiner Features und Verbesserungen.



Schnelle Rückmeldungen

Erst wenn Kunden mit der Software „gespielt“ haben, bekommen Entwickler Feedback.

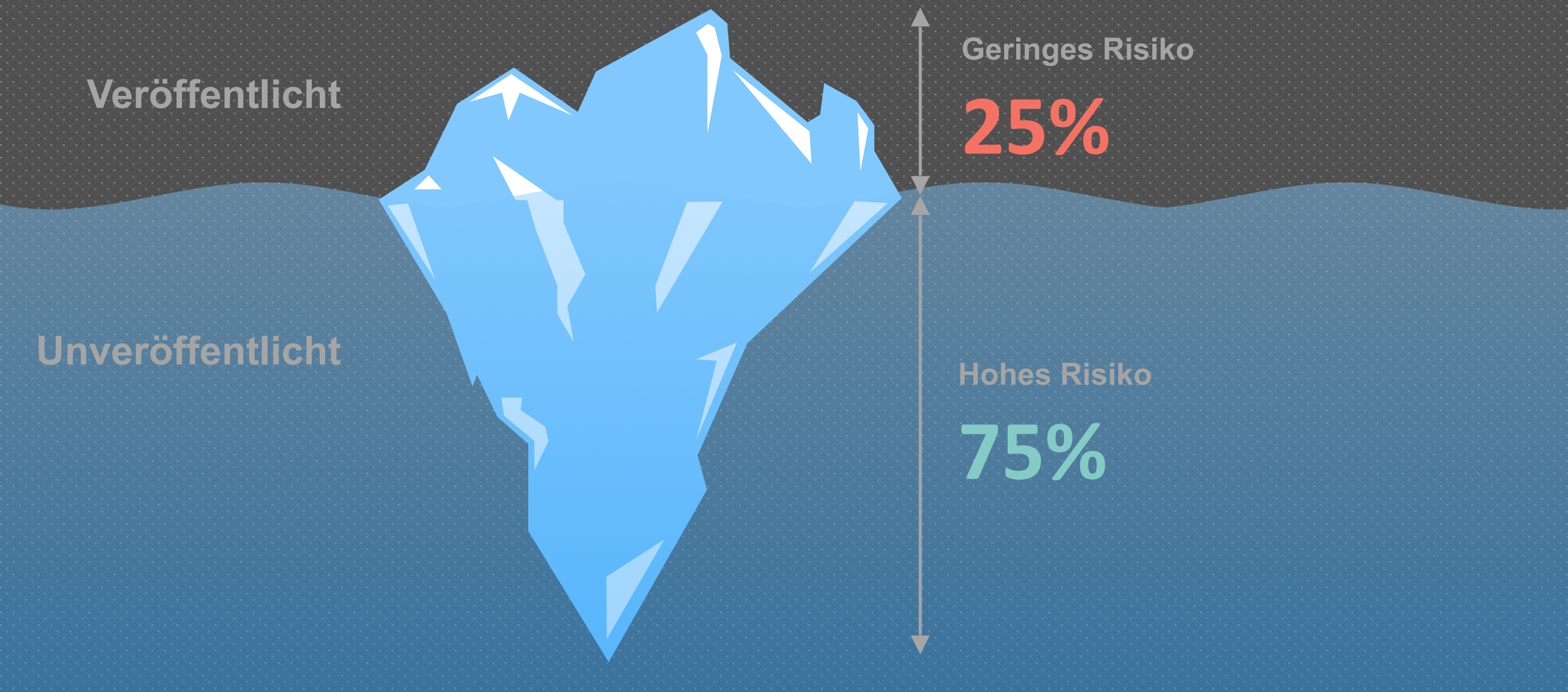


Verbesserungen

Schnelles Umsetzen des Feedbacks hilft auch beim Lernprozess.

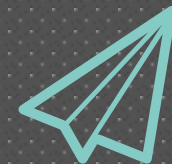
2. Risiken bei Deployment reduzieren

9



„Echte“ Fortschritte

- ✓ Regressionstests, Integrationstests, Akzeptanztests
- ✓ Aufsetzen der Infrastruktur
- ✓ Performanztests und Security Audits



Wie passt CD in Agile Projekte?

Continuous Delivery muss gelebt werden

Step-by-Step Process

12



Iterationen

Jedes Release besteht aus 1-n Iterationen



Stories

Jede Iteration besteht aus 1-n User Stories



Commits

Jede User Story besteht aus 1-n Commits

Step

Description

Ein Release

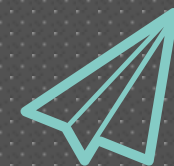


besteht somit immer aus einer Anzahl Commits



CD dreht den Sachverhalt

- ✓ Jeder Commit ist potenziell ein Release
- ✓ Somit ist ein Release eine Business Decision
- ✓ Bedingt Prüfung jedes Commits

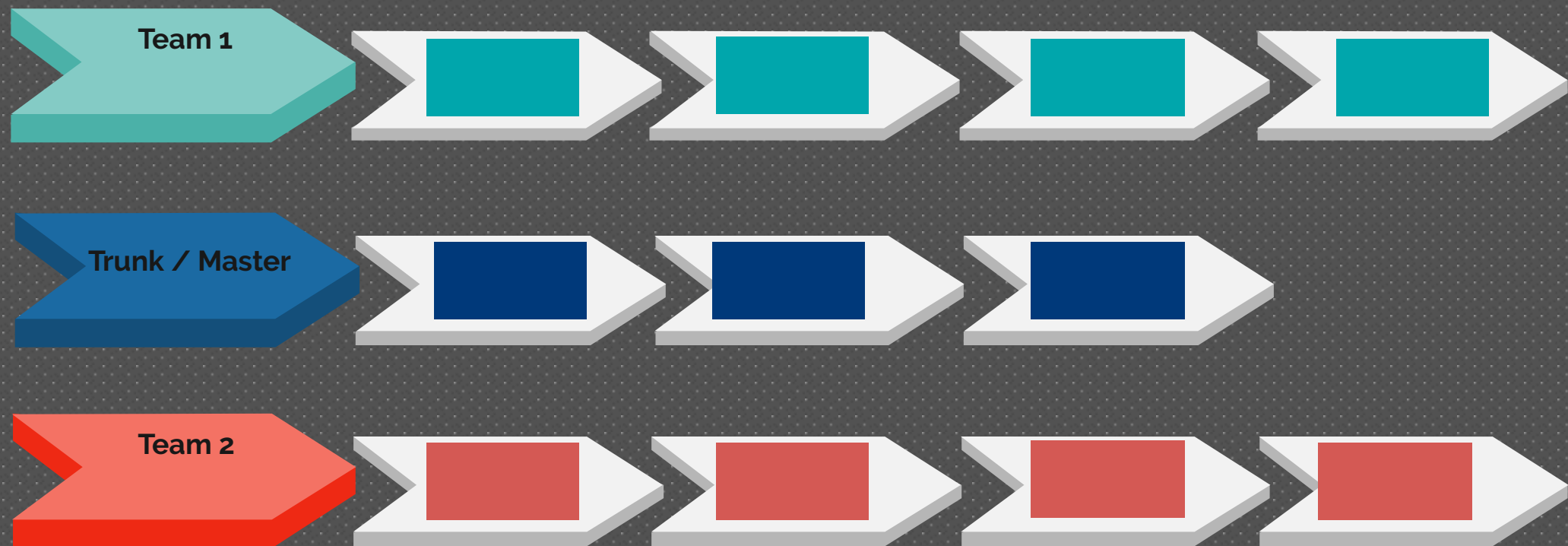


Wie wird meine Codebasis gemanagt?

Praxisrelevante Hinweise

Verschiedene Entwicklungsstränge

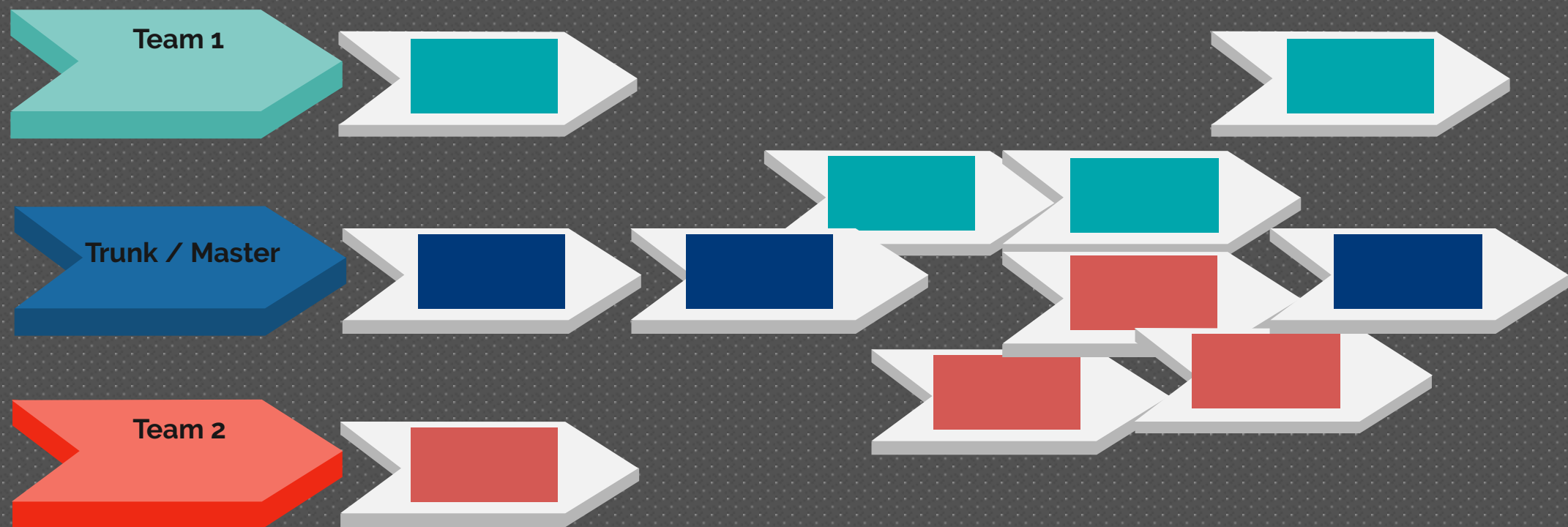
16



Wie integrieren verschiedene Teams ihre Entwicklungen mit Continuous Delivery?

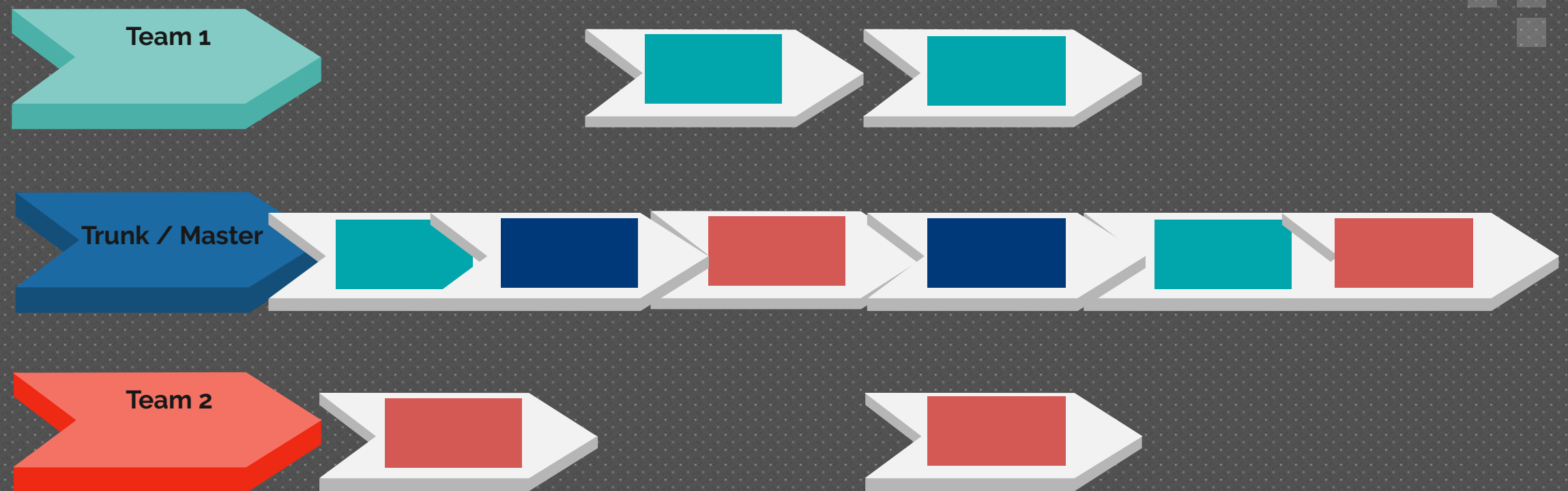
Mergekonflikte?

17



Wie integrieren verschiedene Teams ihre Entwicklungen mit Continuous Delivery?

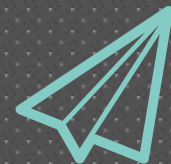
Keine Konflikte durch CD



Wie integrieren verschiedene Teams ihre Entwicklungen mit Continuous Delivery?

Best Practices

- ✓ Jeden Tag auf master committen
- ✓ Nach jedem Commit laufen automatisierte Tests
- ✓ Nur lokale Branches verwenden.



Wie werden Zwischenstände ausgeliefert?

Unvollständige Features müssen eventuell von Nutzern getestet werden können

Feature Toggles

- ✓ Konfigurationsdateien
- ✓ Features freischalten

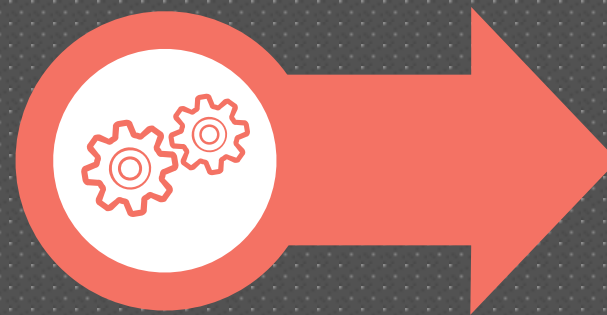
Branch Abstraction



Application

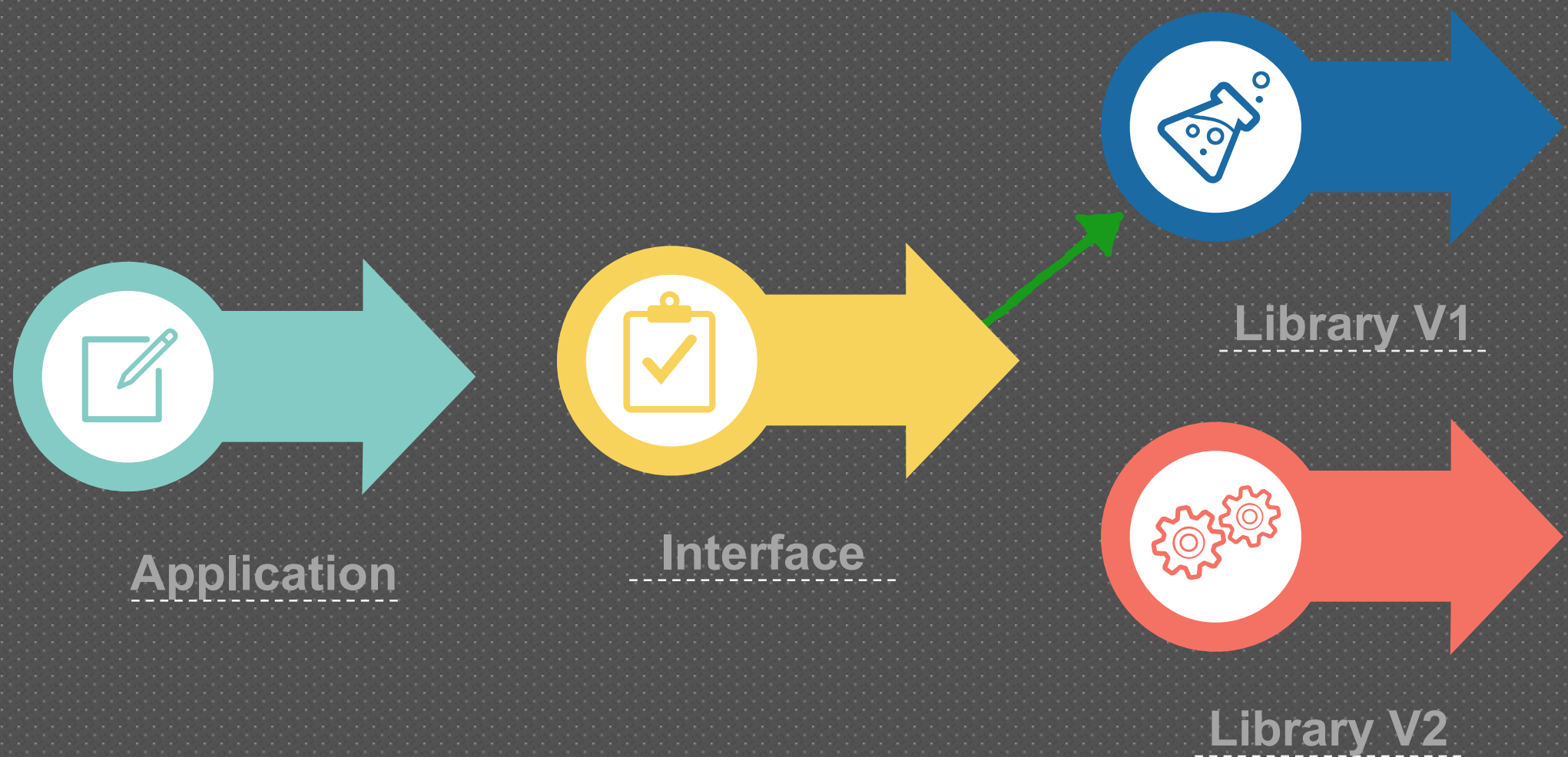


Library V1

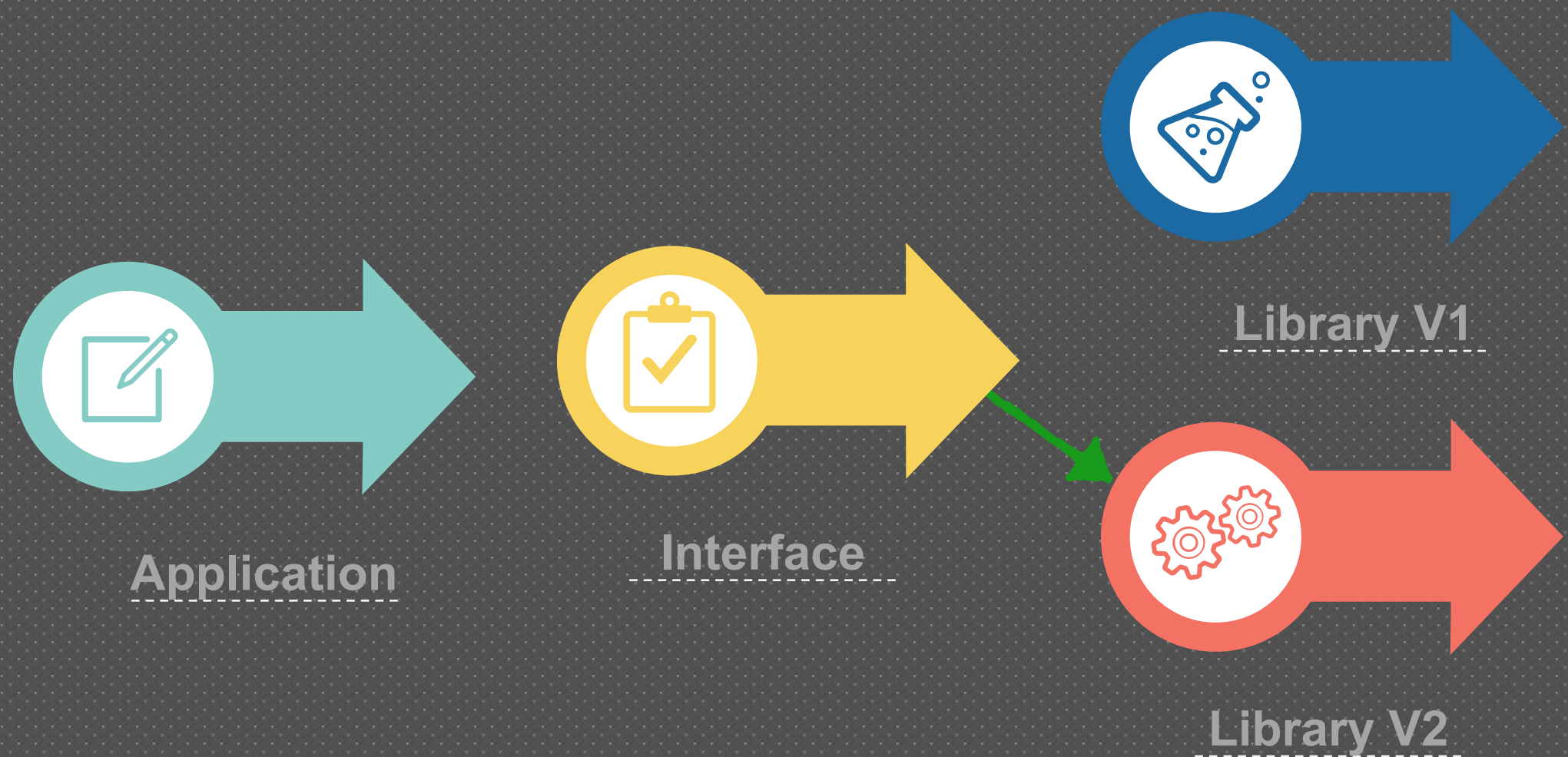


Library V2

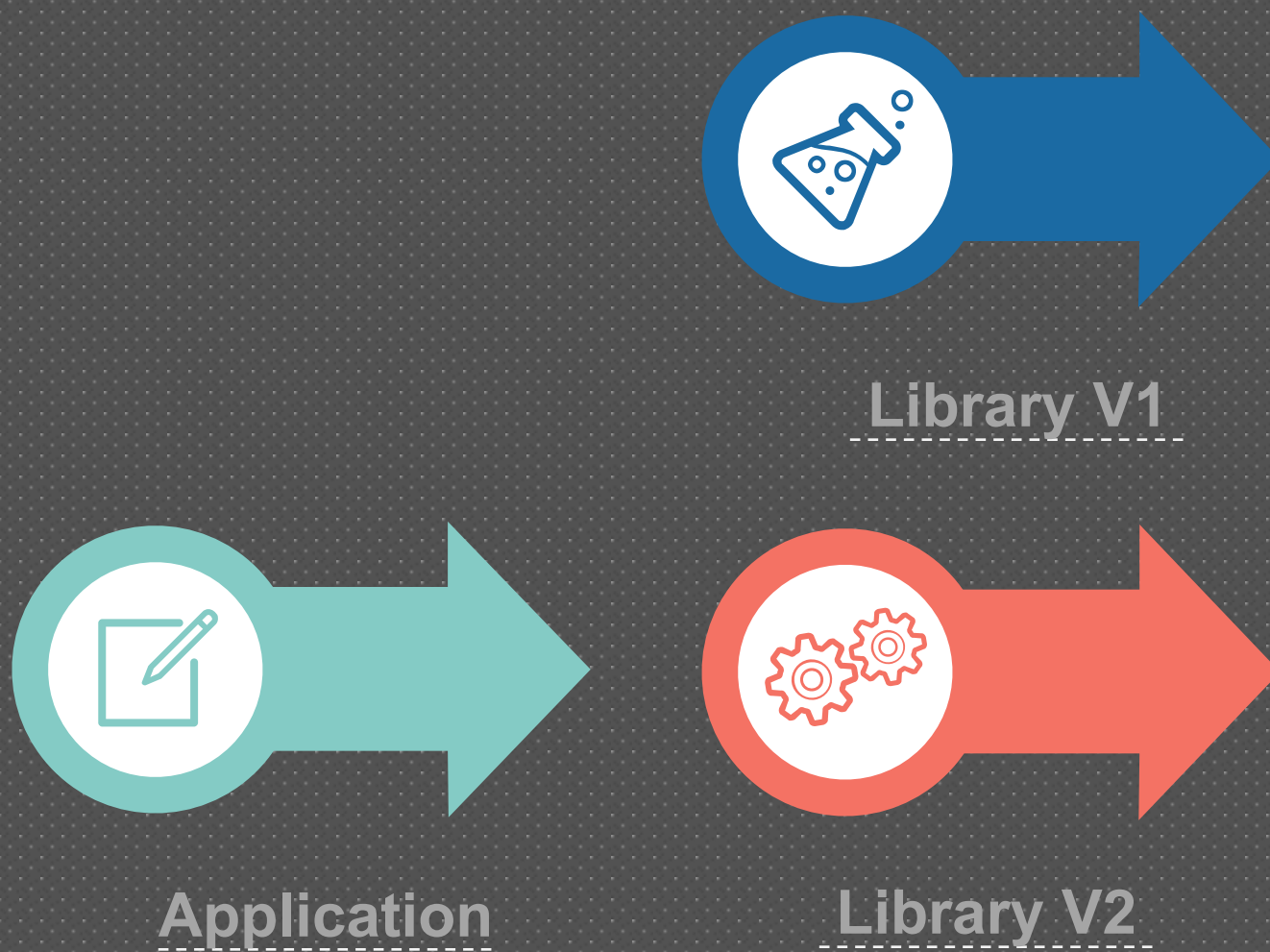
Branch Abstraction

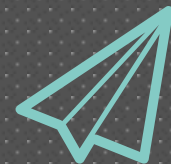


Branch Abstraction



Branch Abstraction





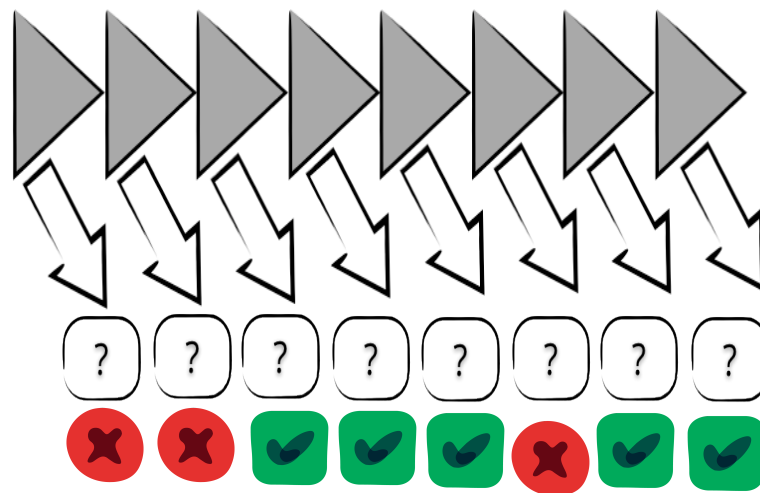
Pipeline für progressives Testen

Schritt für Schritt zum Erfolg

Pipelines



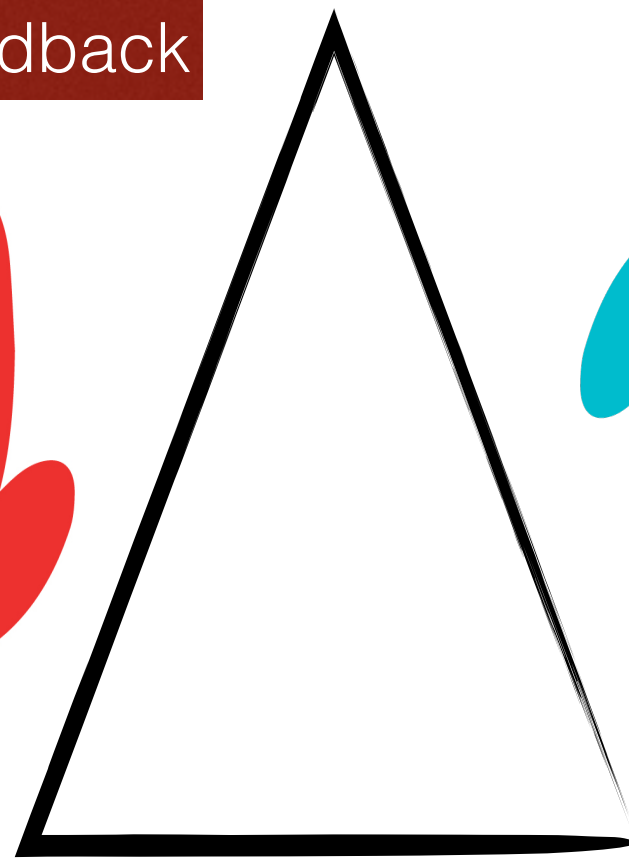
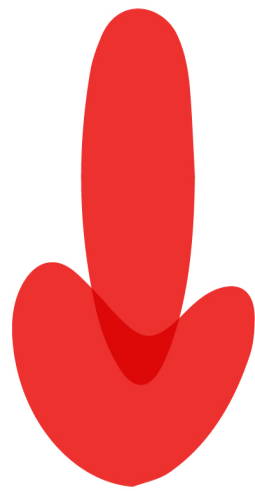
Jeder Build ist ein Releasekandidat



Test Pyramide

28

Schnelleres Feedback

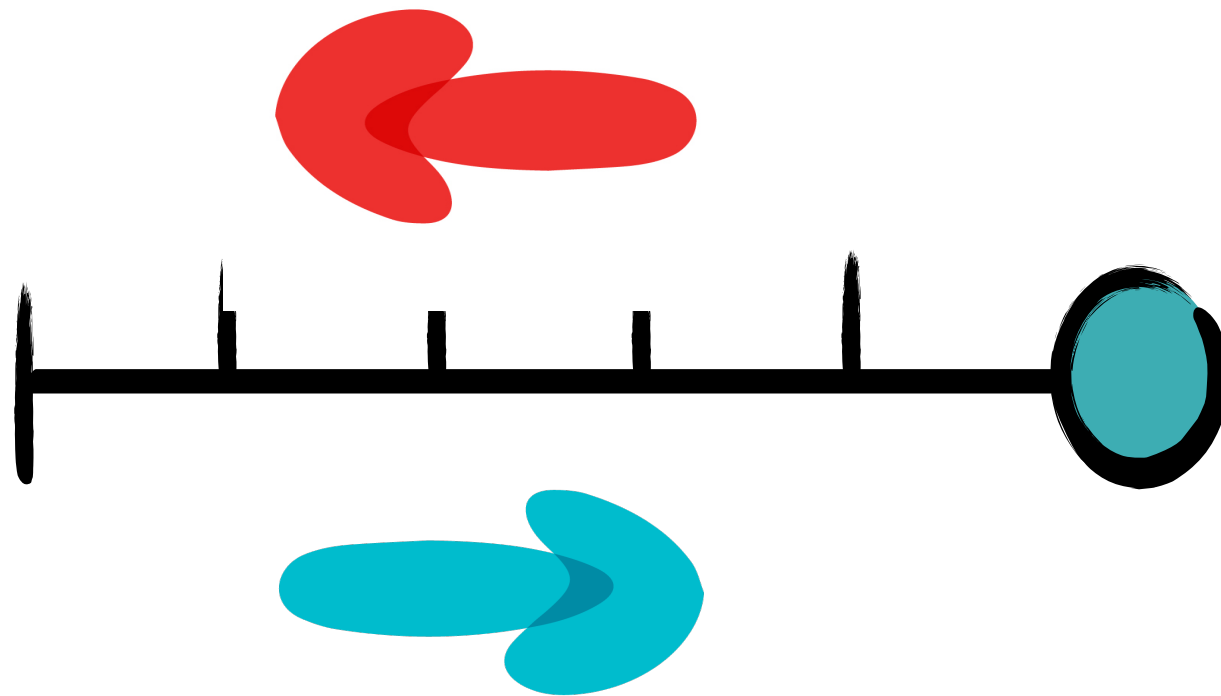


Größerer Scope

CD Pipeline

29

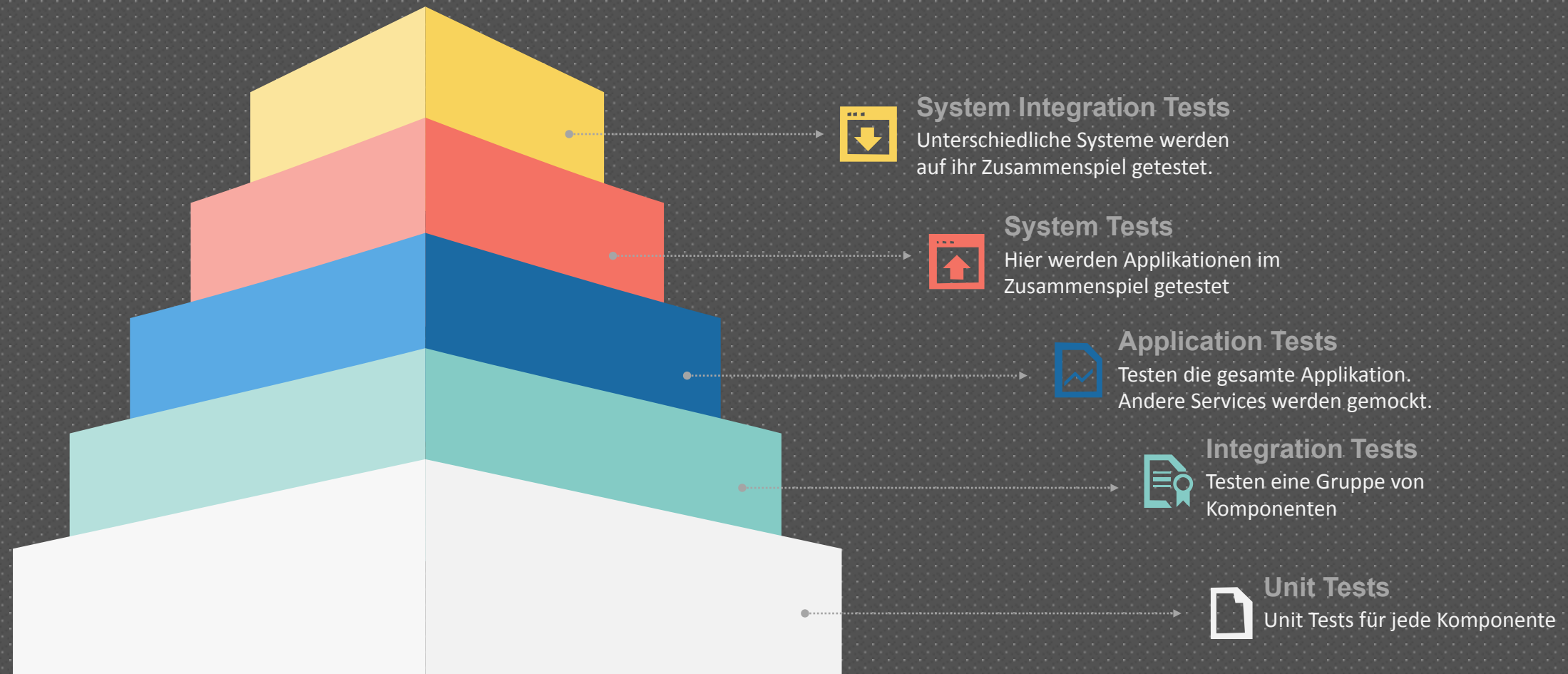
Schnelleres Feedback



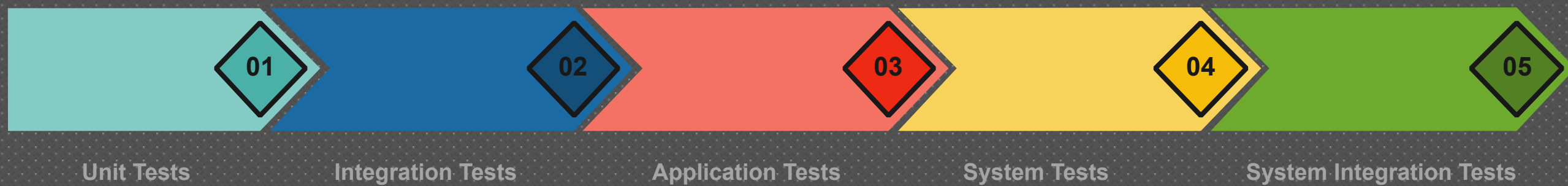
Zunehmende Zufriedenheit

Test Pyramide

30



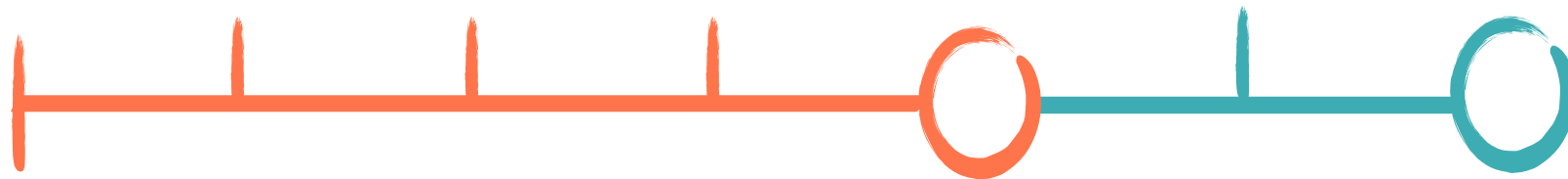
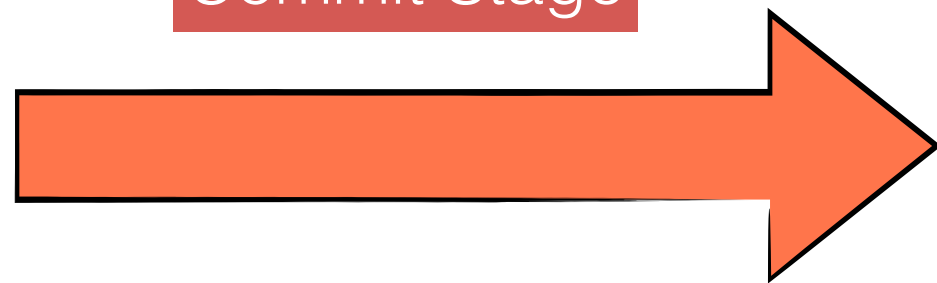
CD Pipeline



31

CD Pipeline

Commit Stage



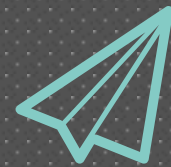
One Click Deployment

Typischer Commit Stage

- ✓ Push an das Reposiory
- ✓ Build Server pulled und baut
- ✓ Unit Tests
- ✓ Deploy

Best Practices für Pipelines

- ✓ Alles im Repository halten
- ✓ Artefakte nur einmal bauen
- ✓ Immer auf die selbe Art deployen
- ✓ Produktionsnähe schaffen



Strategien für das Deployment

Schritt für Schritt zum Erfolg

**” *Risikoreduzierte Deployments sind
inkrementell***

Die Erfahrung

Warum?

Risikoreduzierte Deployments sind inkrementell



Monolithische Deployments

Riesige Deployments mit vielen Abhängigkeiten, Datenbankänderungen können unangenehme Seiteneffekte haben



Inkrementelle Deployments

Sparen Geld und Nerven, denn sämtliche Abhängigkeiten und Features sind zuvor ausgiebig getestet worden



Beispiele für inkrementelle Deployment Muster

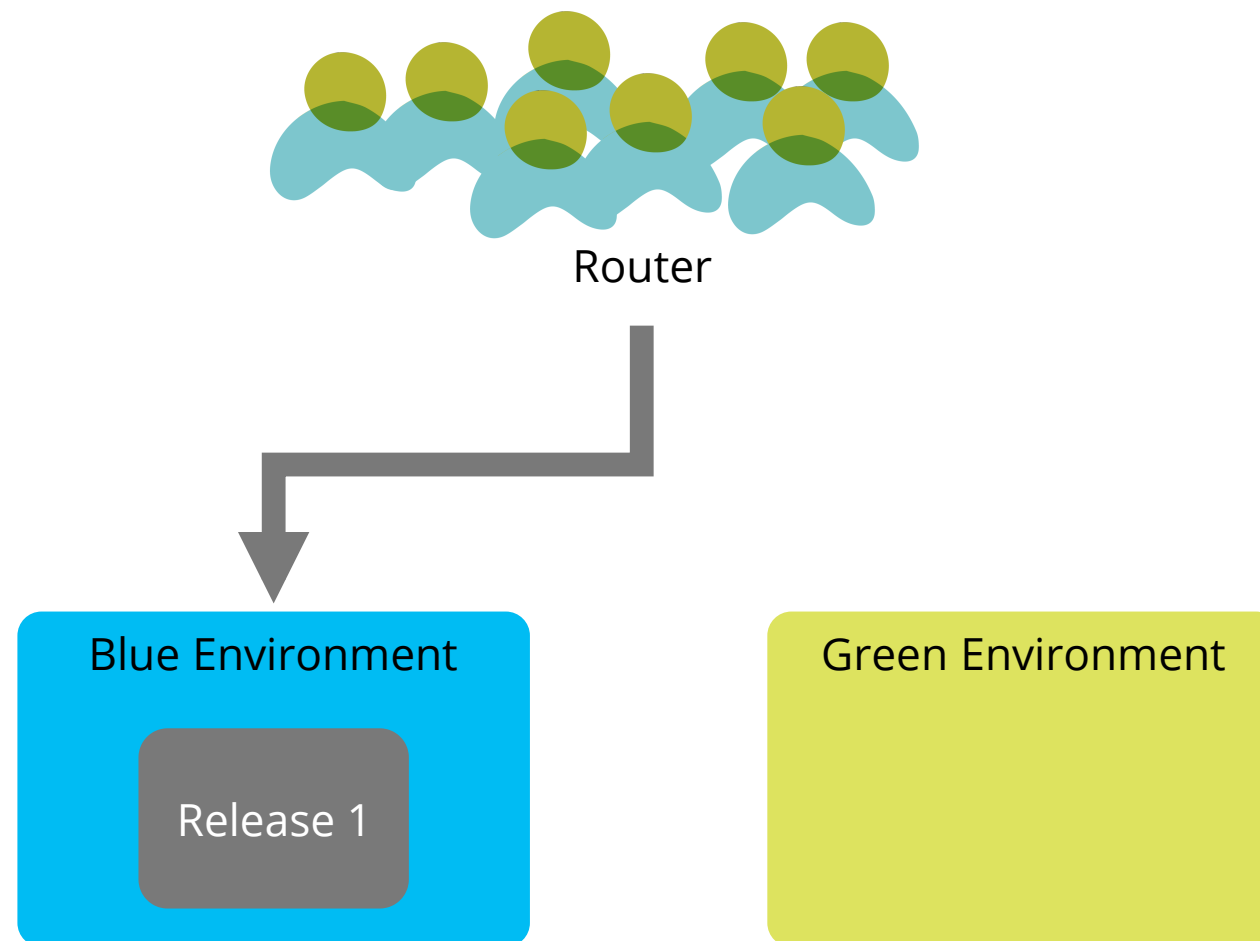
Sehen wir uns einige Beispiele hierfür an



BLUE-GREEN DEPLOYMENT MUSTER

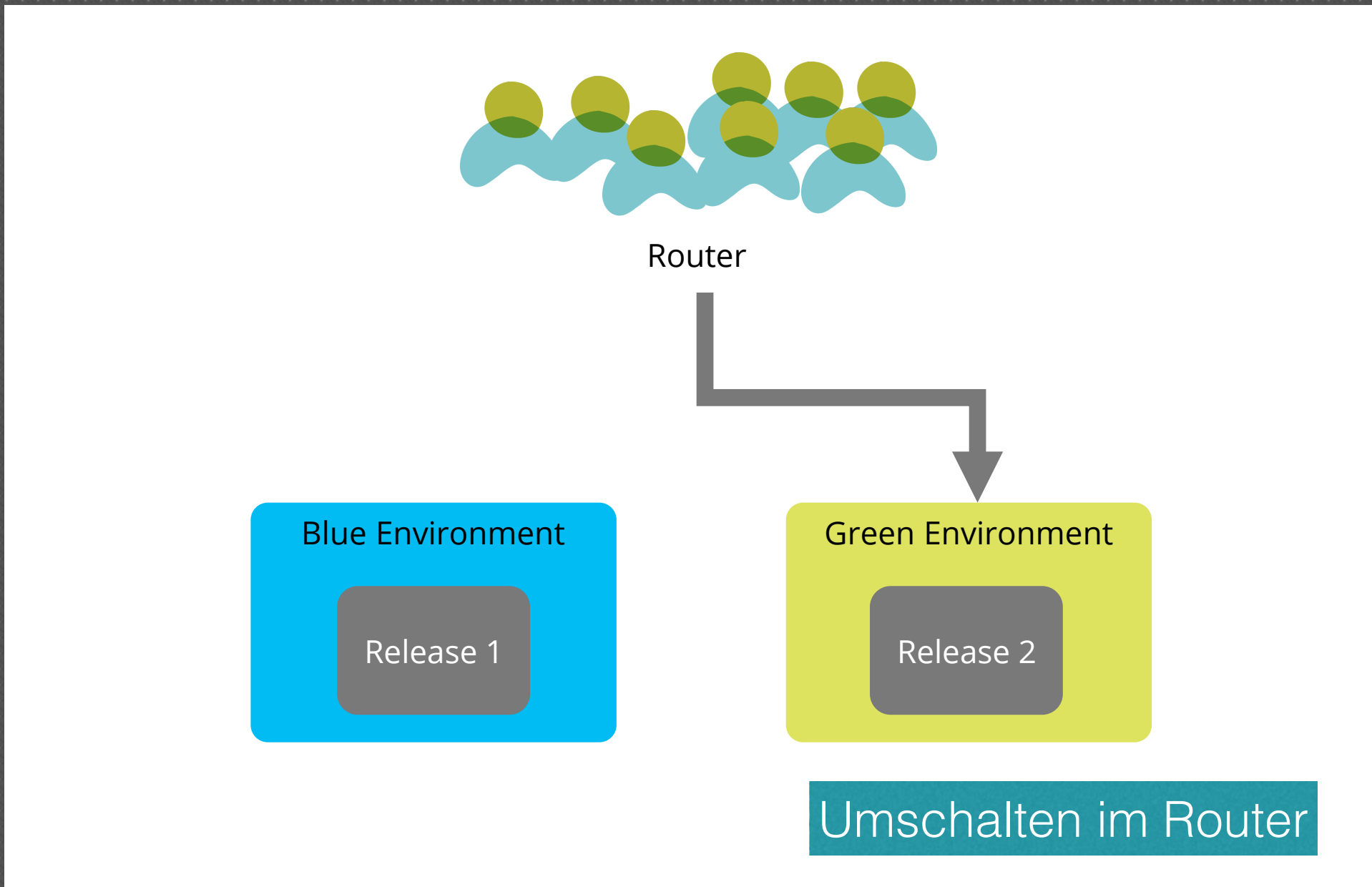
— BLUE-GREEN

39



Nur eine Produktivumgebung ist live

— BLUE-GREEN

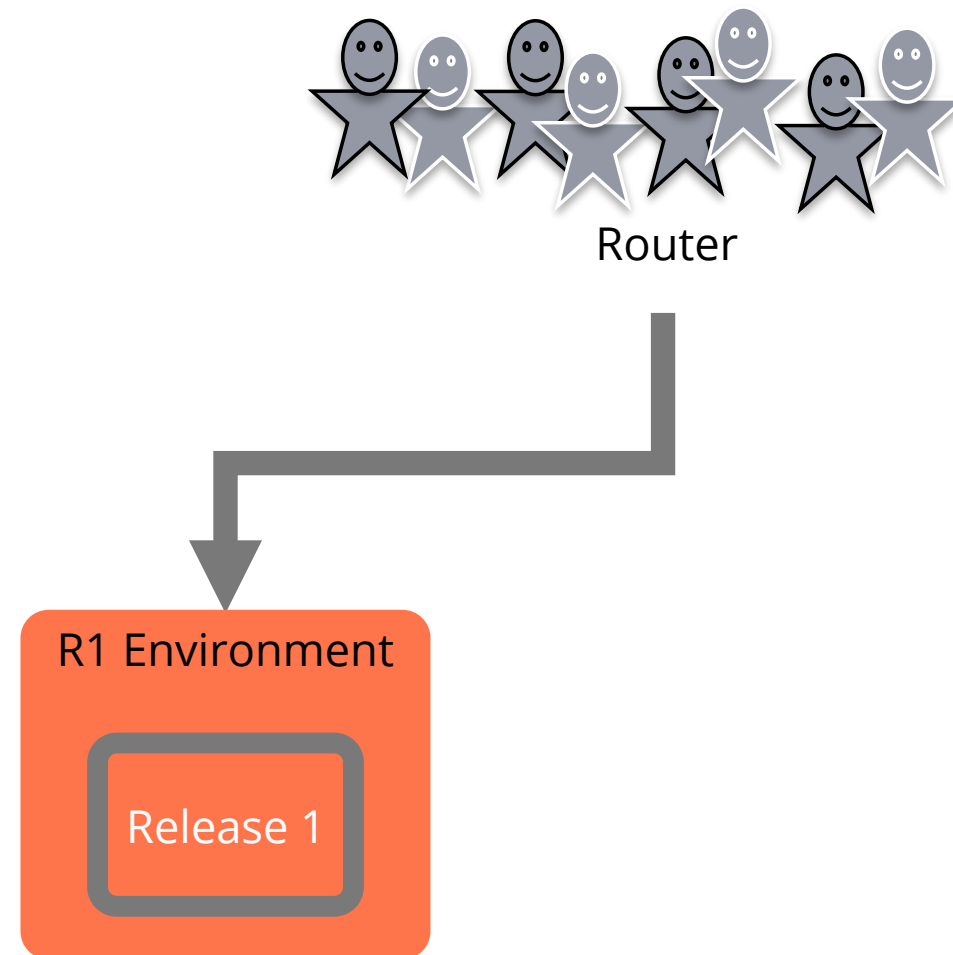


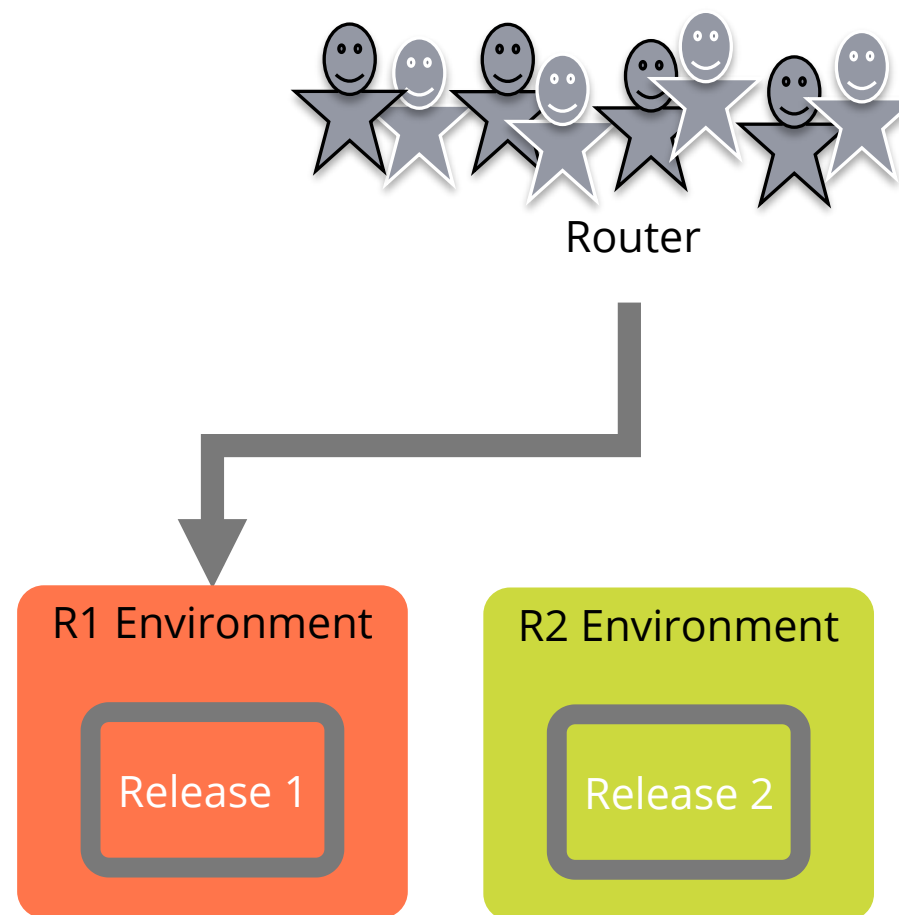


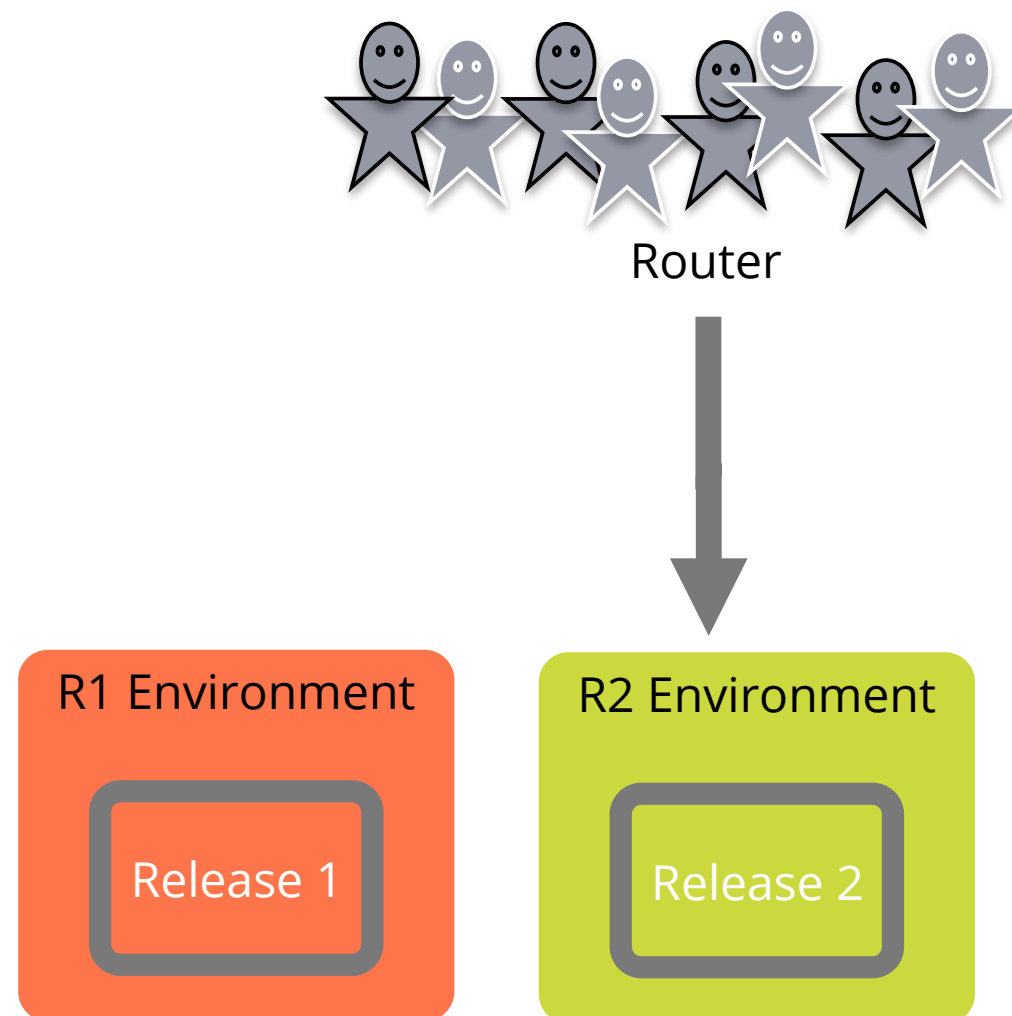
PHOENIX DEPLOYMENT MUSTER

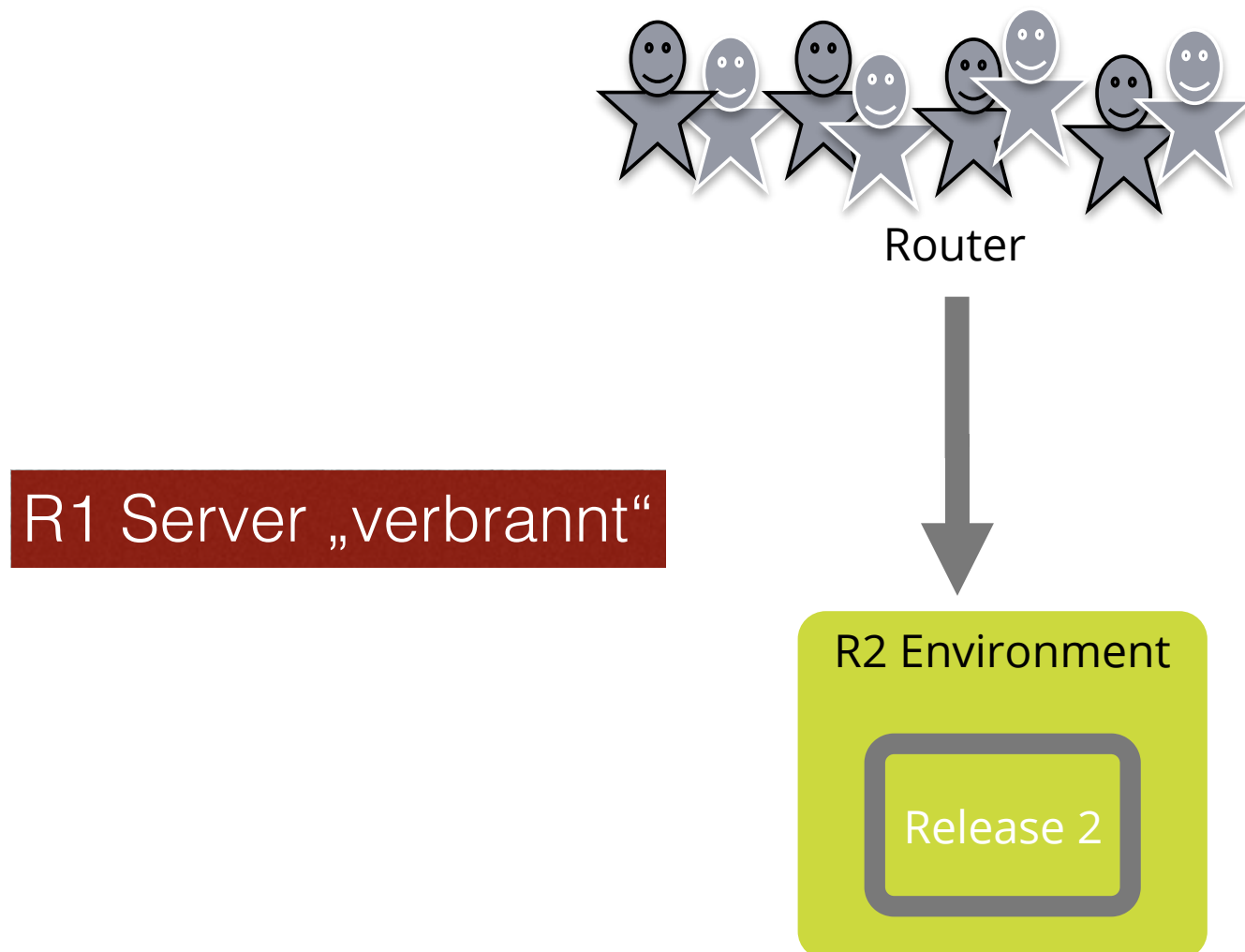
Phoenix Deployment Muster

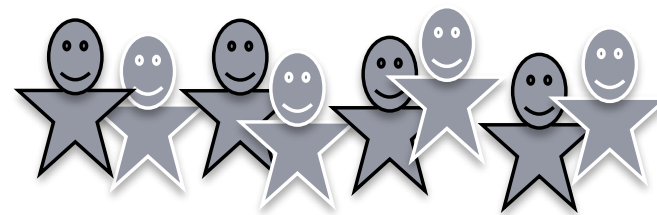
- ✓ Phoenix server werden „verbrannt“
- ✓ So werden Konfigurationsabweichungen reduziert
- ✓ Frische Server sind zuverlässiger
- ✓ Besonders mit Docker einfach











Router



Prozess weiterführen mit R3

R2 Environment

Release 2

R3 Environment

Release 3



ENVIRONMENT PROMOTION DEPLOYMENT MUSTER

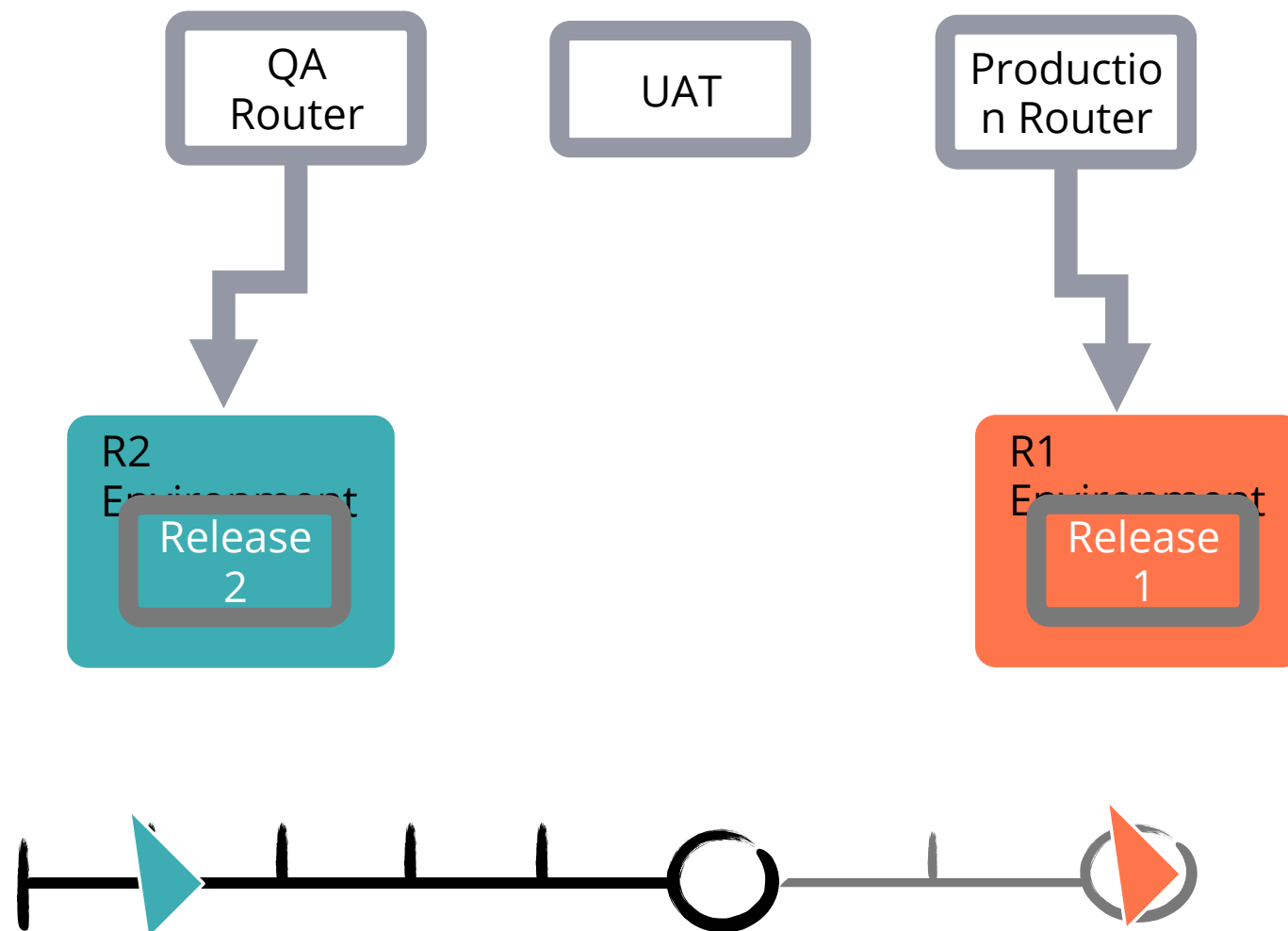
Environment Promotion

- ✓ Für jedes Deployment wird die Umgebung neu aufgesetzt
- ✓ Die Umgebung wird in der Pipeline weitergereicht.
- ✓ Stellt sicher, dass die gesamte Umgebung getestet ist.

ENVIRONMENT PROMOTION

50

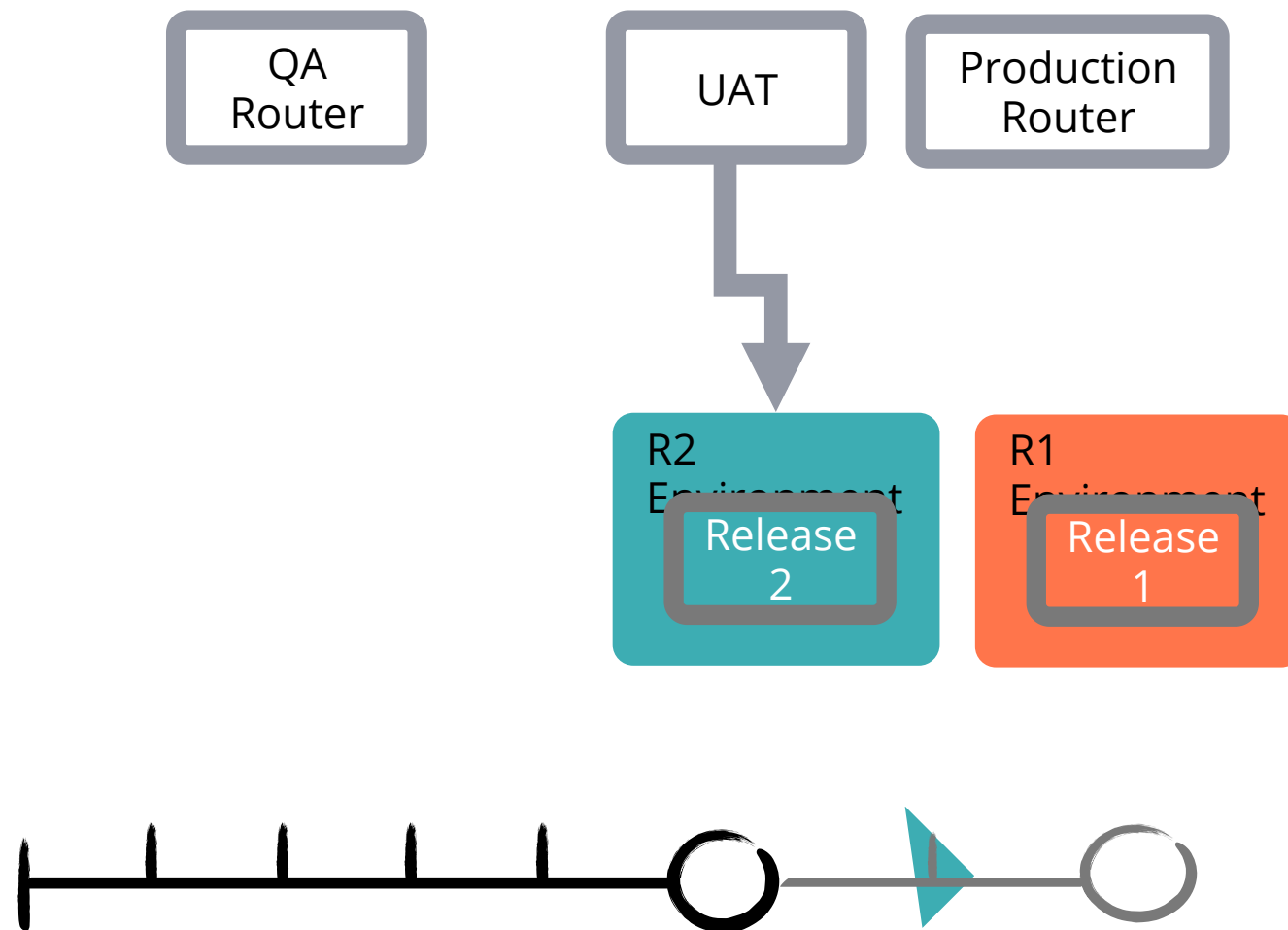
R2 wird im QA State getestet



ENVIRONMENT PROMOTION

51

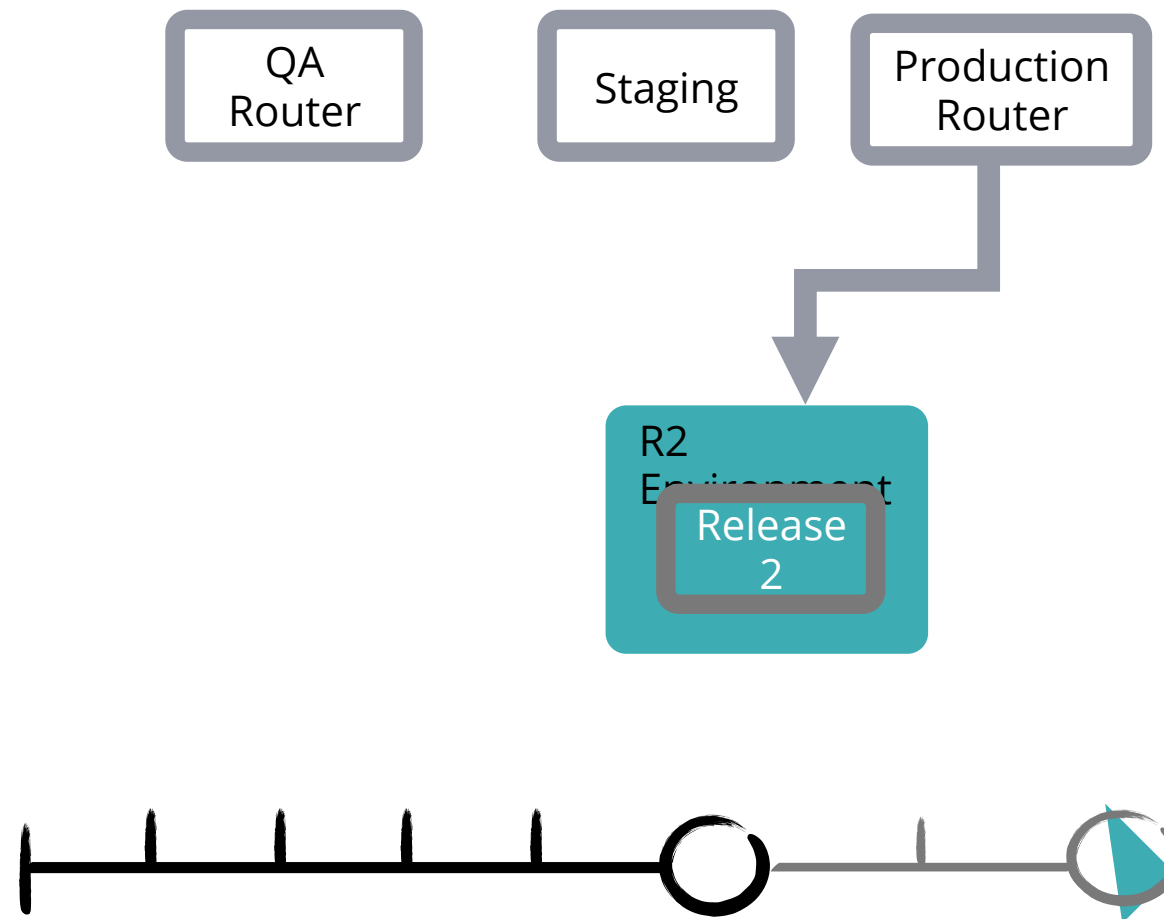
R2 wird im UAT getestet



ENVIRONMENT PROMOTION

52

R2 geht in Produktion und R1 wird zerstört





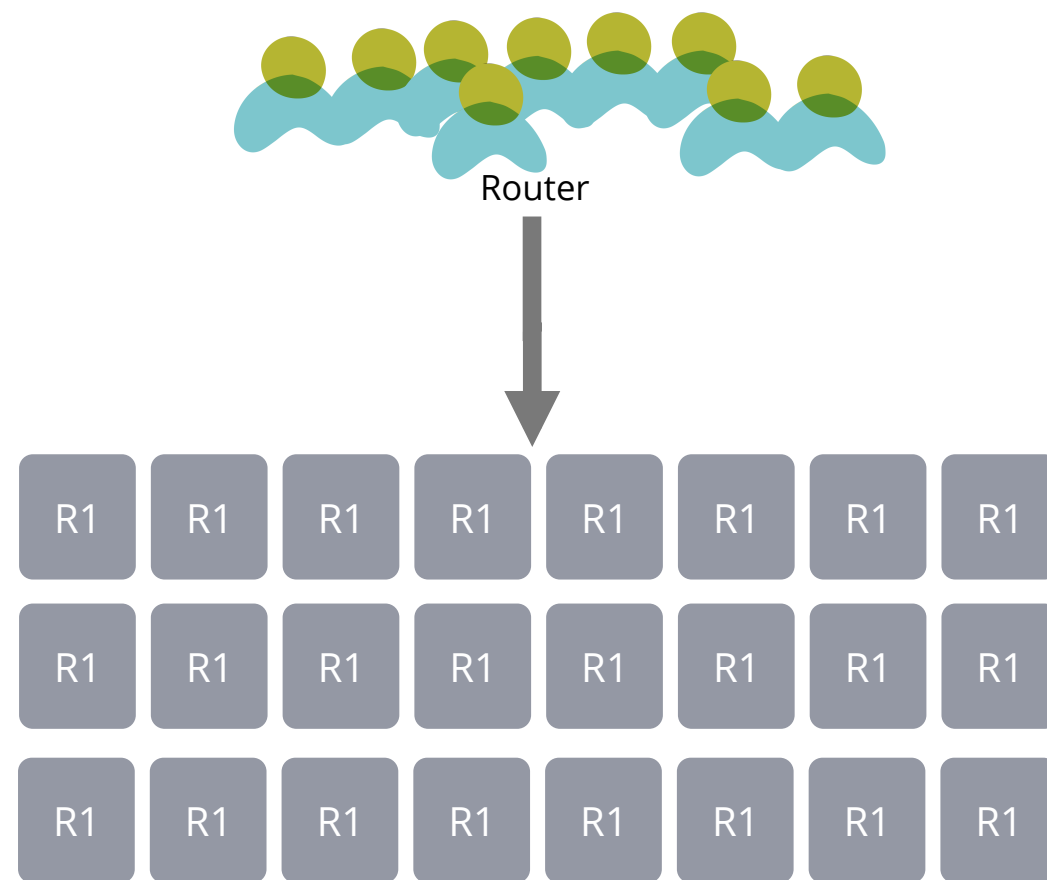
CANARY RELEASE DEPLOYMENT MUSTER

Canary Release

- ✓ Abwandlung von BLUE-GREEN für Clusterbetrieb
- ✓ Updates im Cluster werden schrittweise vorgenommen. Node by Node.
- ✓ Sichert Feedback einer kleinen Benutzergruppe

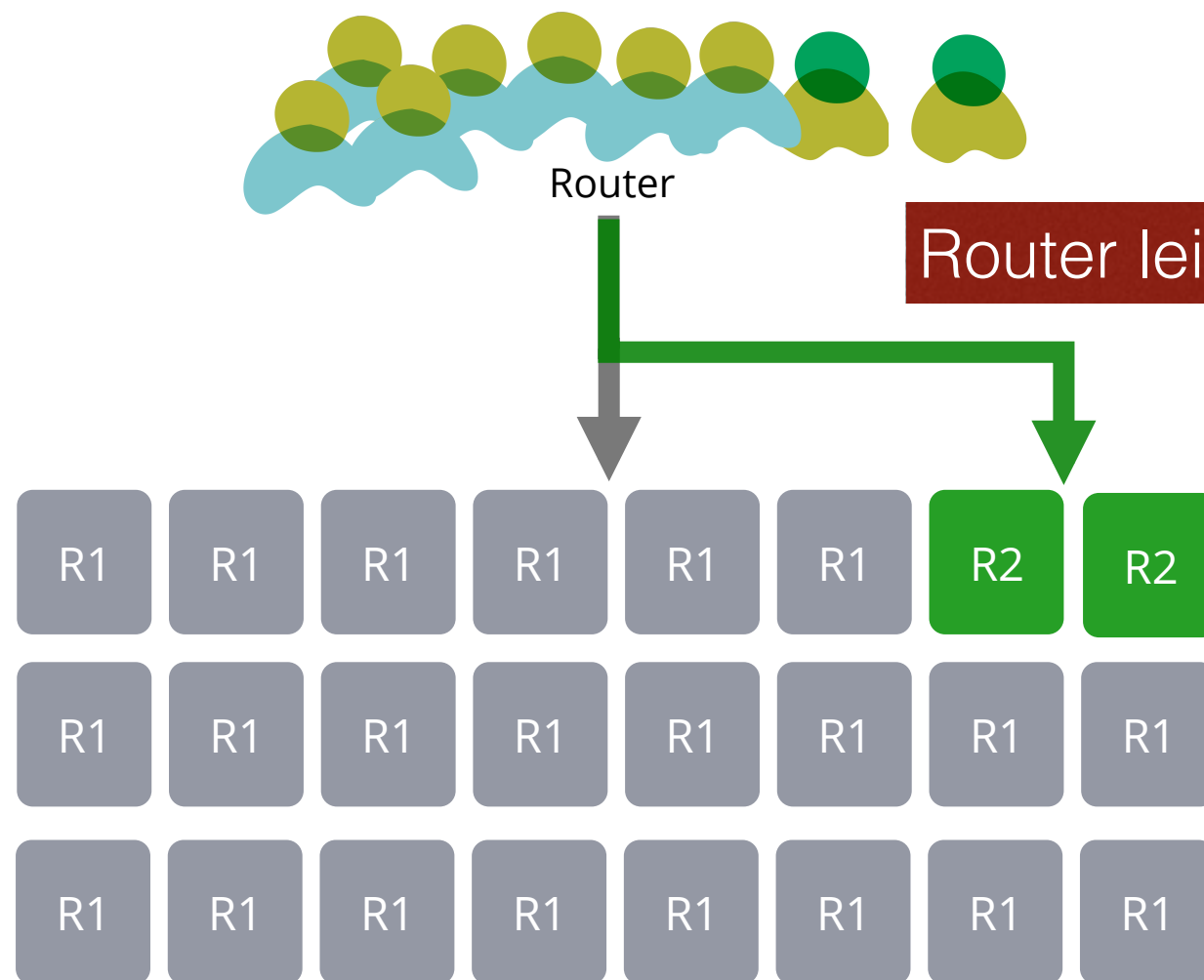
CANARY RELEASE

Servercluster



CANARY RELEASE

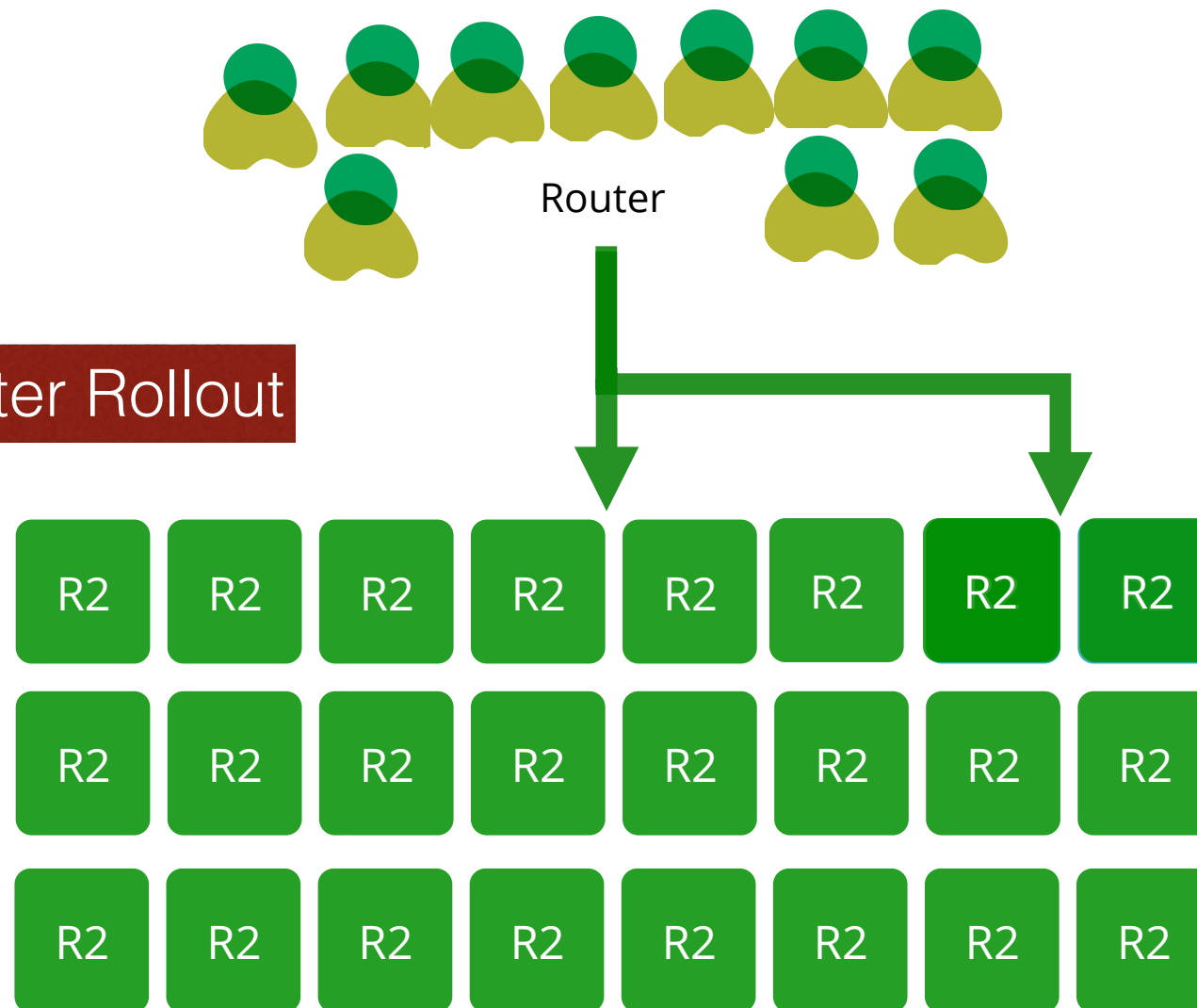
56

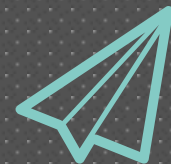


CANARY RELEASE

57

Kompletter Rollout





Tools für Continuous Deployment

Warum kompliziert, wenn es auch einfach geht



Jenkins



Travis CI