Continuous Delivery und Deployment

Dipl-Ing. (FH) Michael Johann

Warum Continuous Delivery?

Vor dem Wie klären wir erstmal das Warum

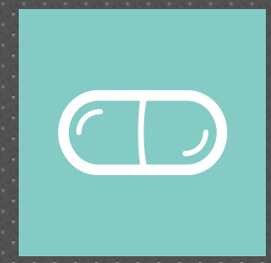
Warum CD?

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

Design

Aufnahme der Anforderungen

Definieren der Architektur

Entwicklung

Umsetzung der Anforderungen

Test

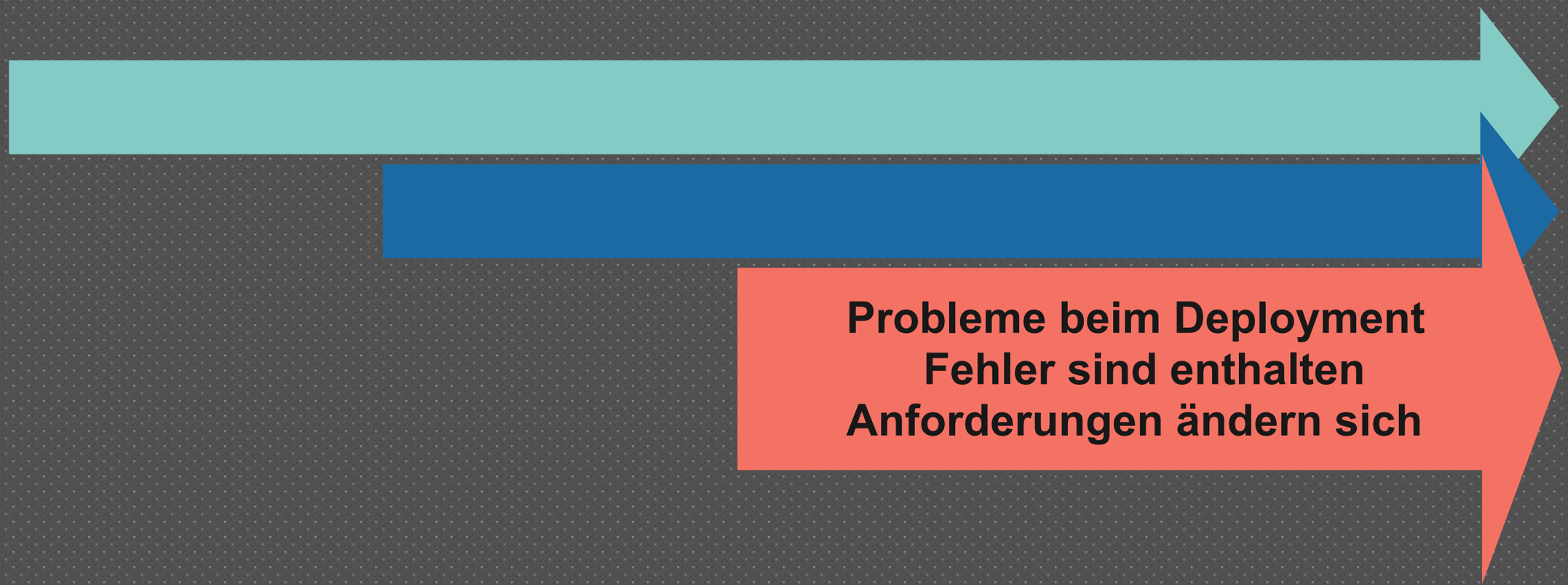
Test der Software

Release

Auslieferung der Software

In großen Unternehmen kann dies ein oder zweimal im Jahr üblich sein.

Die Realität



Die Realität

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.

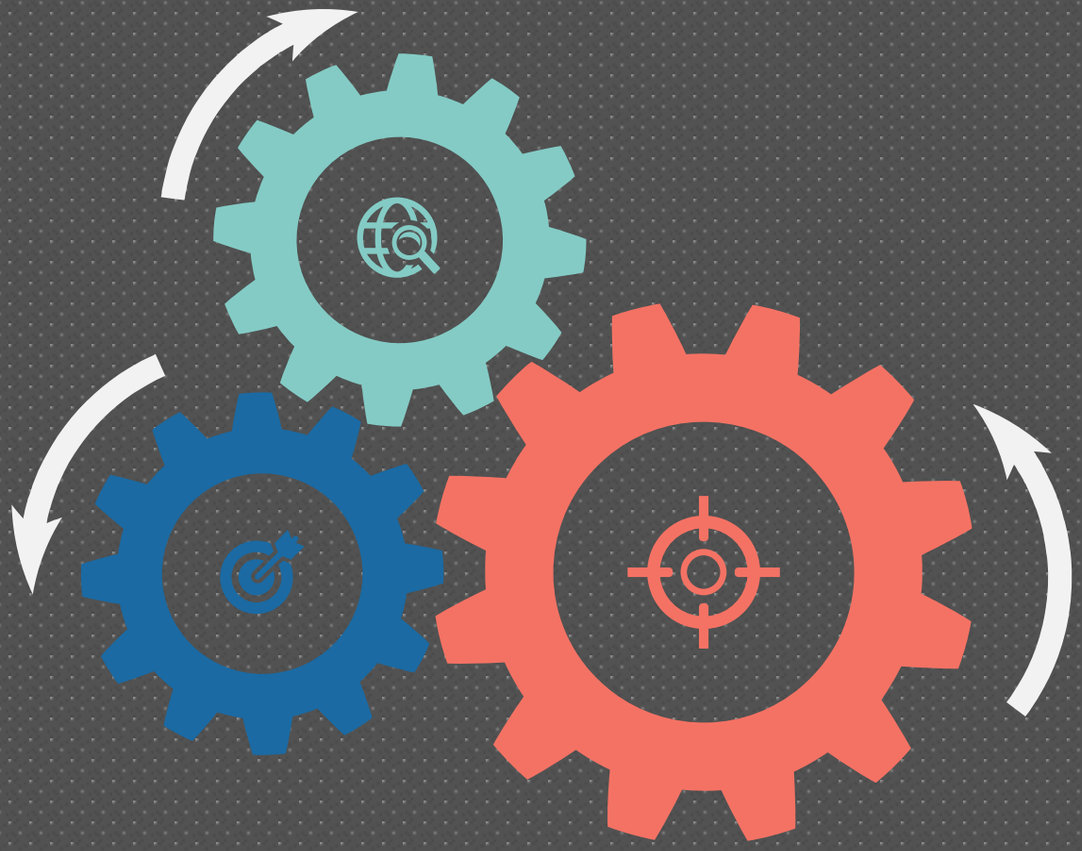
Probleme

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.

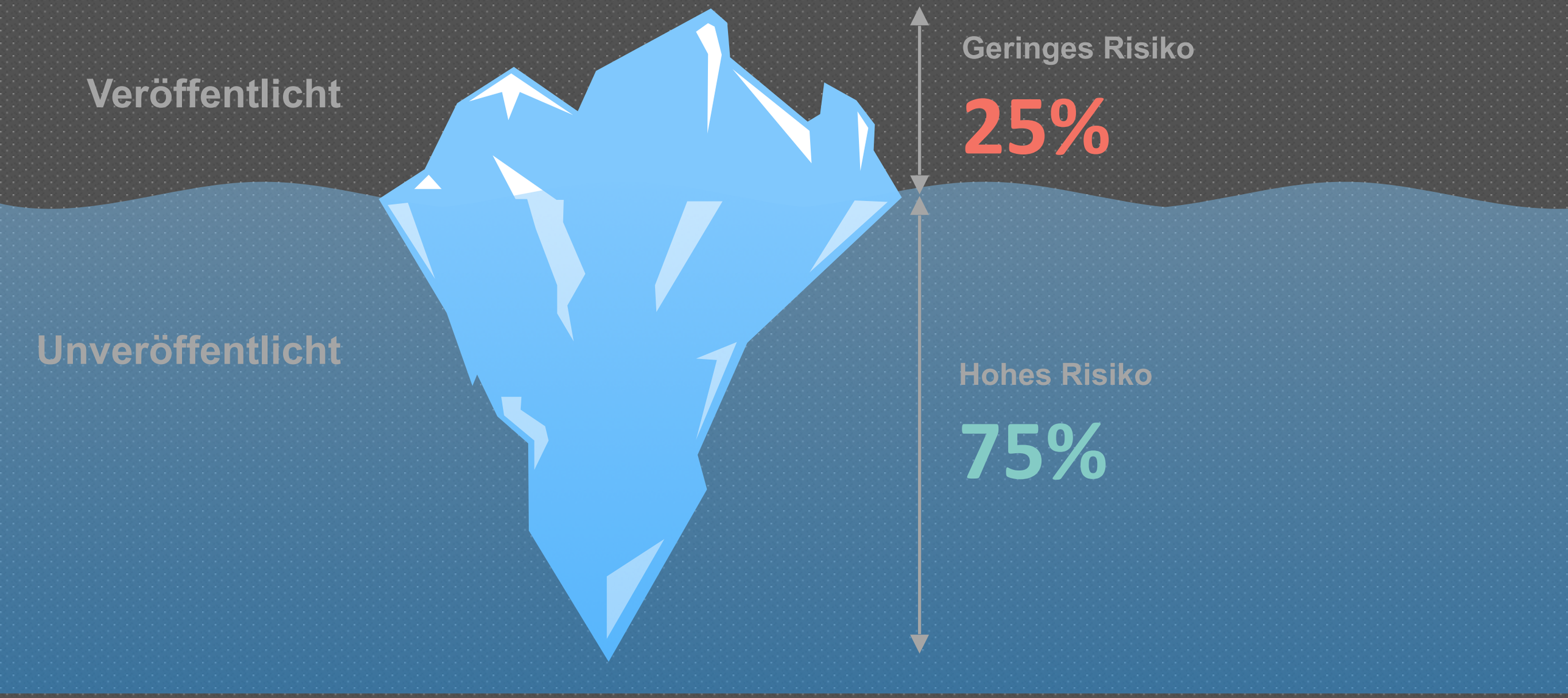
Schnelle Rückmeldungen

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



„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

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

Stufe

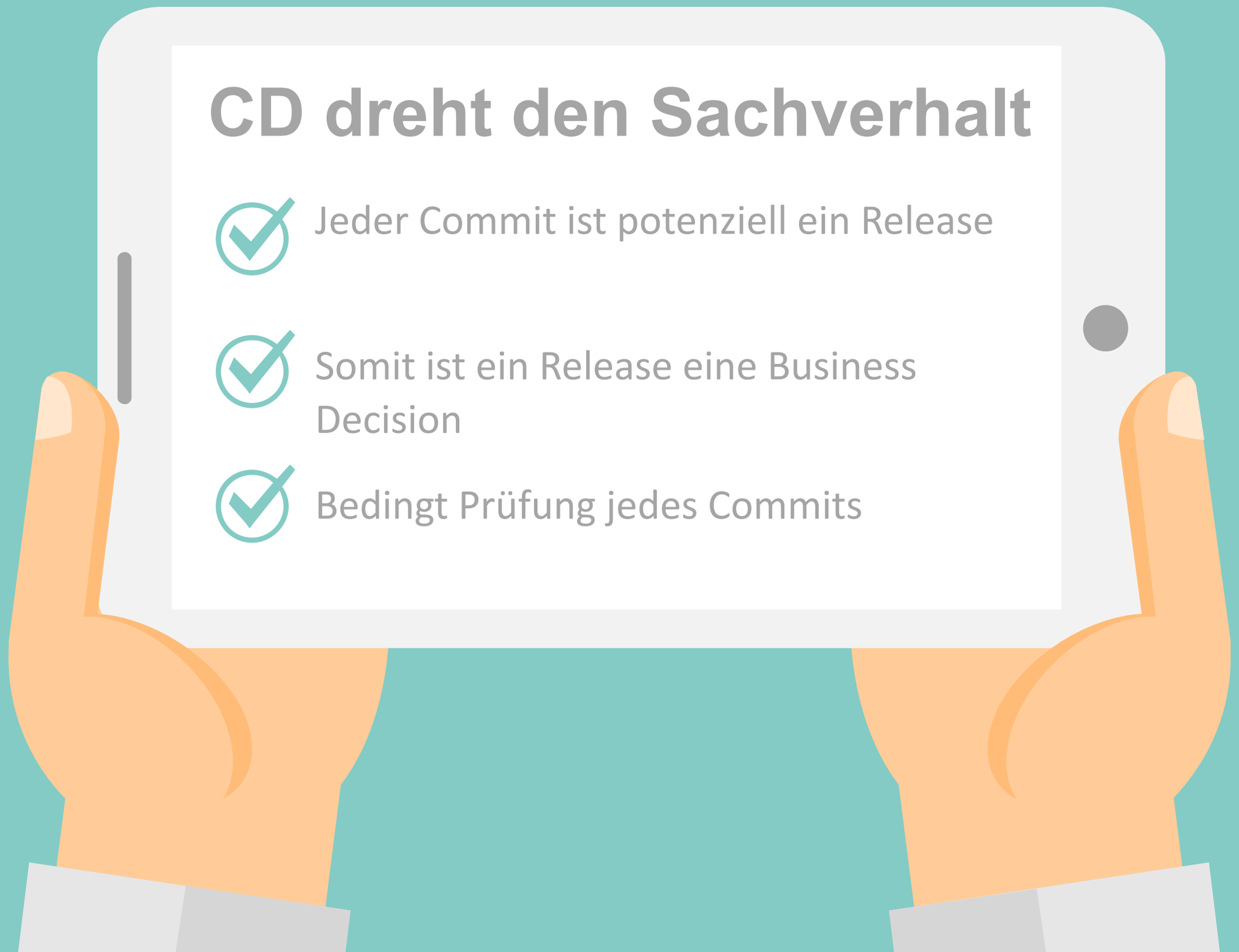
Beschreibung

Ein Release

- besteht somit immer aus einer Anzahl Commits

CD dreht den Sachverhalt

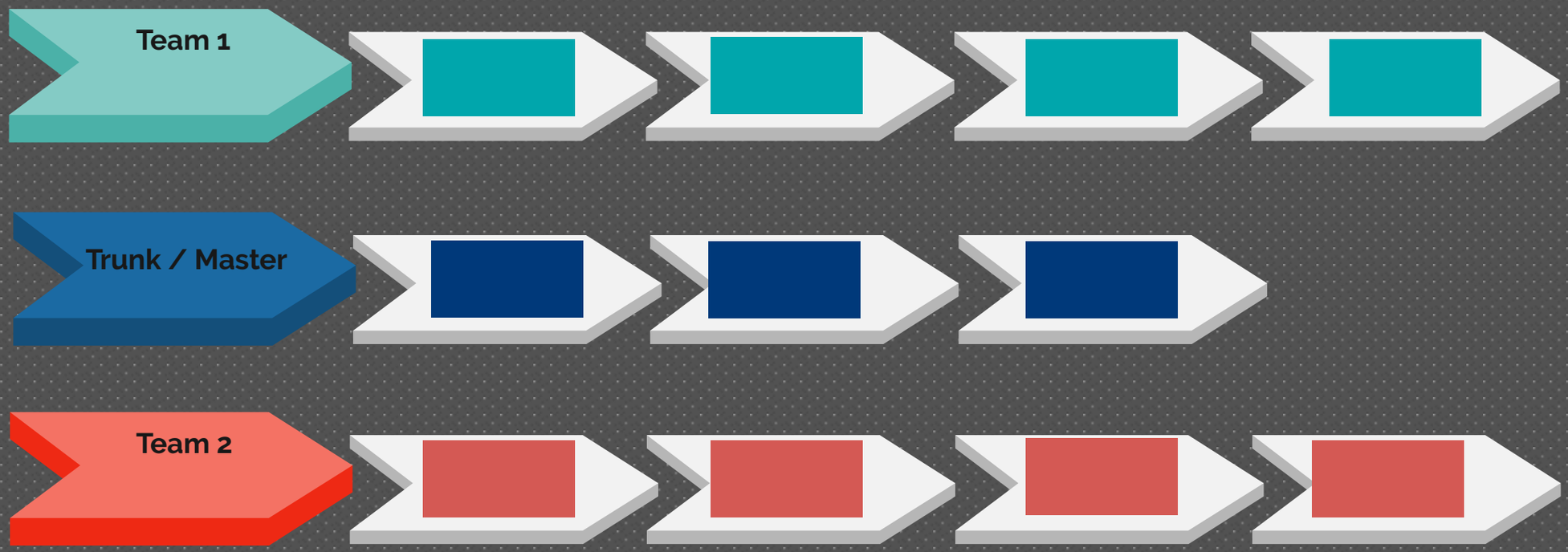
- Jeder Commit ist potenziell ein Release
- Somit ist ein Release eine Business Decision
- Bedingt Prüfung jedes Commits



Wie wird meine Codebasis gemanagt?

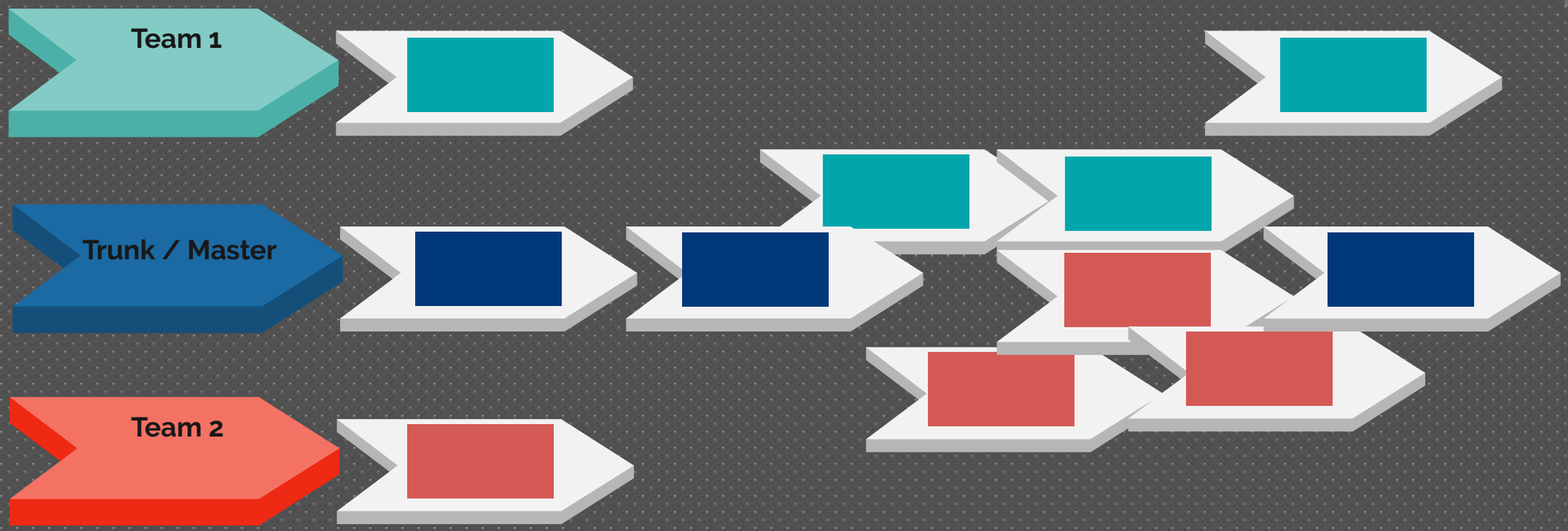
Praxisrelevante Hinweise

Verschiedene Entwicklungsstränge



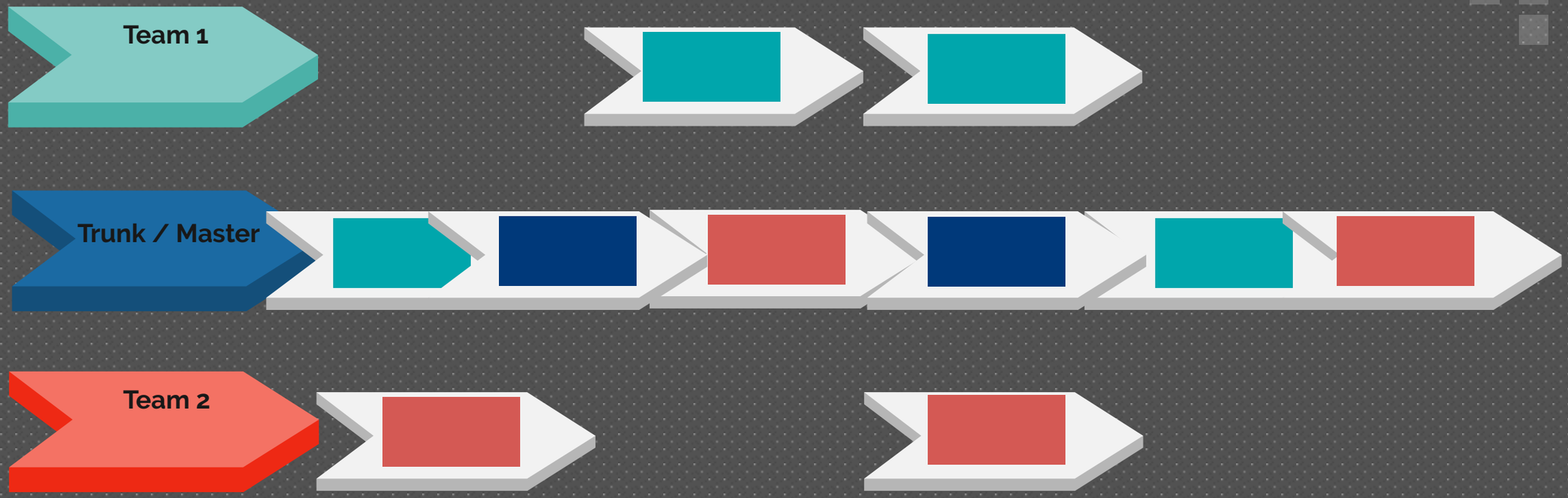
Wie integrieren verschiedene Teams ihre Entwicklungen mit Continuous Delivery?

Mergekonflikte?



Wie integrieren verschiedene Teams ihre Entwicklungen mit Continuous Delivery?

Keine Konflikte durch CD



Wie integrieren verschiedene Teams ihre Entwicklungen mit Continuous Delivery?

http://martinfowler.com/articles/continuousIntegration.html

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

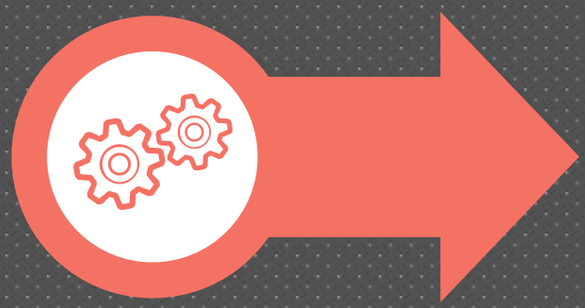
Feature Toggles

- Konfigurationsdateien
- Features freischalten

Branch Abstraction

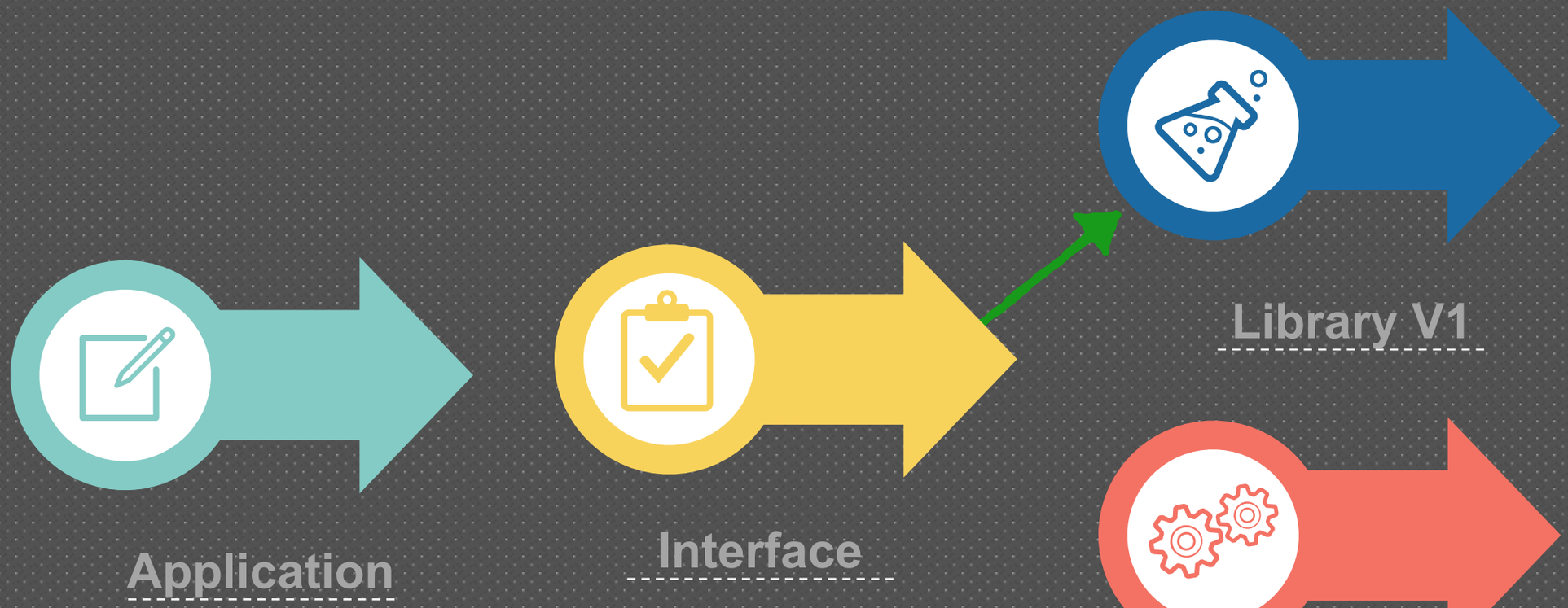
Application

Library V1

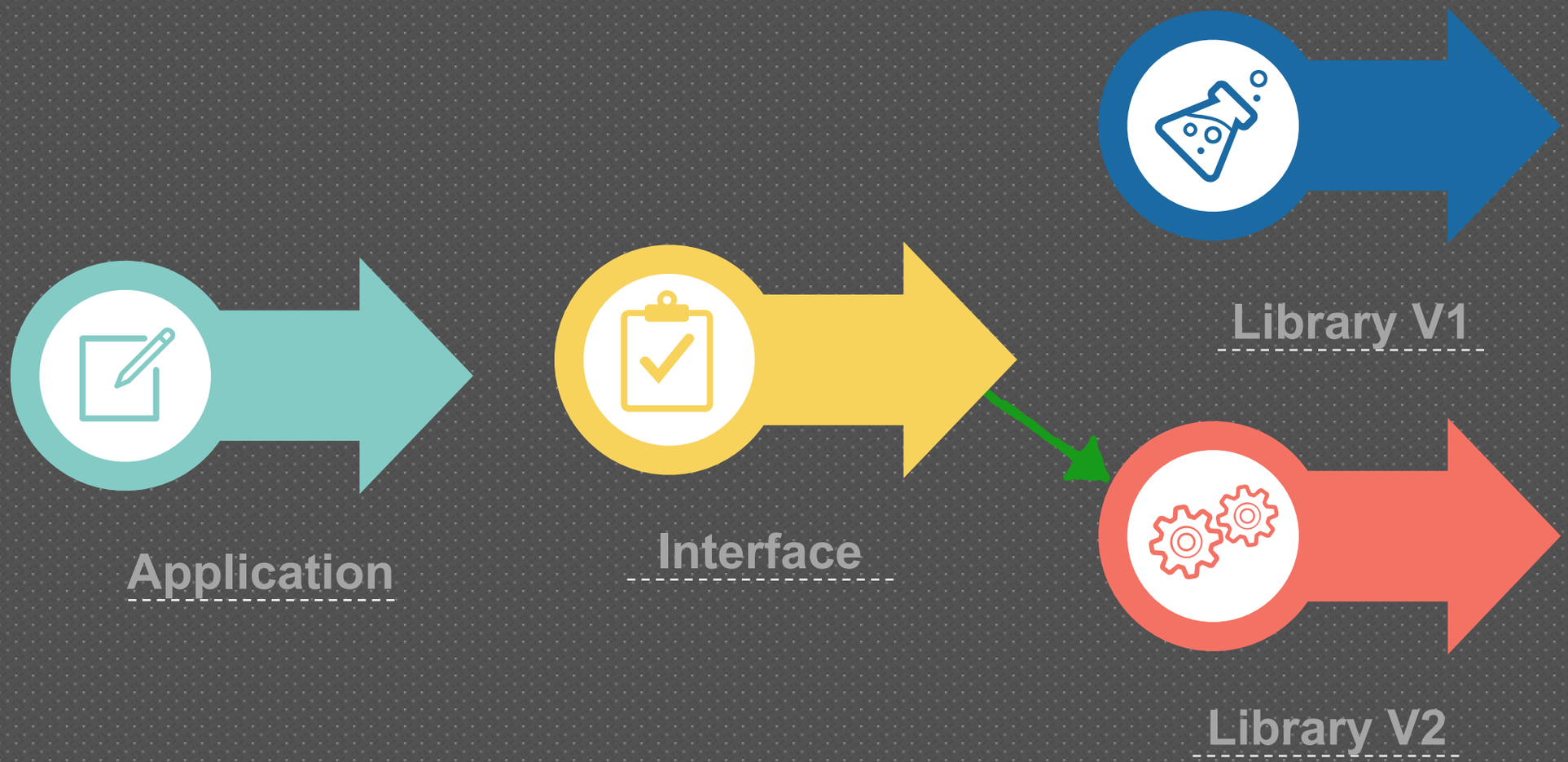


Library V2

Branch Abstraction

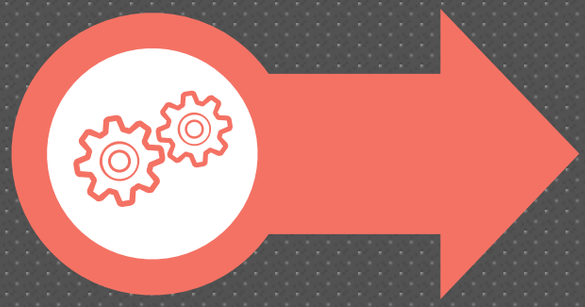


Branch Abstraction



Branch Abstraction

Library V1



Application

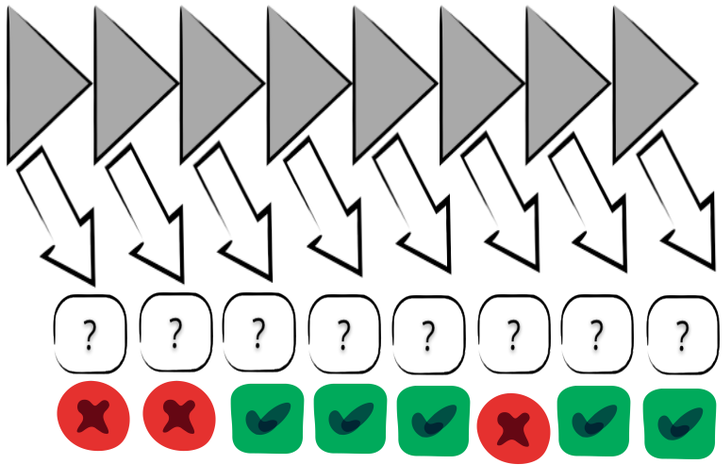
Library V2

Pipeline für progessives Testen

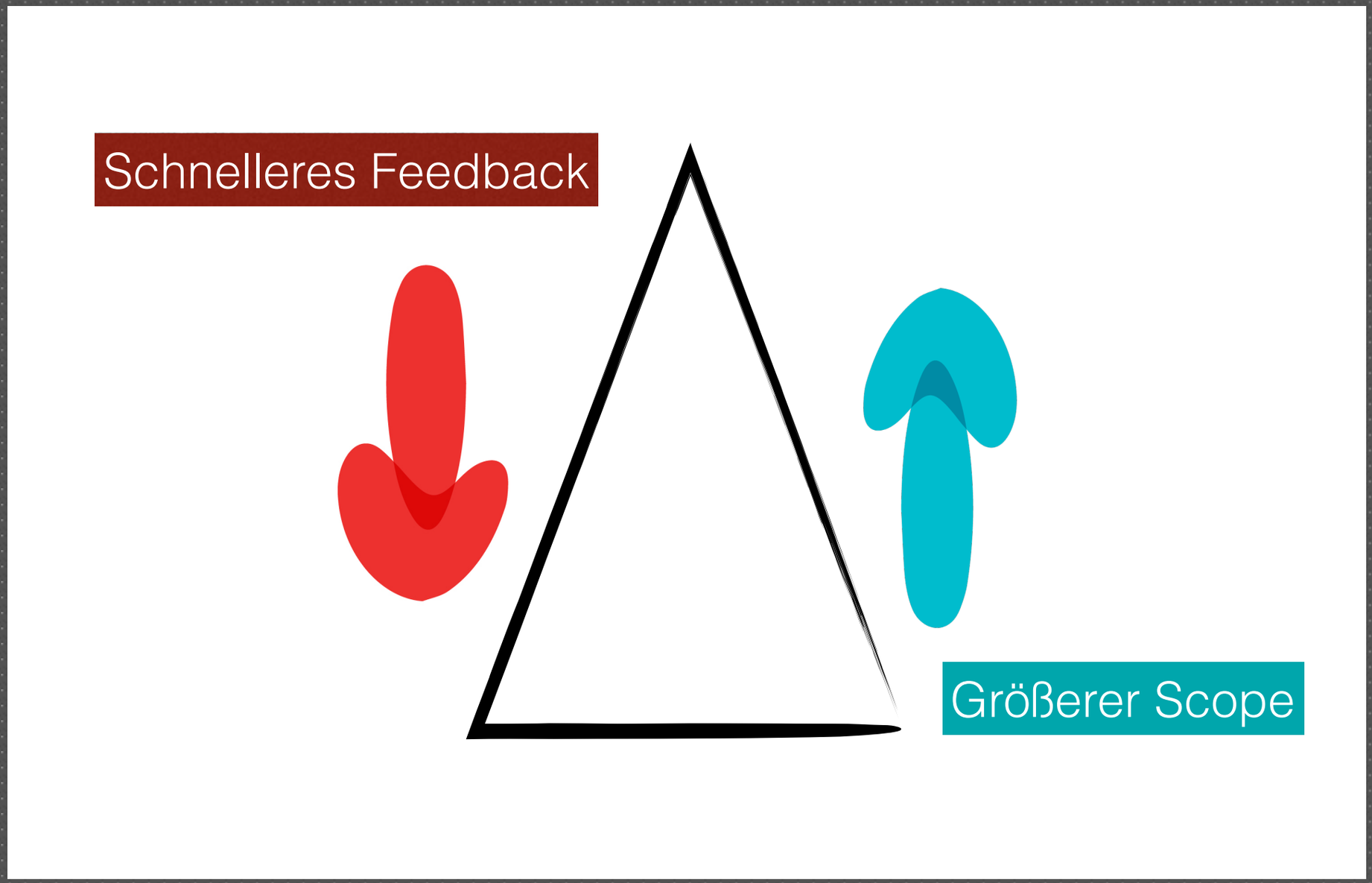
Schritt für Schritt zum Erfolg

Pipelines

Jeder Build ist ein Releasekandidat

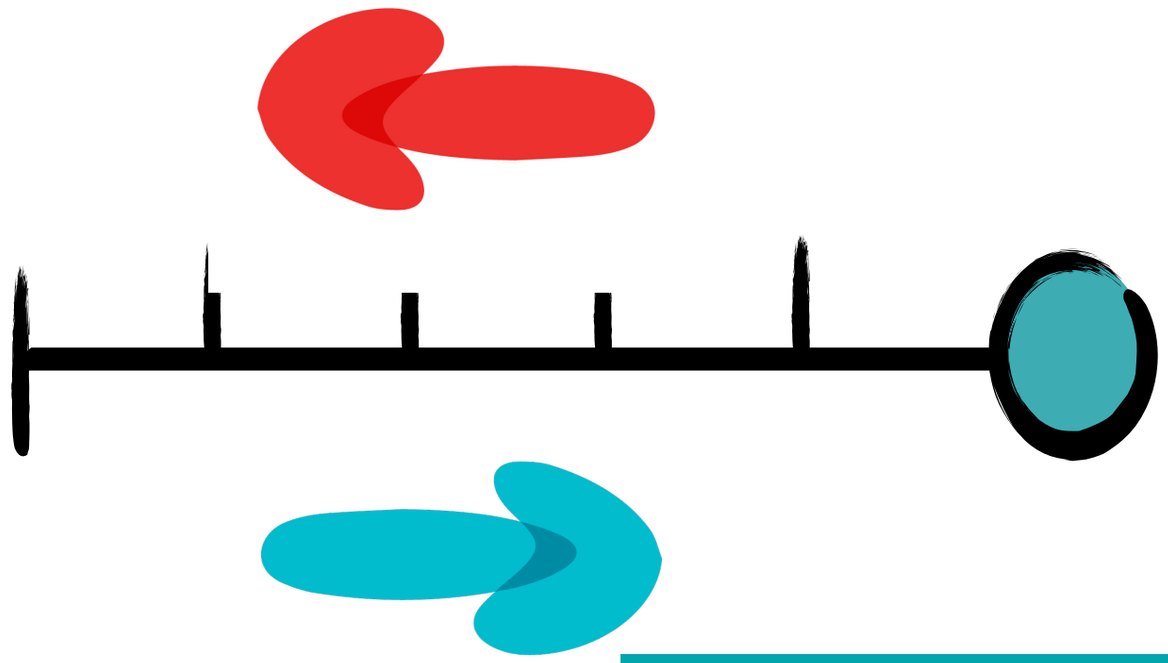


Test Pyramide



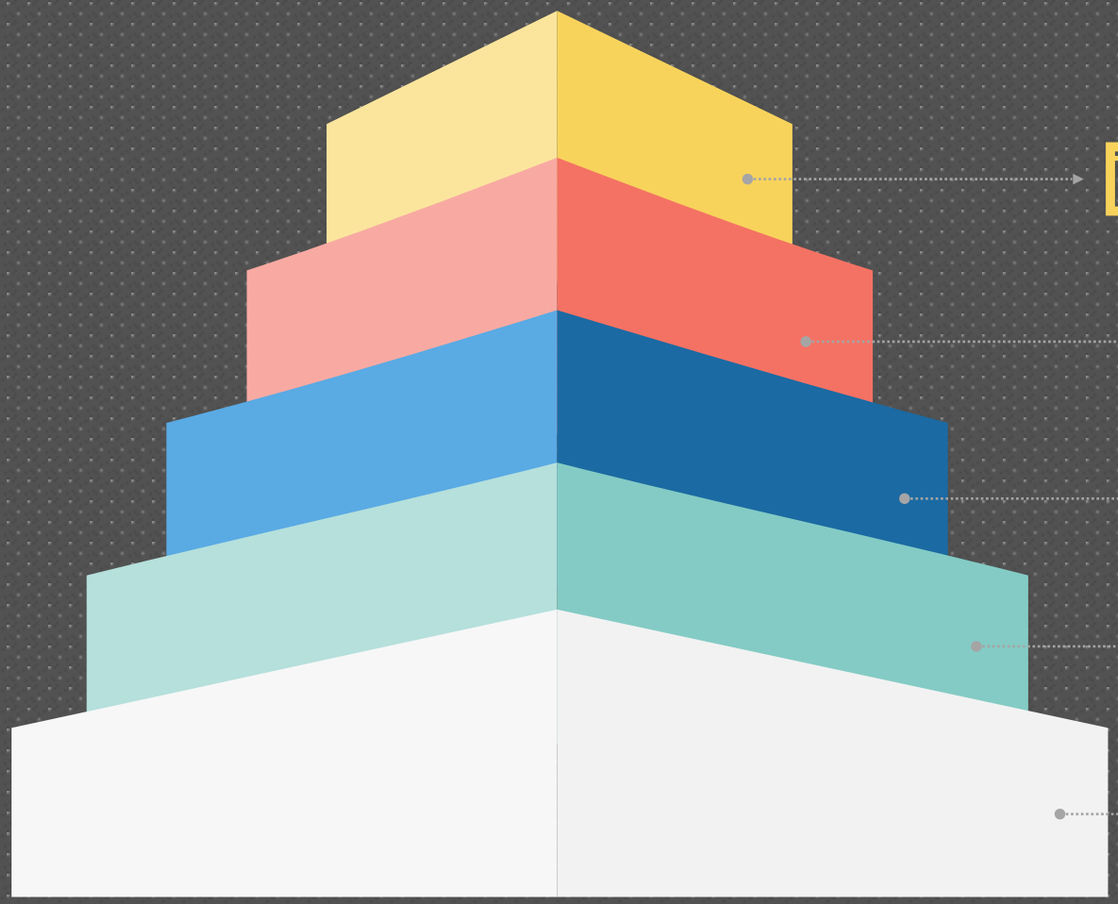
CD Pipeline

Schnelleres Feedback



Zunehmende Zufriedenheit

Test Pyramide



System Integration Tests
Unterschiedliche Systeme werden auf ihr Zusammenspiel getestet.

System Tests
Hier werden Applikationen im Zusammenspiel getestet

Application Tests
Testen die gesamte Applikation. Andere Services werden gemockt.

Integration Tests
Testen eine Gruppe von Komponenten

Unit Tests
Unit Tests für jede Komponente

CD Pipeline

Unit Tests

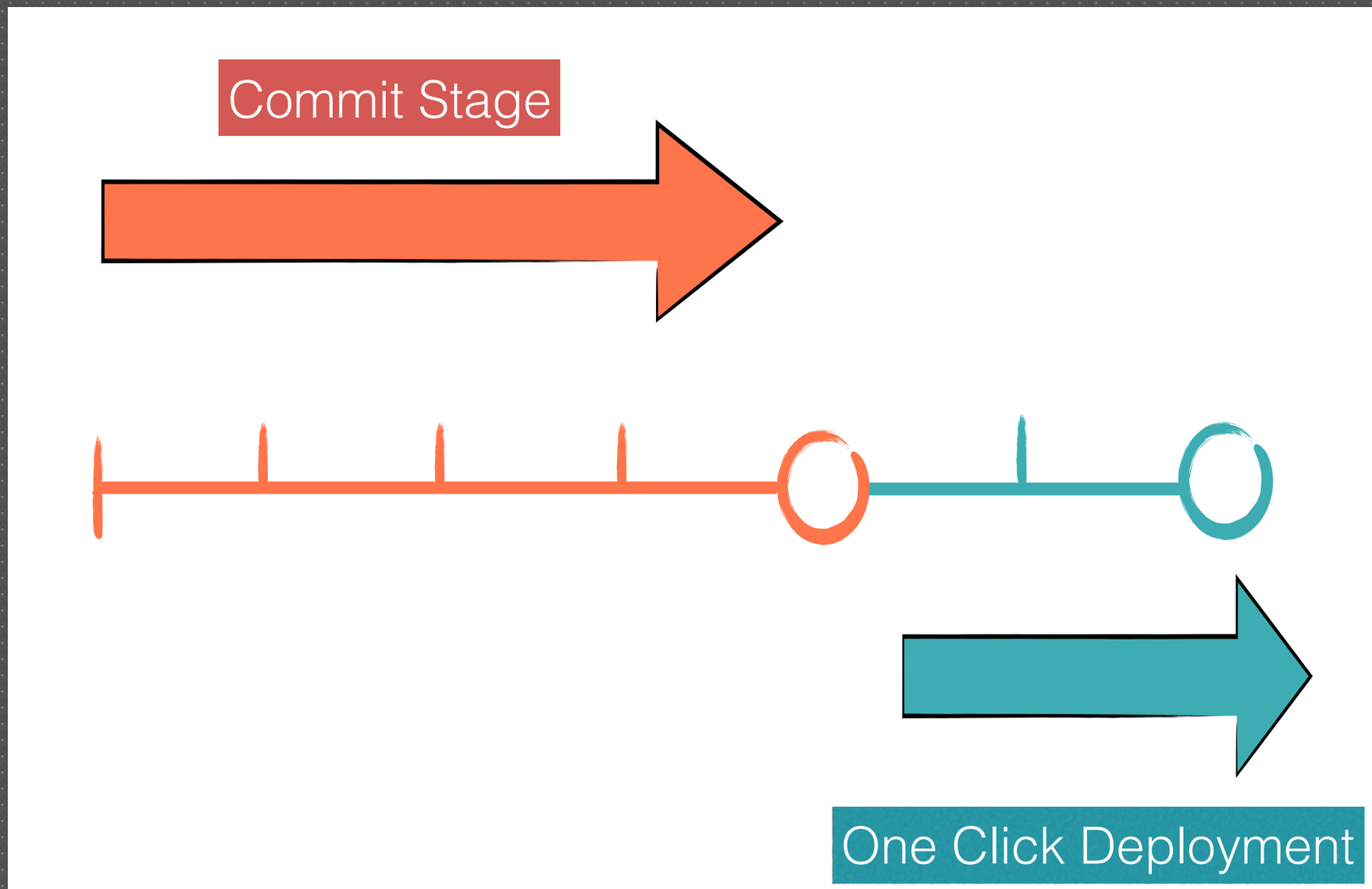
Integration Tests

Application Tests

System Tests

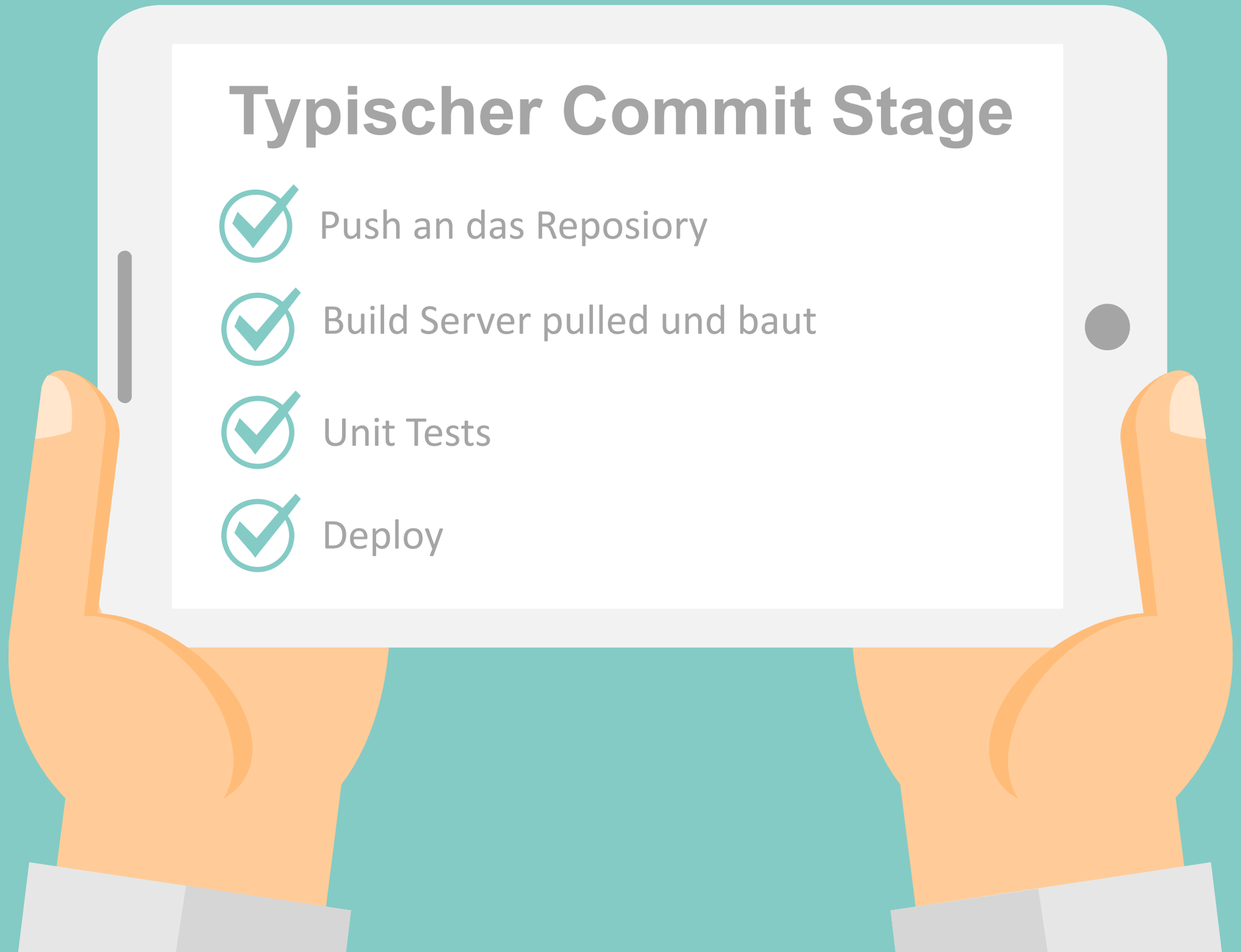
System Integration Tests

CD Pipeline



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

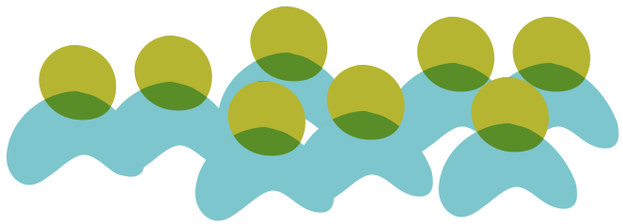
Sparen Geld und Nerven, denn sämtliche Abhängigkeiten und Features sind zuvor ausgiebig getestet worden

Beispiele für inkrementelle Deployment Muster

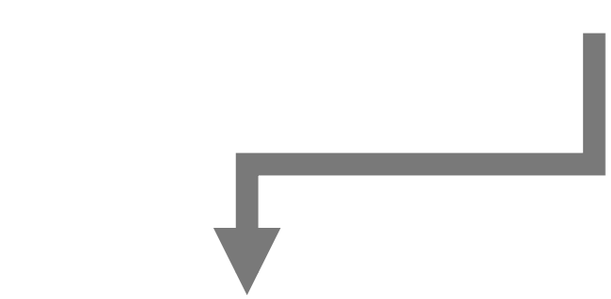
Sehen wir uns einige Beispiele hierfür an

BLUE-GREEN DEPLOYMENT MUSTER

BLUE-GREEN



Router



Blue Environment

Release 1

Green Environment

Nur eine Produktivumgebung ist live

BLUE-GREEN

Router

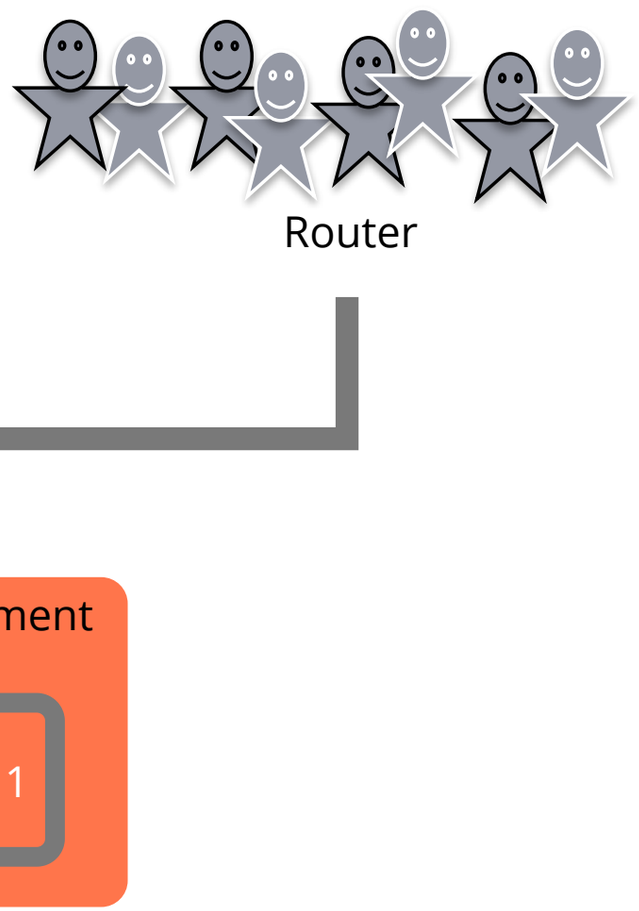
Umschalten im Router

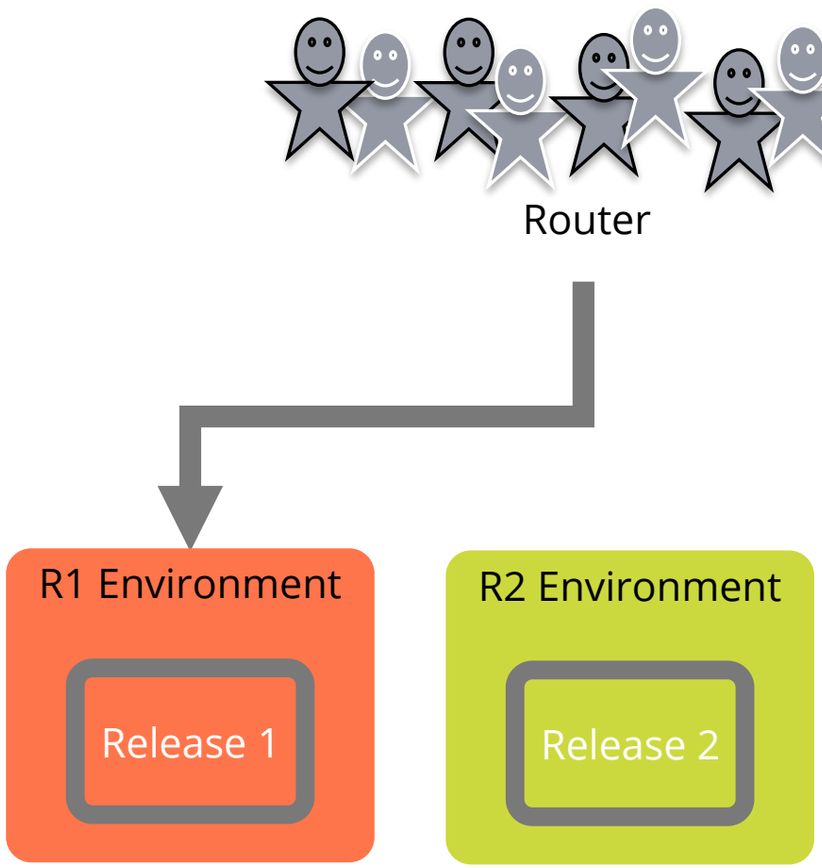
PHOENIX

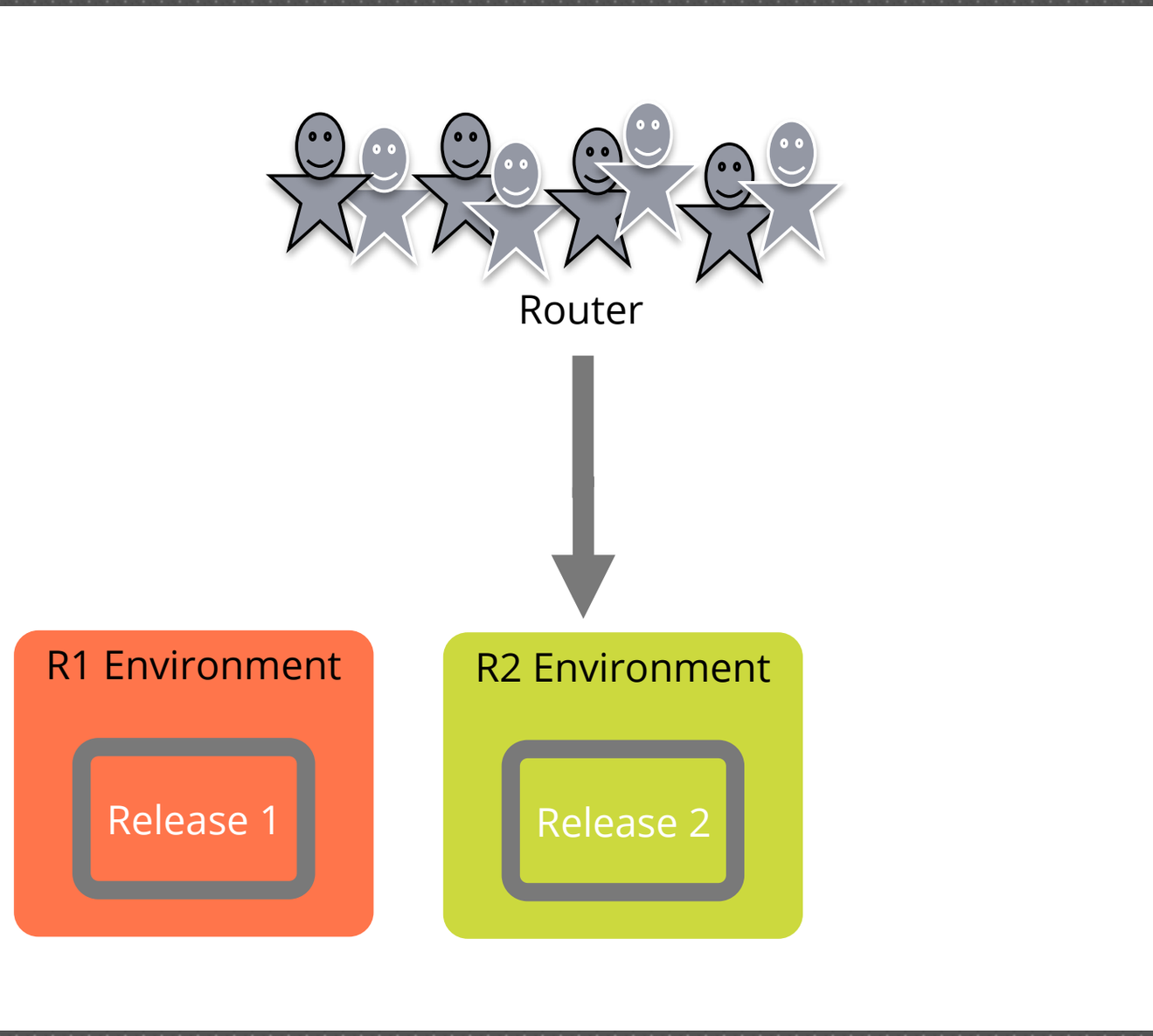
DEPLOYMENT MUSTER

Phoenix Deployment Muster

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


Router
R1 Environment
Release 1


Router
R1 Environment
Release 1
R2 Environment
Release 2


Router
R1 Environment
Release 1
R2 Environment
Release 2

Router

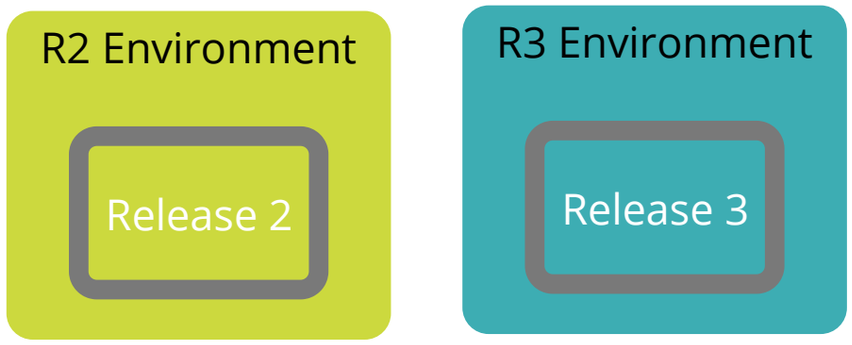
R1 Server „verbrannt“

R2 Environment

Release 2

Router

Prozess weiterführen mit R3

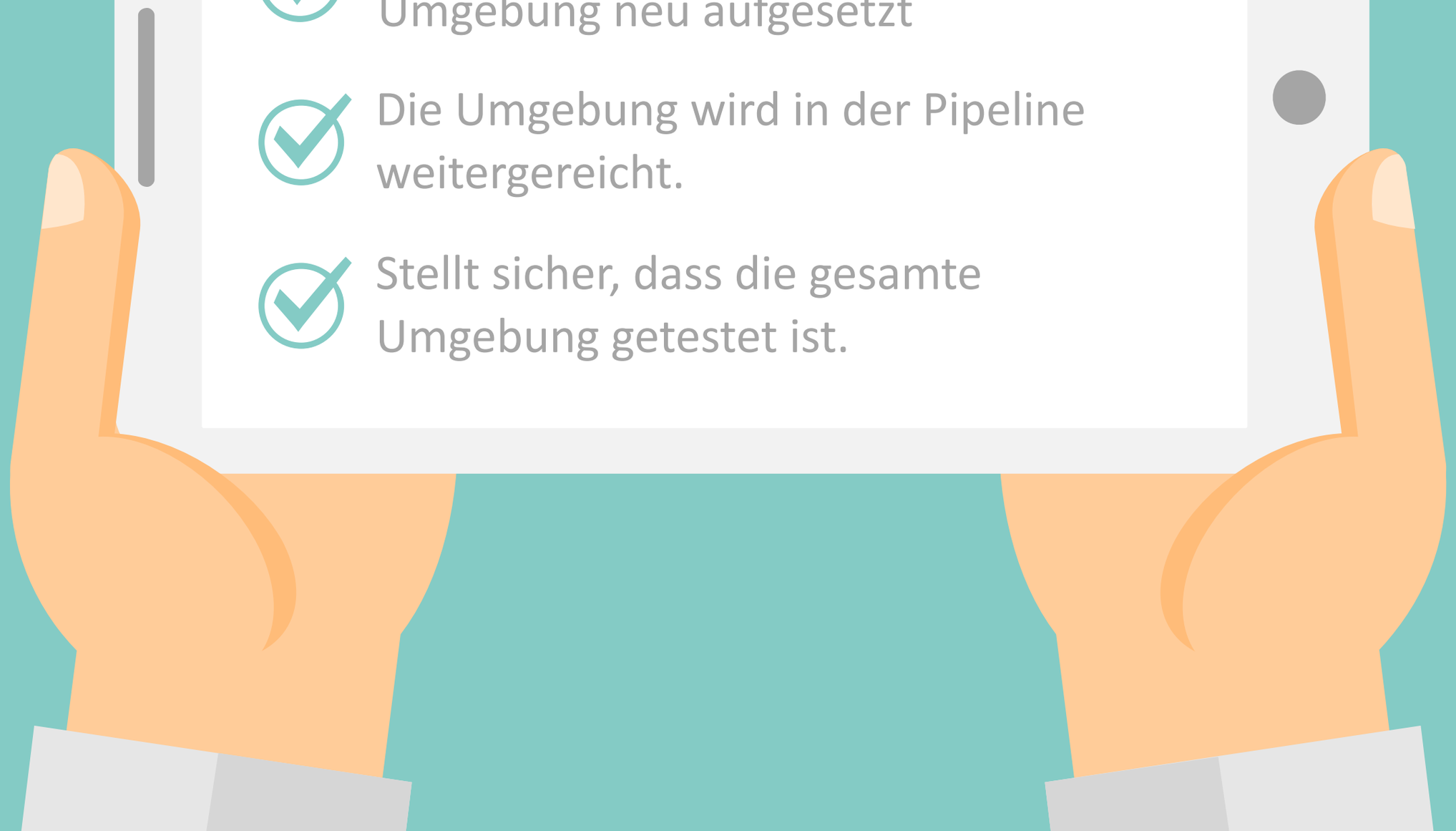


ENVIRONMENT PROMOTION

DEPLOYMENT MUSTER

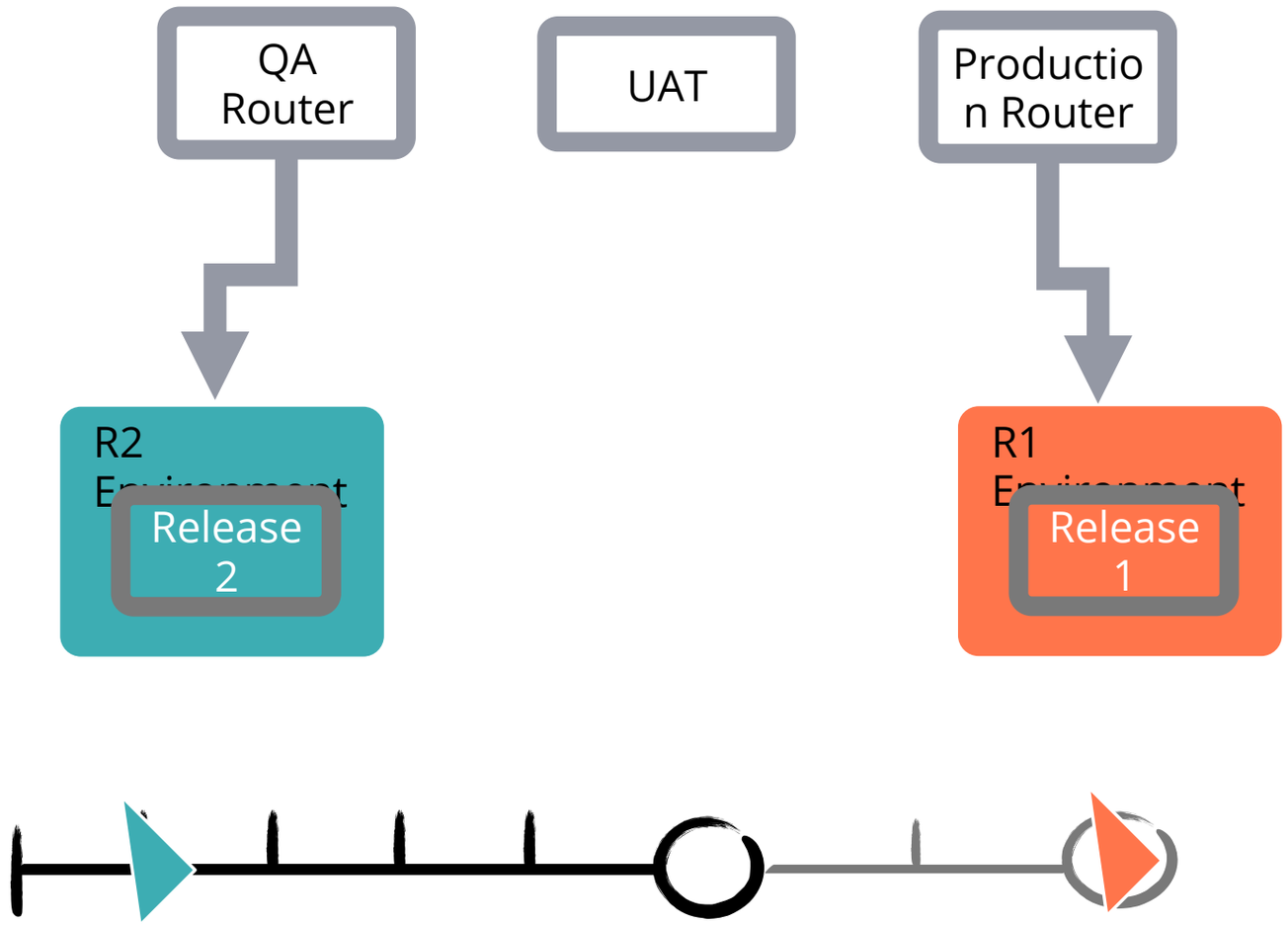
Environment Promotion

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



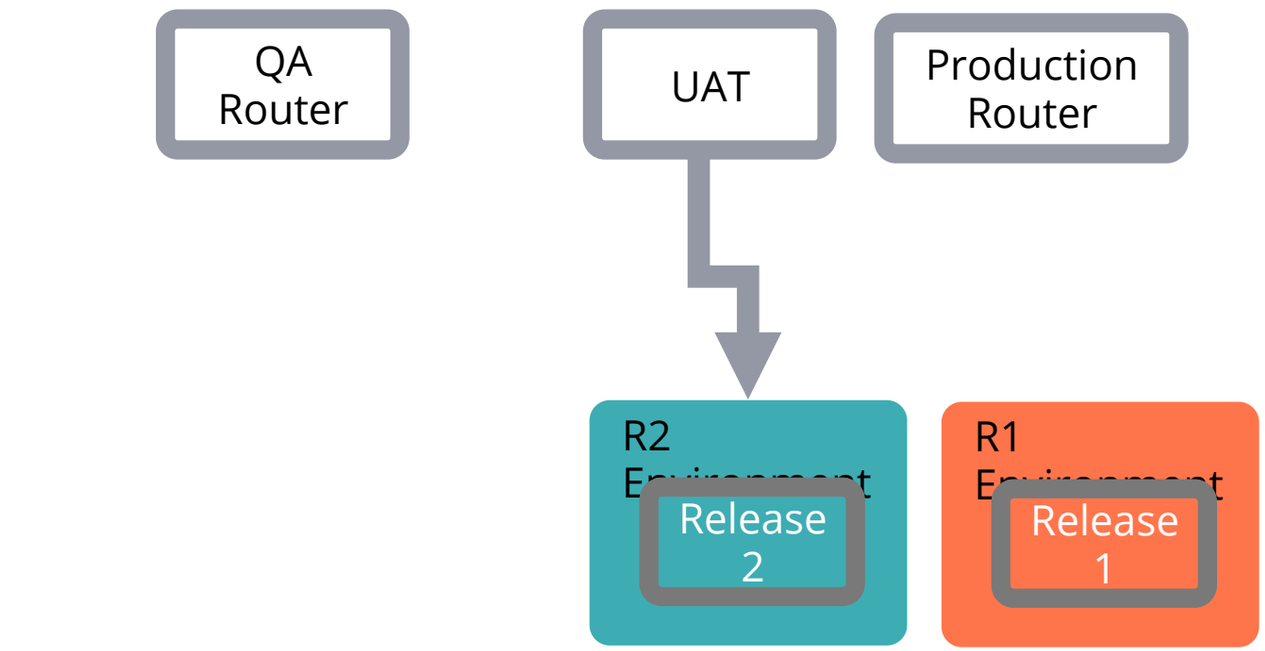
ENVIRONMENT PROMOTION

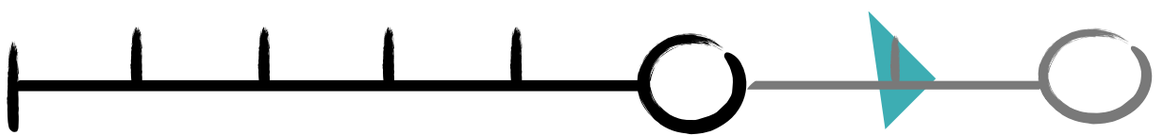
R2 wird im QA State getestet



ENVIRONMENT PROMOTION

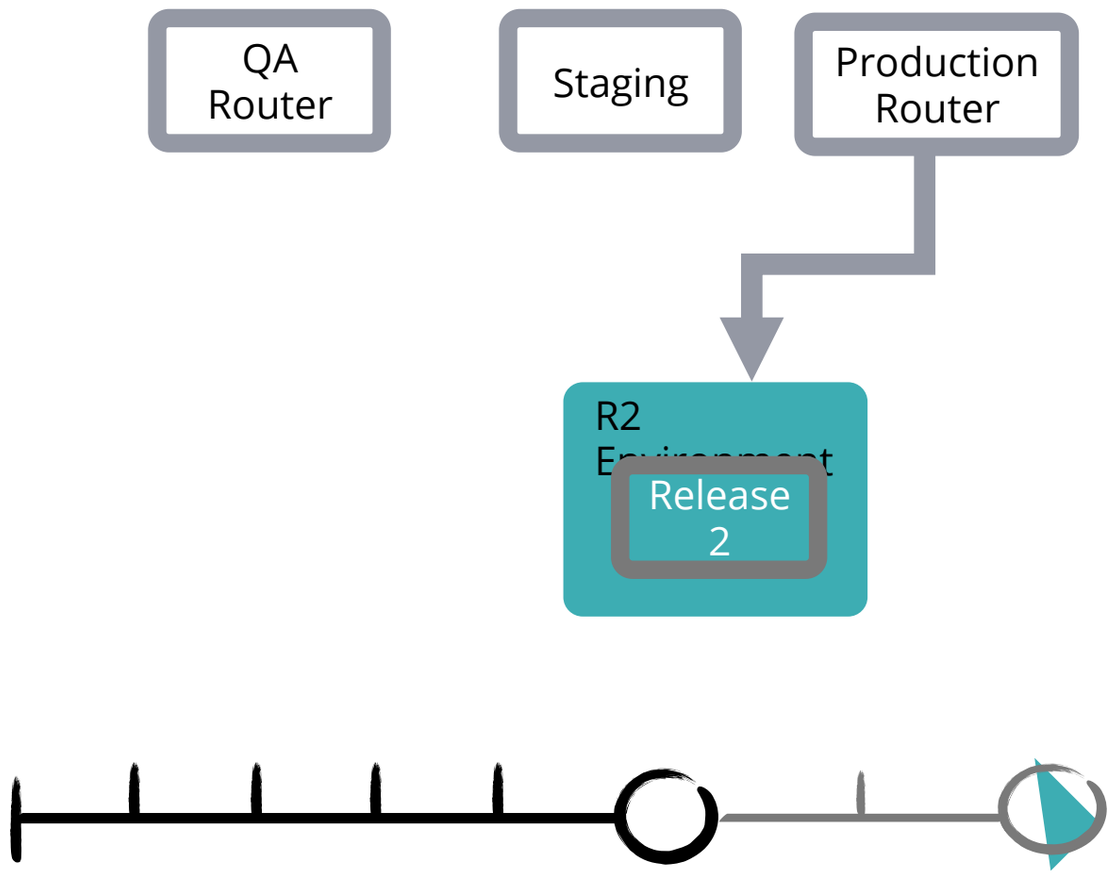
R2 wird im UAT getestet





ENVIRONMENT PROMOTION

R2 geht in Produktion und R1 wird zerstört



CANARY RELEASE

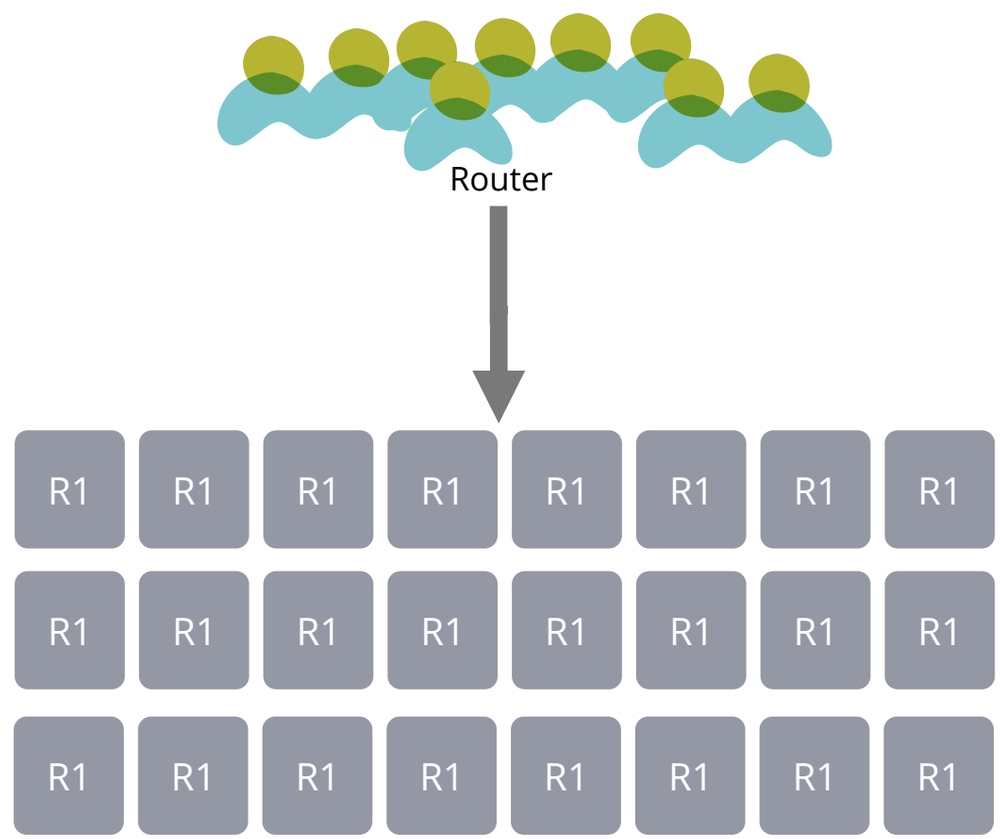
DEPLOYMENT MUSTER

Canary Release

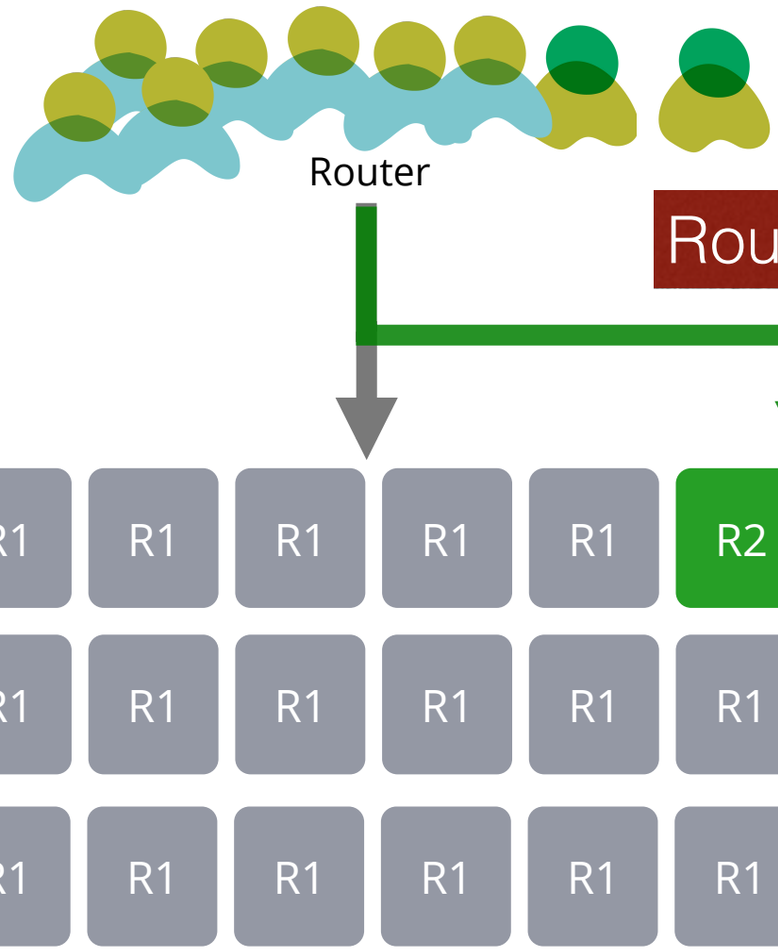
- Abwandlung von BLUE-GREEN für Clusterbetrieb
- Updates im Cluster werden schrittweise vorgenommen. Node by Node.
- Sichert Feedback einer kleinen Benutzergruppe

CANARY RELEASE

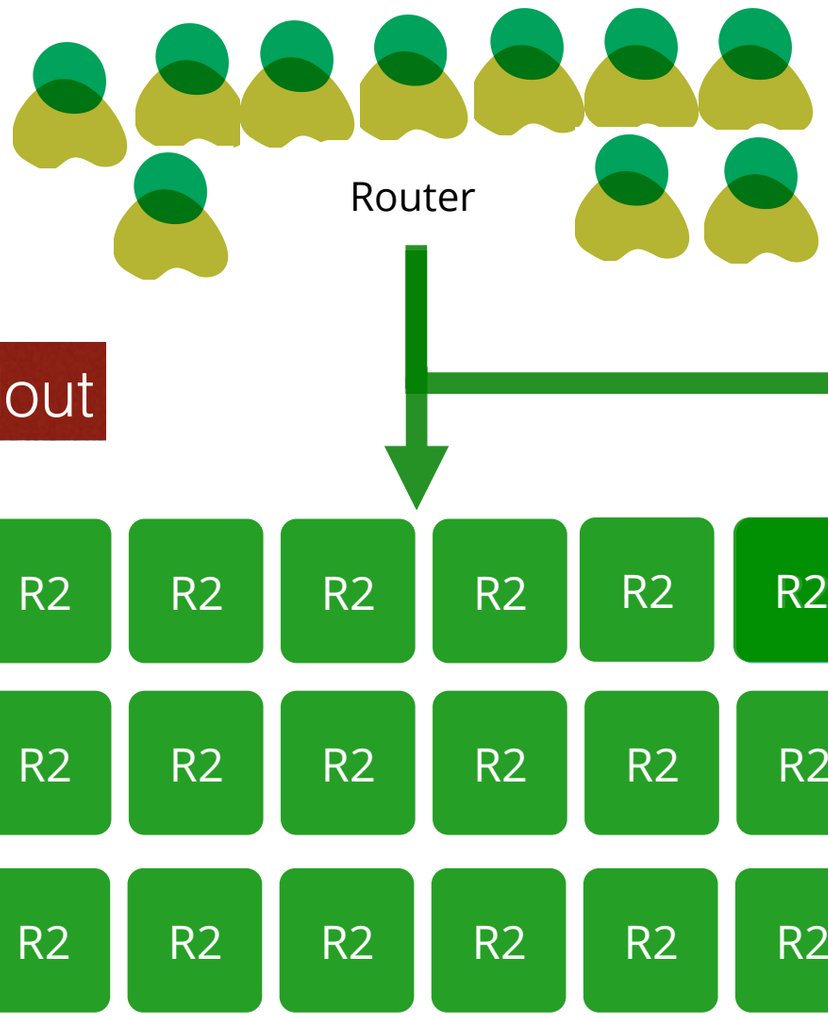
Servercluster



CANARY RELEASE



CANARY RELEASE



Tools für Continuous Deployment

Warum kompliziert, wenn es auch einfach geht

Jenkins

Atlassian Bamboo

Travis CI