

API-Tests

Contract-Tests

Consumer-driven Contract-Tests

Alles dasselbe?

Wer
braucht das?

Wozu
ist
das
gut?

WTF?!?

Thomas Much, Techniker Krankenkasse
JavaLand 2026

Europapark Rust, 10. März 2026

 @thmuch

JavaLand

MARCH 10 - 12, 2026

in EUROPA PARK in Rust



Architektur

Fachlichkeit
(Domäne)

Grundlagen.

Konzepte.

„Warum“ verstehen.

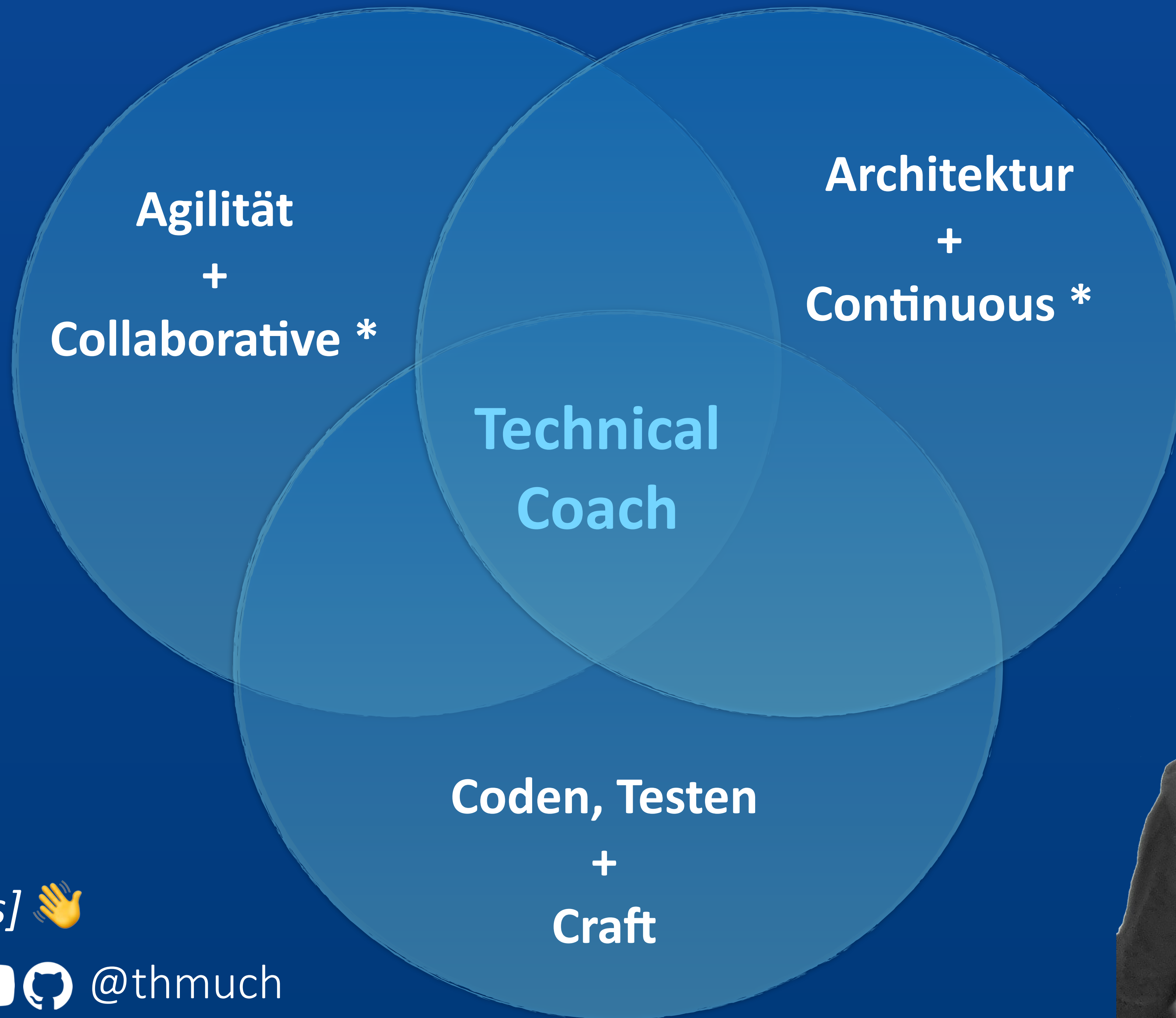
Was wird damit gelöst?

Herausforderungen.

Ausblick.

Contract-Tests

INKL.
KLEINER
LIVE-
DEMOS



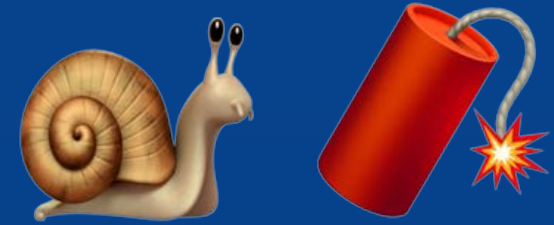
www.tk.de/IT



['to:mas] 🖐️

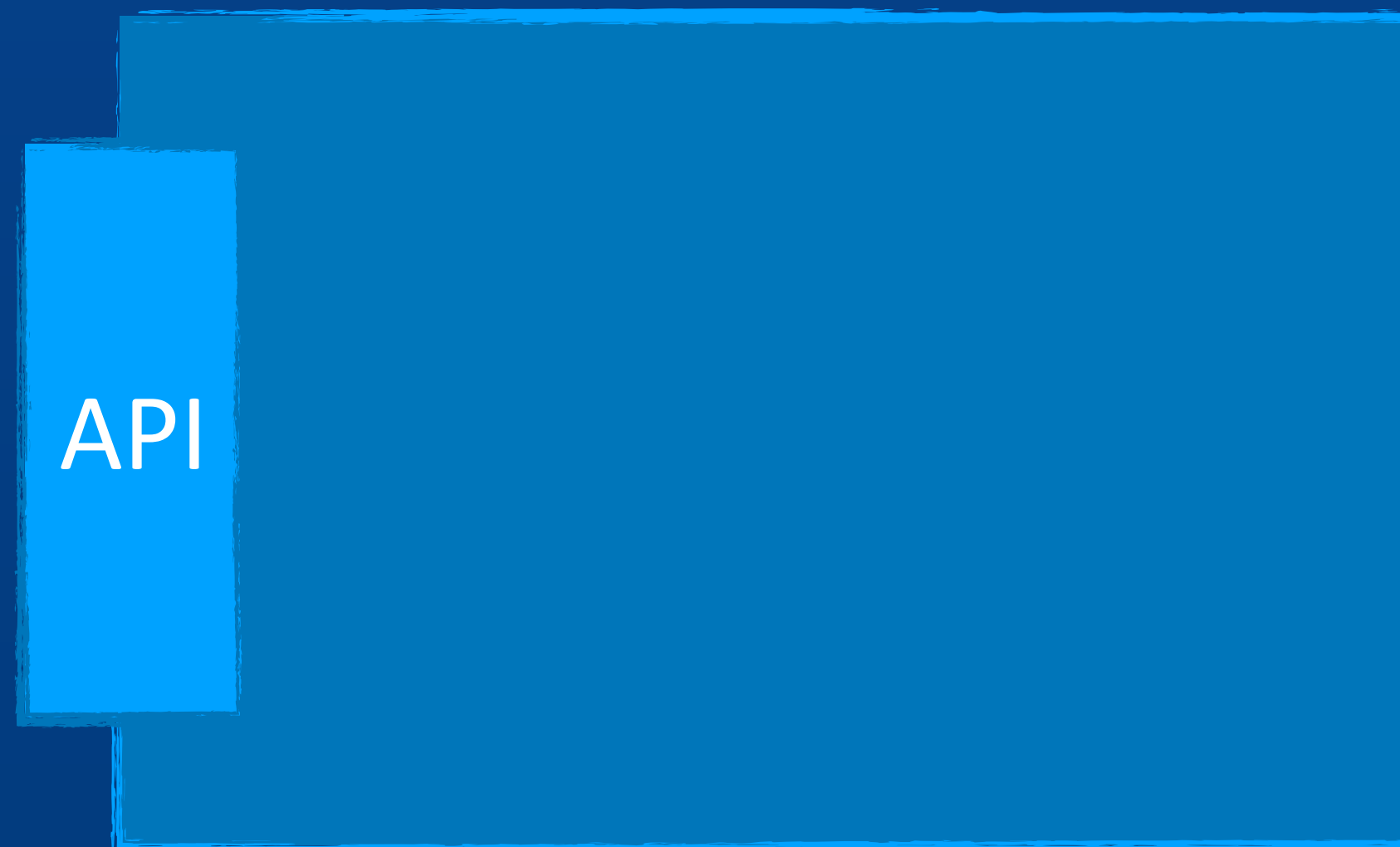
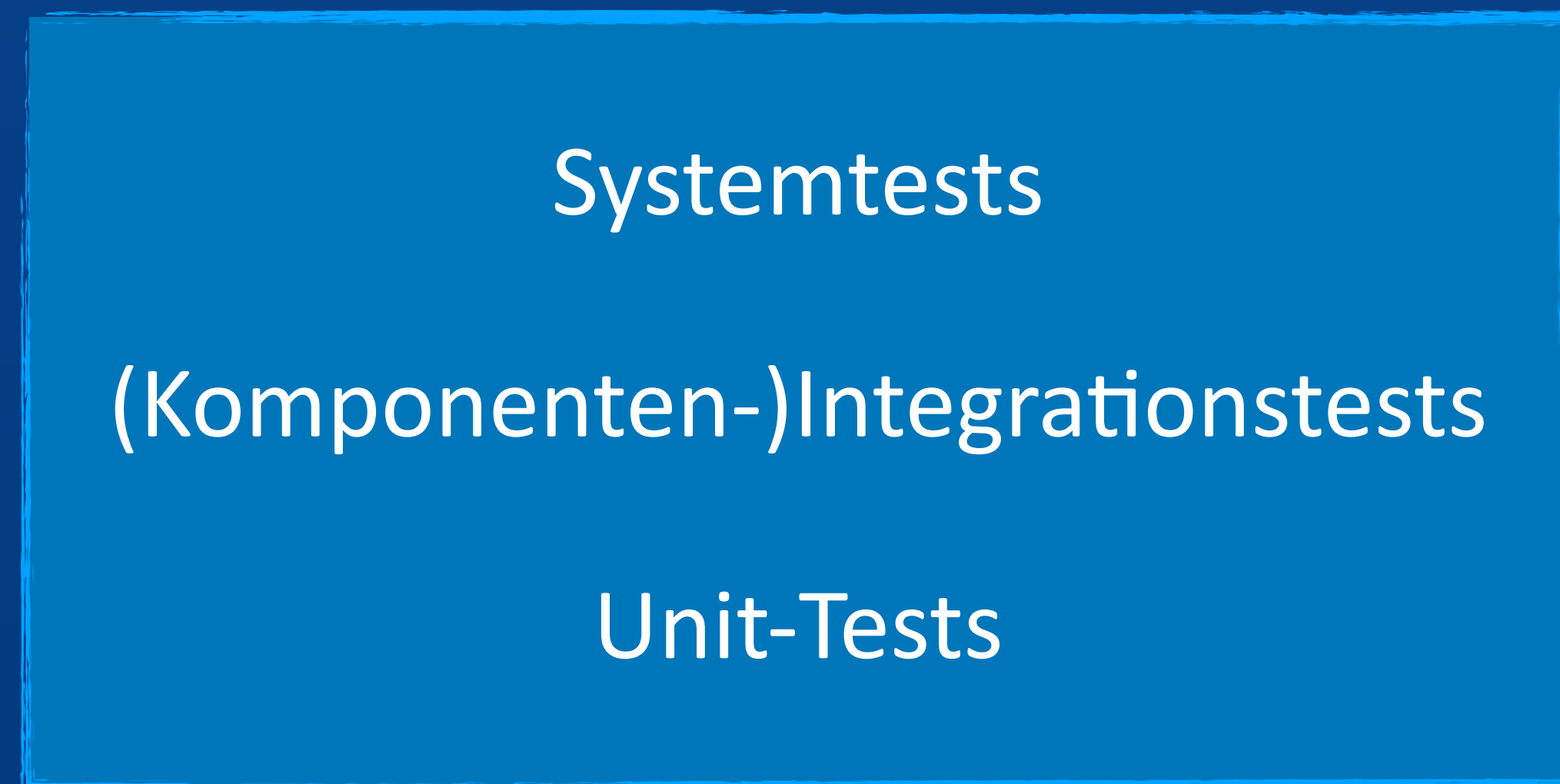
📧 🦋 📺 🐙 @thmuch

Testen, testen, testen ...



Was sind die
„Enden“?

◀.....Systemintegrationstests bzw. „End-to-end“ (E2E).....▶



Immer alles „echt“ mittesten?

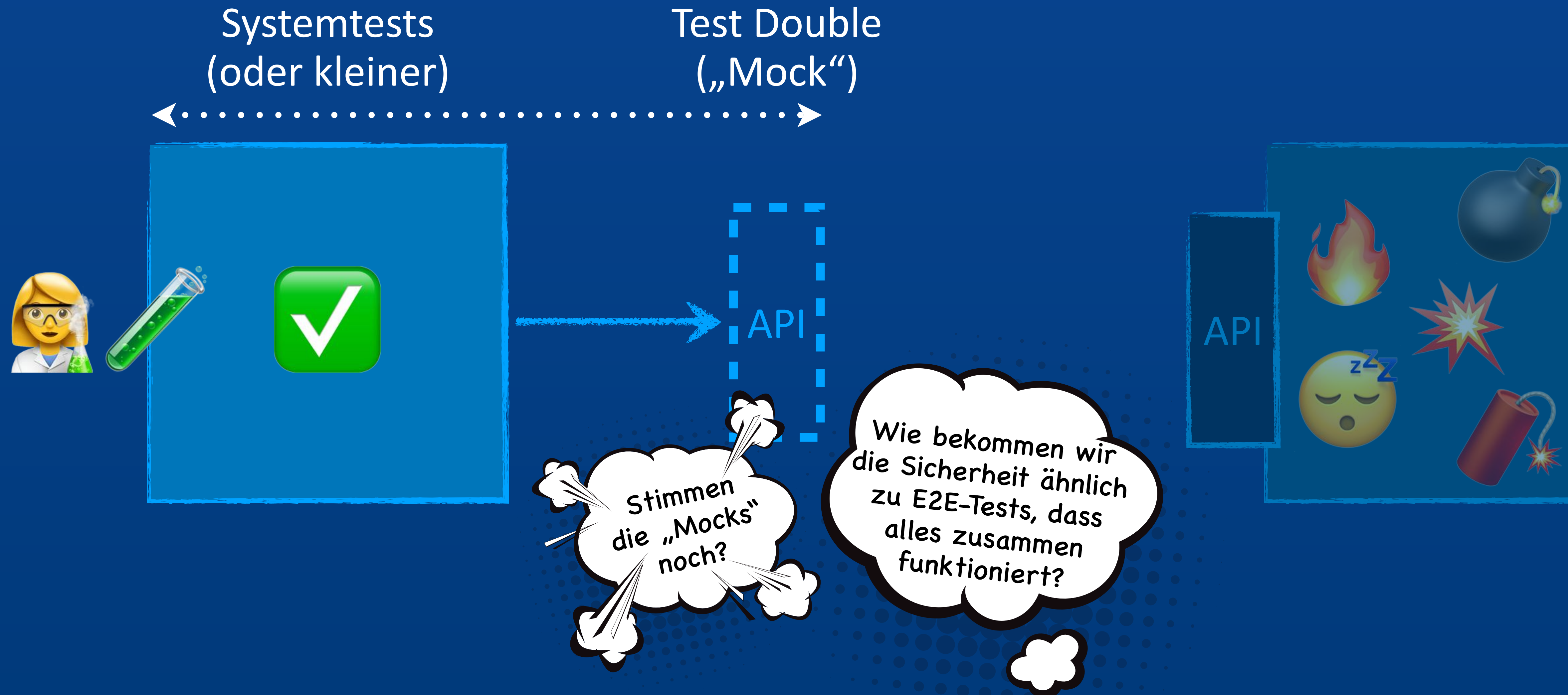
◀..... Systemintegrationstests bzw. „End-to-end“ (E2E)▶



„Ansteckung“ bzw. „Infektion“ („Contagion“)

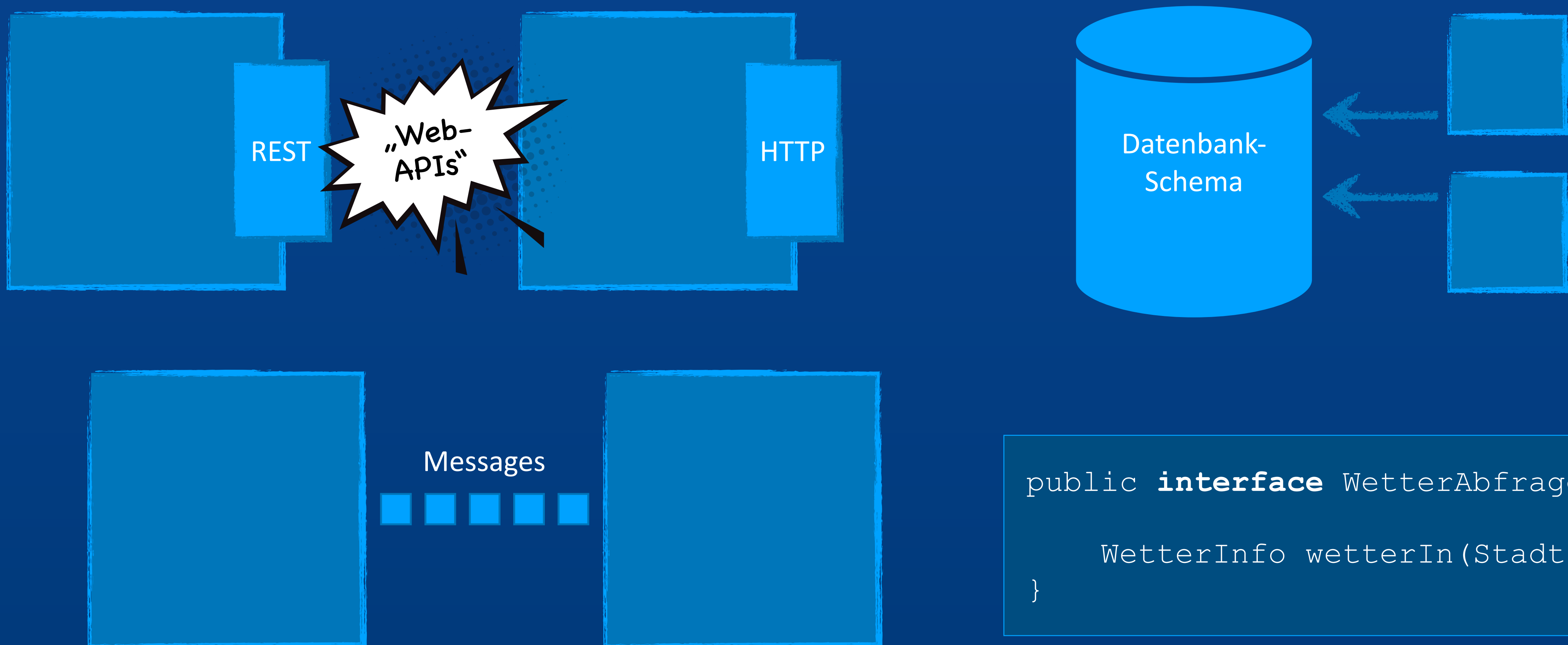


Isolierter testen heißt stabiler testen



Was sind „APIs“?

“Application Programming Interfaces” – Programmierschnittstellen



```
public interface WetterAbfrage {  
    WetterInfo wetterIn(Stadt stadt);  
}
```

Was sind „Contracts“?

Aussagen über **strukturelle Validität** (inkl. Typen) und **Integrationskompatibilität**.

yaml

```
WetterInfo:
  type: object
  required:
    - stadt
    - temperatur
  properties:
    stadt:
      type: string
      description: lokale
    temperatur:
      type: integer
      description: Temper
      minimum: -90
      maximum: 60
```

XML

```
<definitions xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/"
  targetNamespace="http://www.w3.org/2001/XMLSchema"
  >
  <types>
    <xs:schema targetNamespace="http://www.w3.org/2001/XMLSchema"
      <xs:element name="WetterInfo" type="WetterInfo" />
    <xs:complexType base="xs:string" />
    <xs:sequence base="xs:string" />
    <xs:element name="stadt" type="xs:string" />
```

JSON

```
{
  "consumer": { "name": "wetter-app" },
  "provider": { "name": "wetter-service" },
  "interactions": [
    {
      "description": „aktuelles Wetter der Stadt“,
      "request": {
        "method": "GET",
        "path": "/wetter",
        "query": "stadt=Hamburg"
      },
      "response": {
        "status": 200,
        "body": {
          "stadt": "Hamburg",
          "temperatur": 21
        },
        "matchingRules": {
          "body": {
            "$.temperatur": {
              "matchers": [
                {
                  "match": „integer“
                },
                ...
              ]
            }
          }
        }
      }
    }
  ]
}
```

```
public WetterInfo wetterIn(Stadt stadt) {
    // Vorbedingung(en)
    // Invariante(n)
    // Nachbedingung(en)
}
```

„Design by Contract“ (DbC) – Bertrand Meyer, 1986+

```
public interface WetterAbfrage {

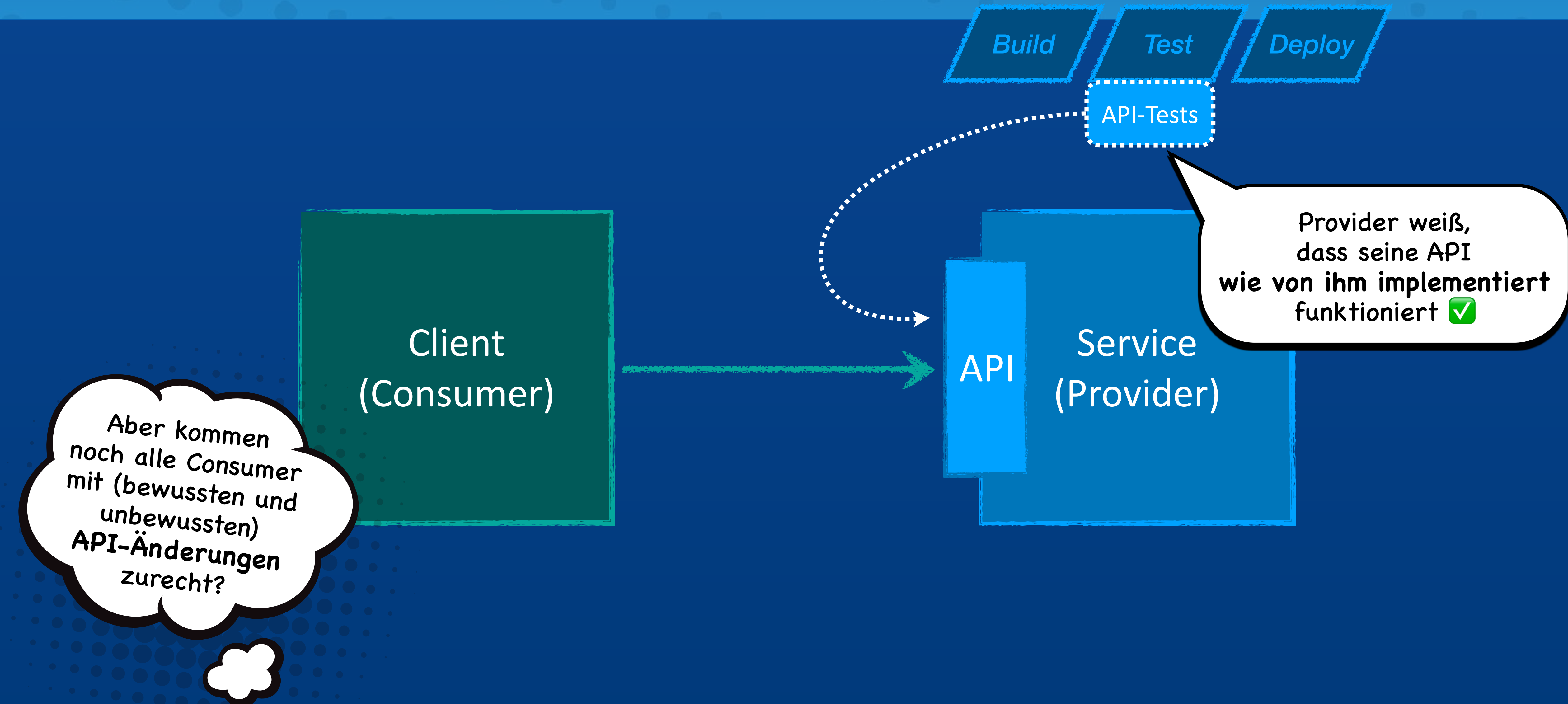
    /**
     * Aktuelle Temperatur in °C.
     * @return niemals null
     * @throws NPE wenn stadt==null
     */
    WetterInfo wetterIn(Stadt stadt);
}
```

Und was ist an Contracts
„consumer-driven“?

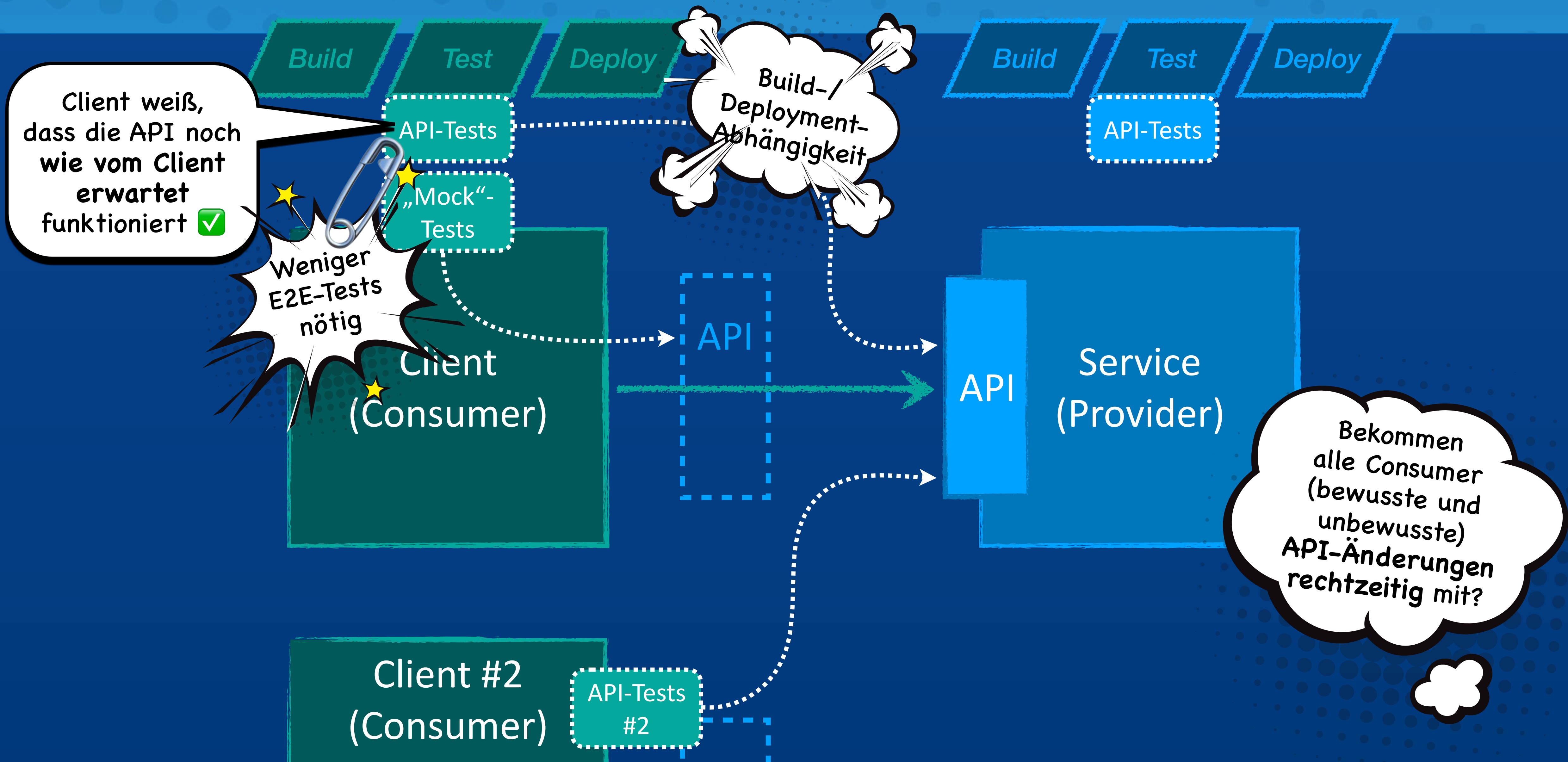
Super Frage!

Fangen wir vorne an.

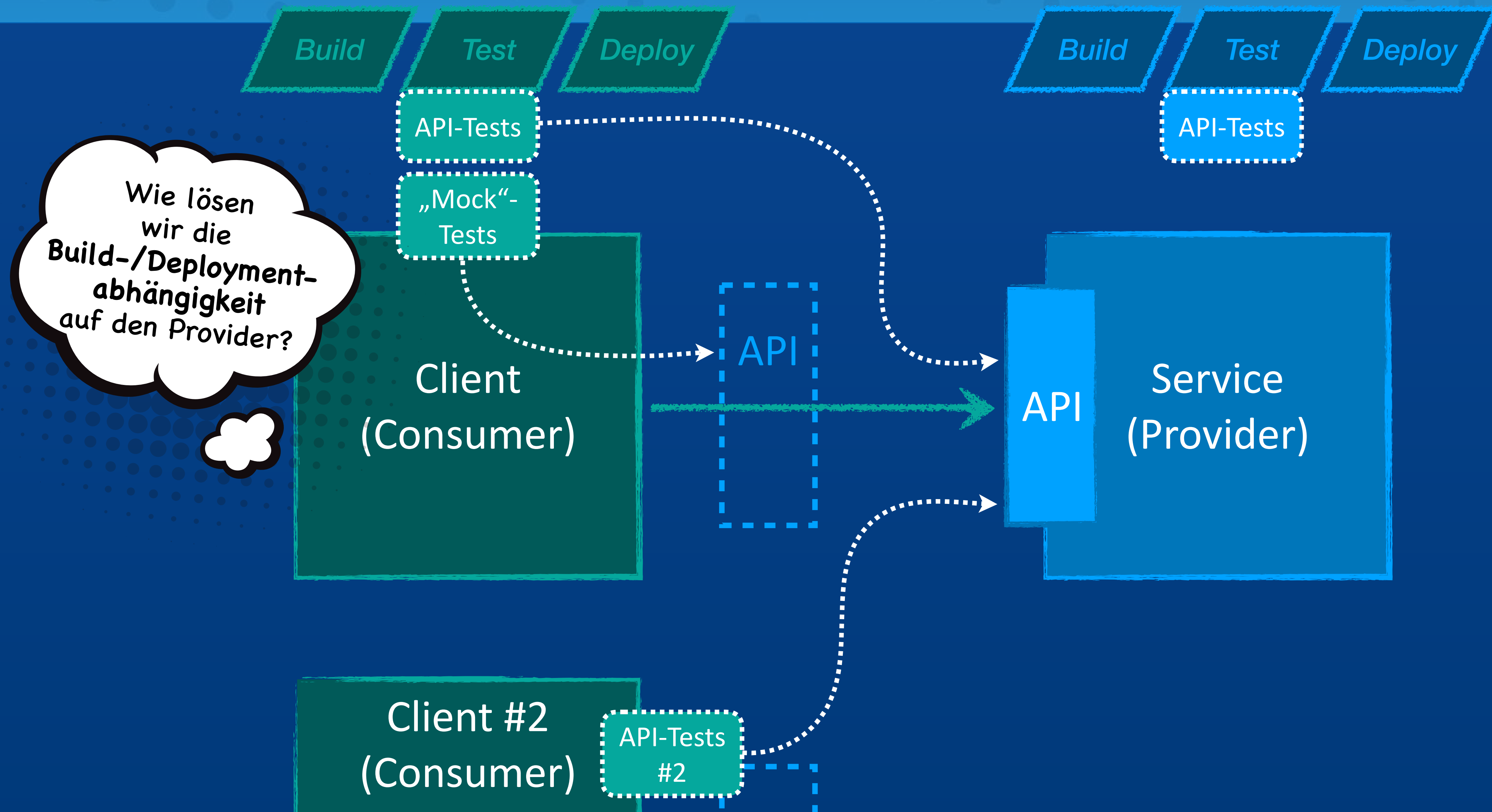
API-Tests – Provider-Seite



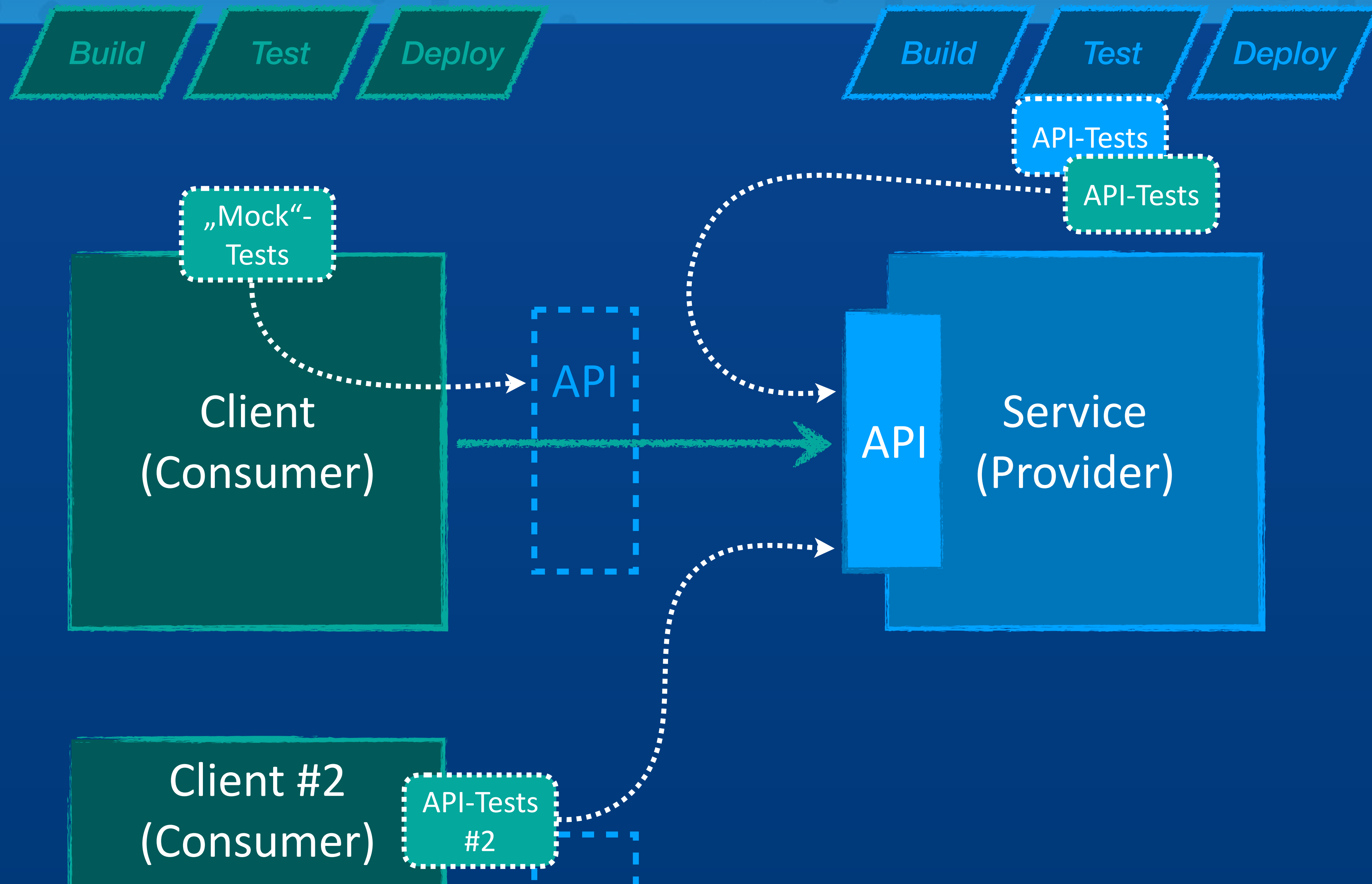
API-Tests – Consumer-Seite



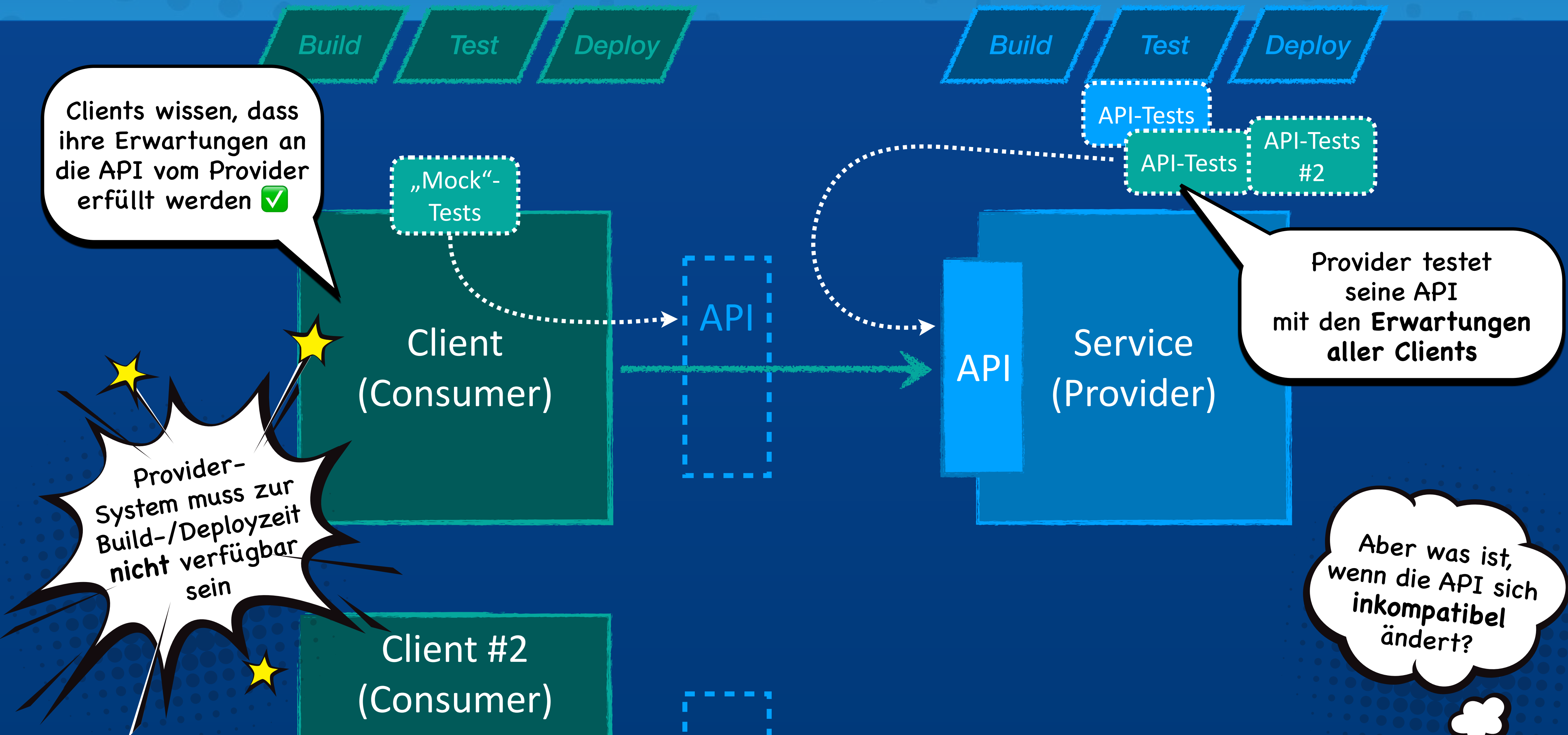
API-Tests – Consumer-Seite



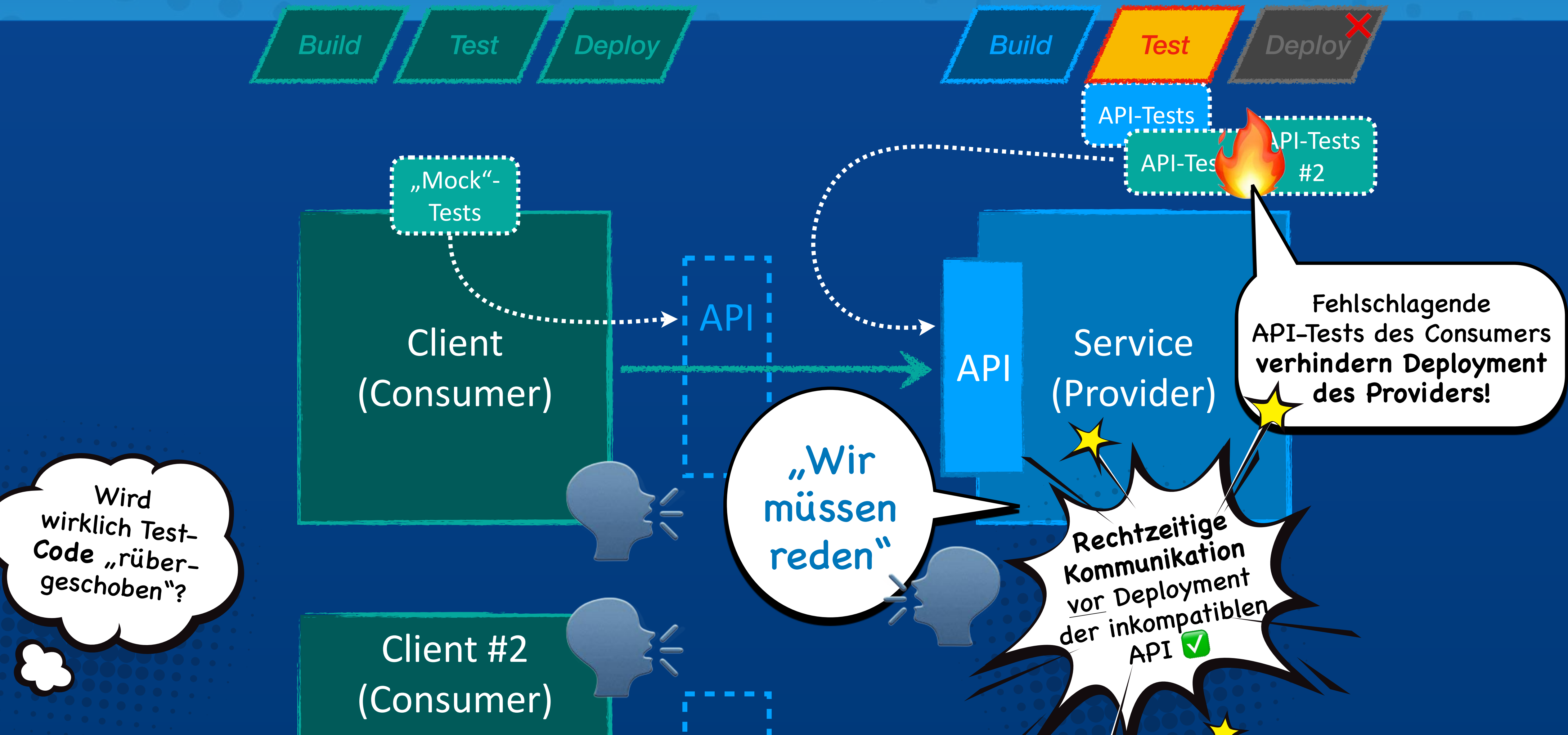
API-Tests – Consumer-driven



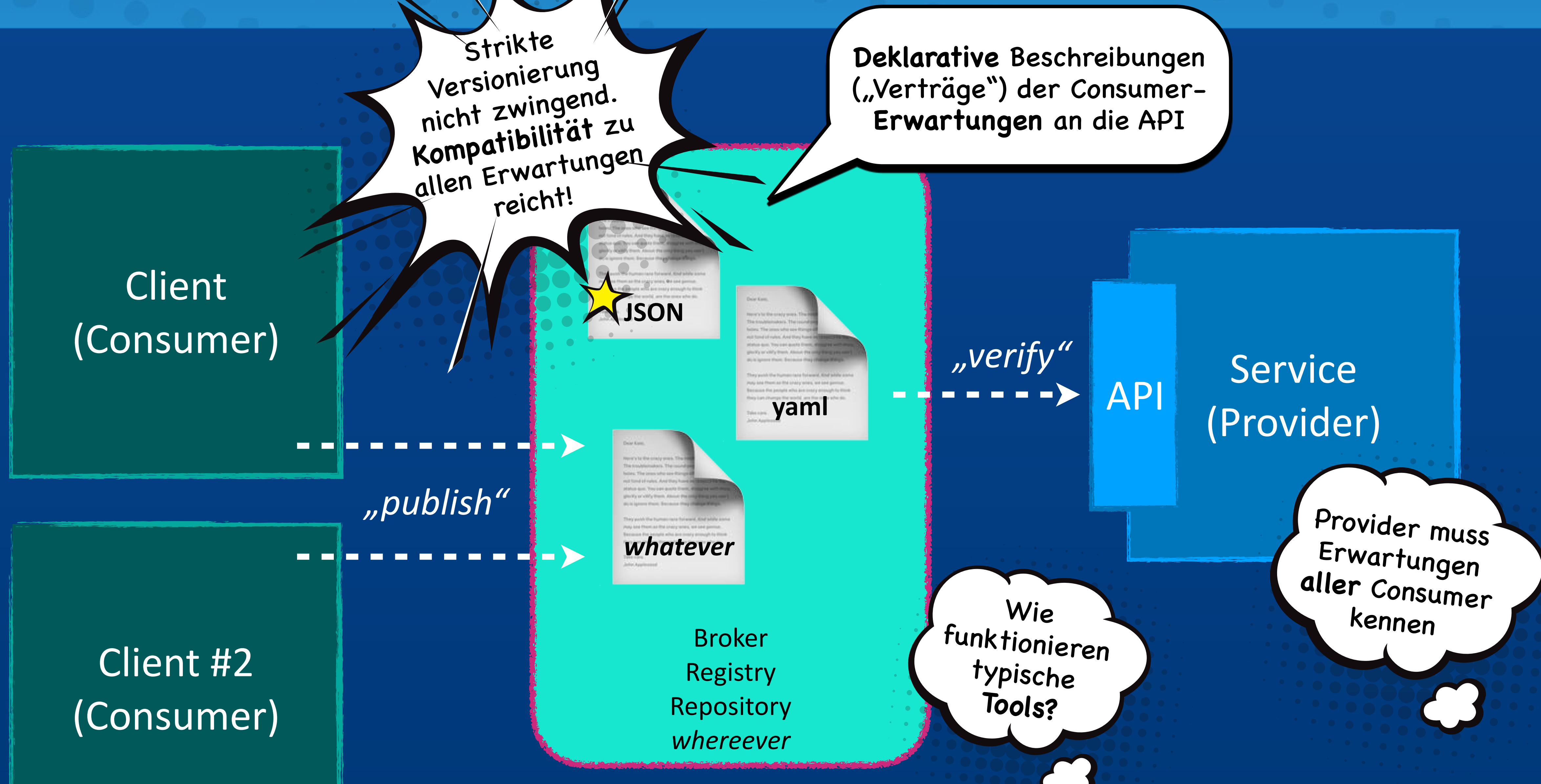
API-Tests – Consumer-driven



API-Tests – Consumer-driven

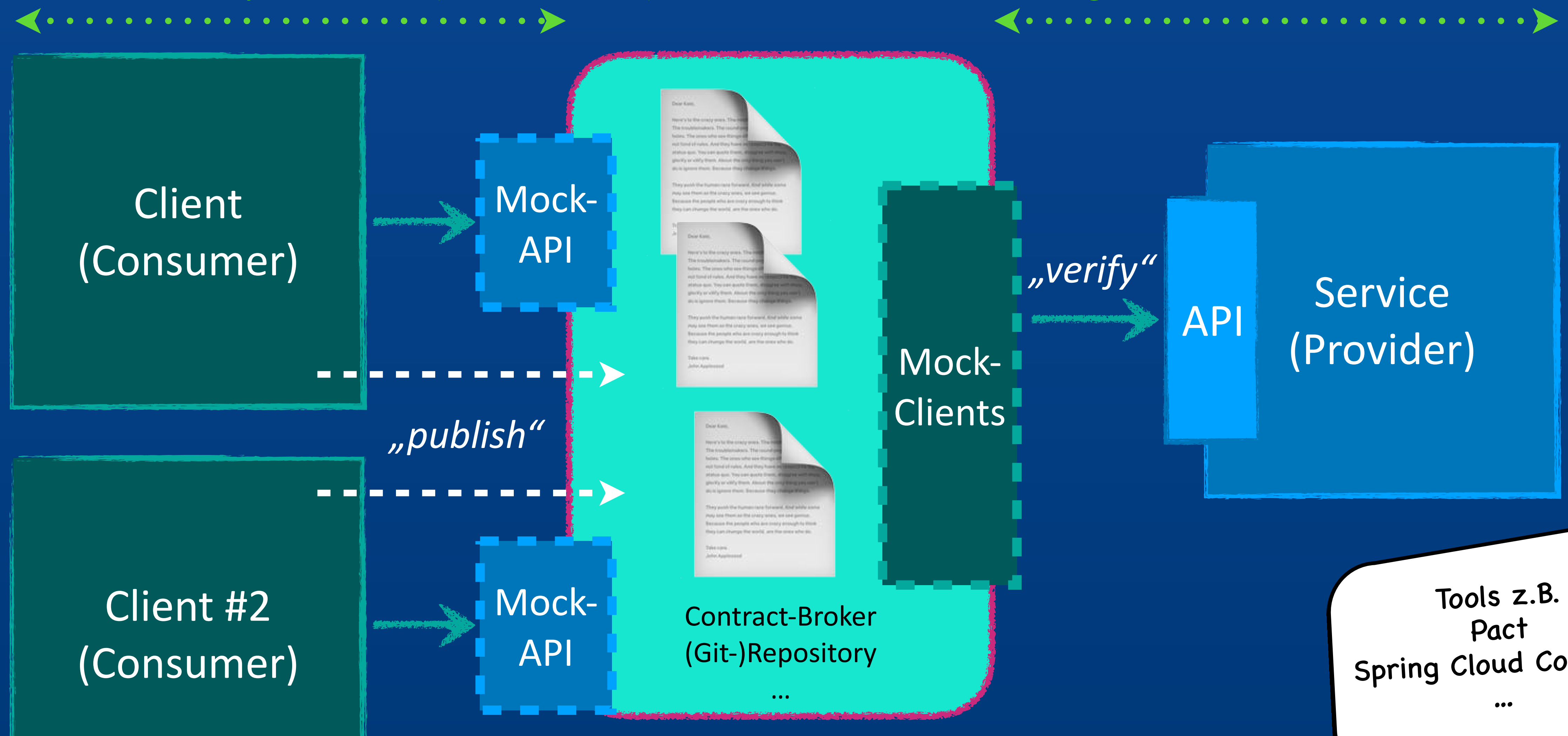


Consumer-driven Contracts



Consumer-driven Contract-Tests

Systemtests (oder kleiner) mit ähnlicher Absicherung wie E2E-Tests

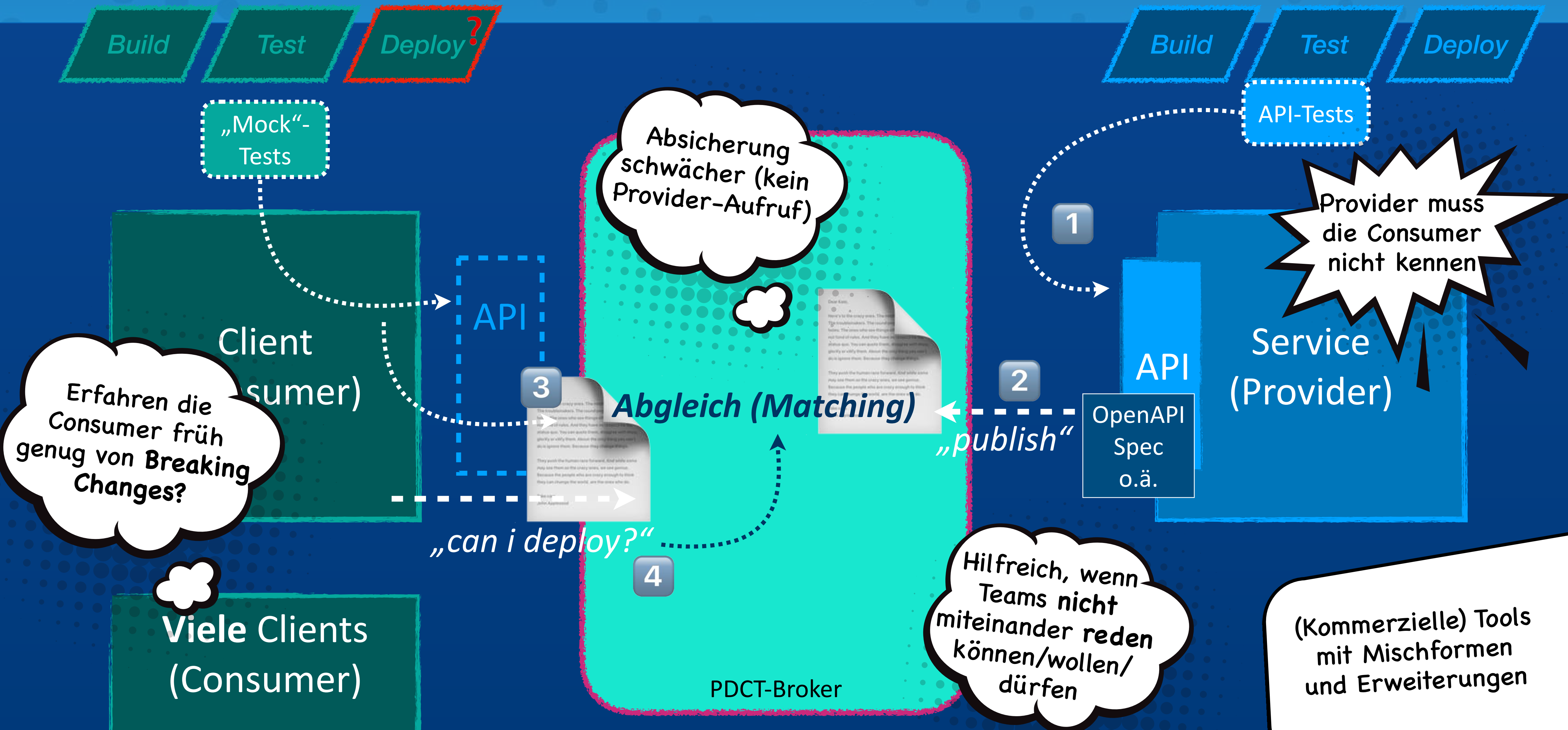




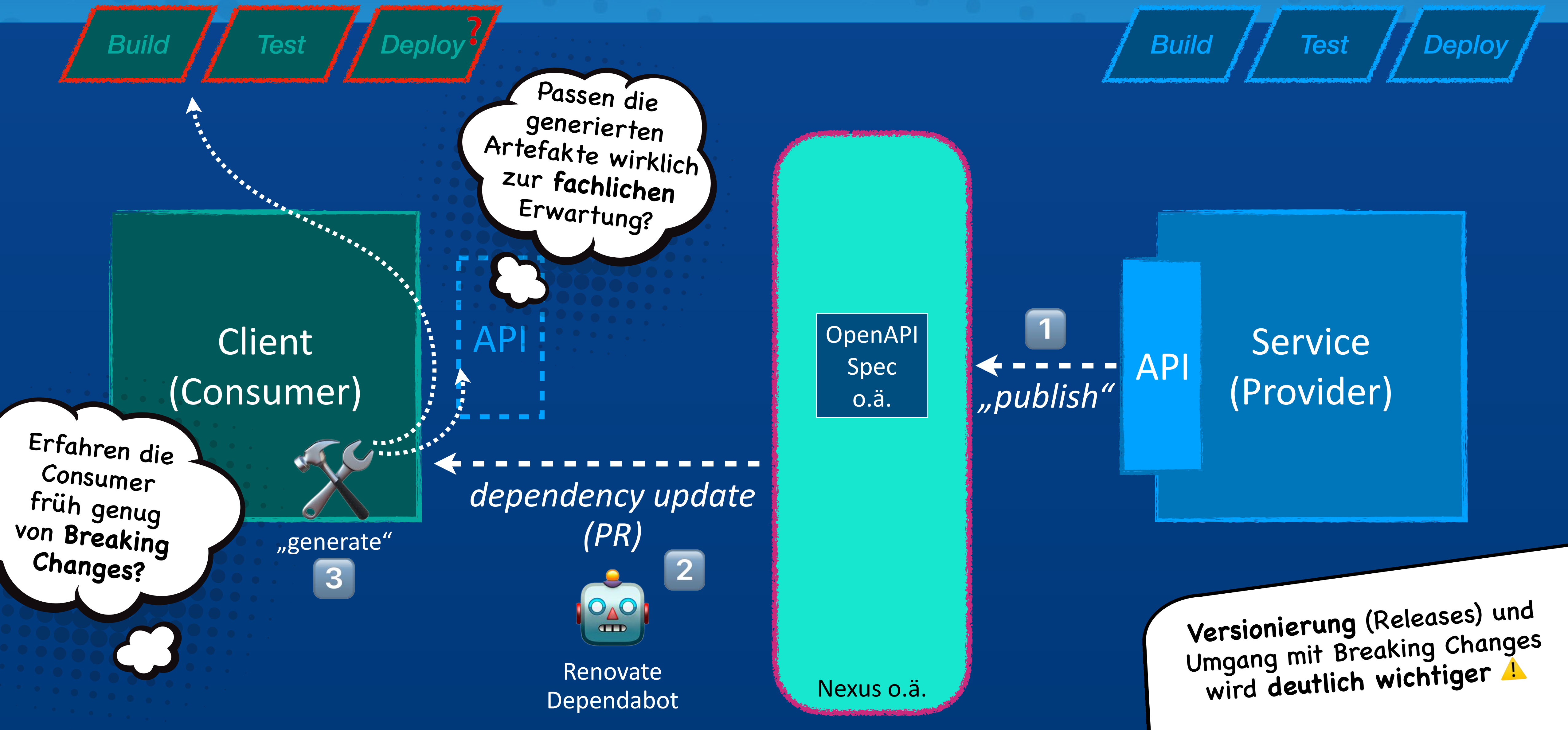
Wenn es
„consumer-driven“
gibt ...

... gibt es dann auch
„provider-driven“?

Provider-driven Contract-Tests

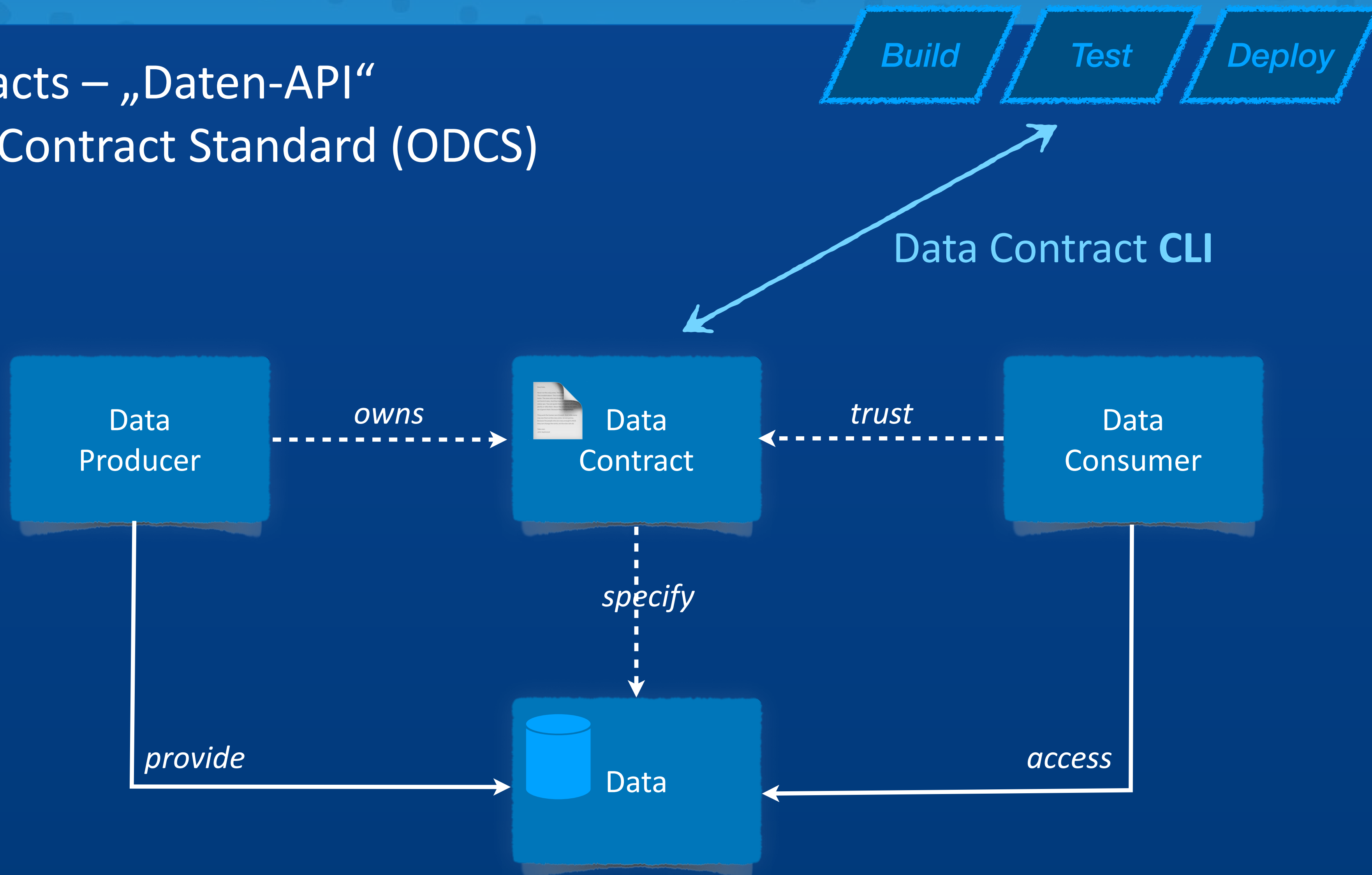


Provider-driven Contracts ohne Broker



Data Contract Tests

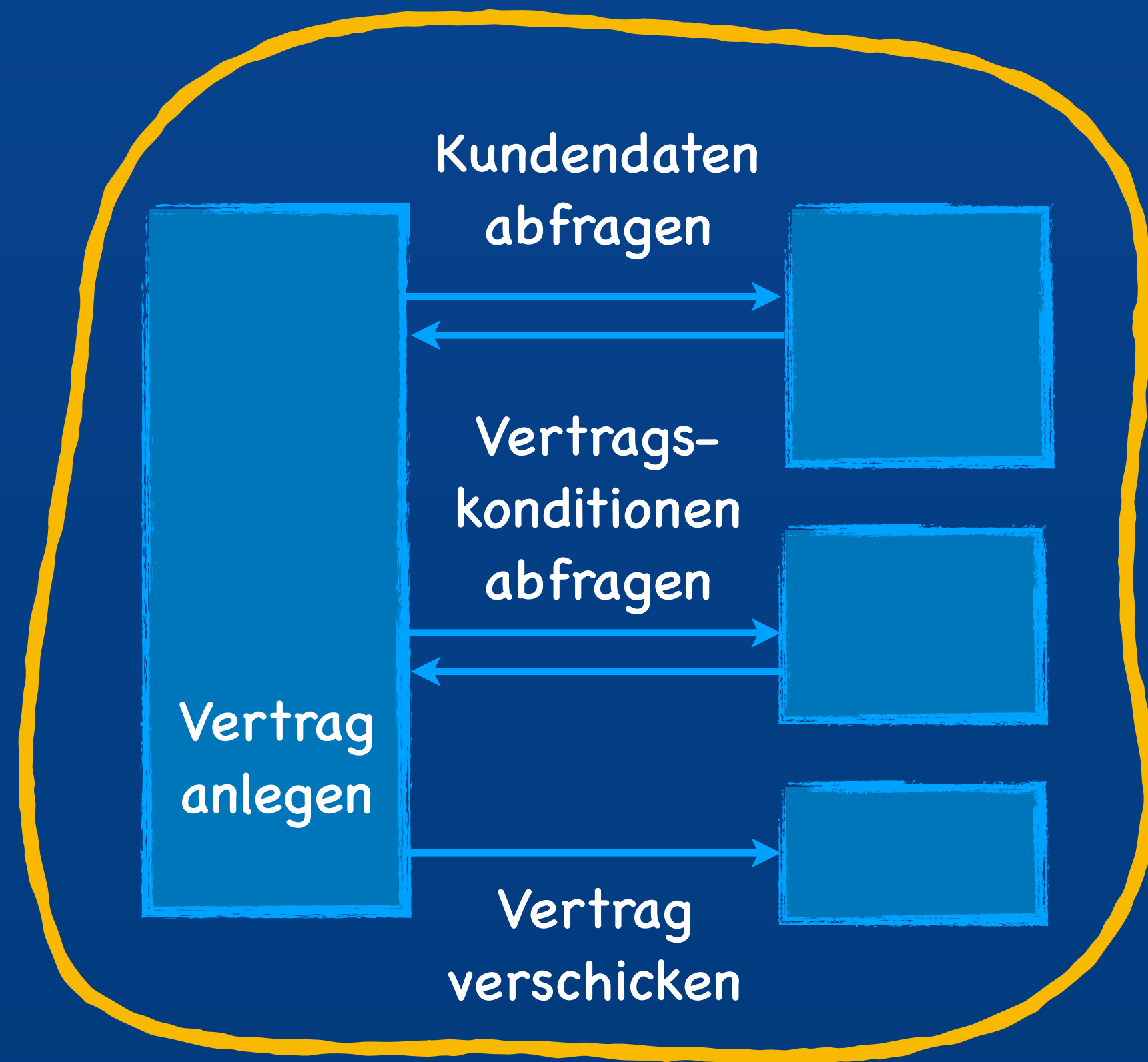
Data Contracts – „Daten-API“
Open Data Contract Standard (ODCS)



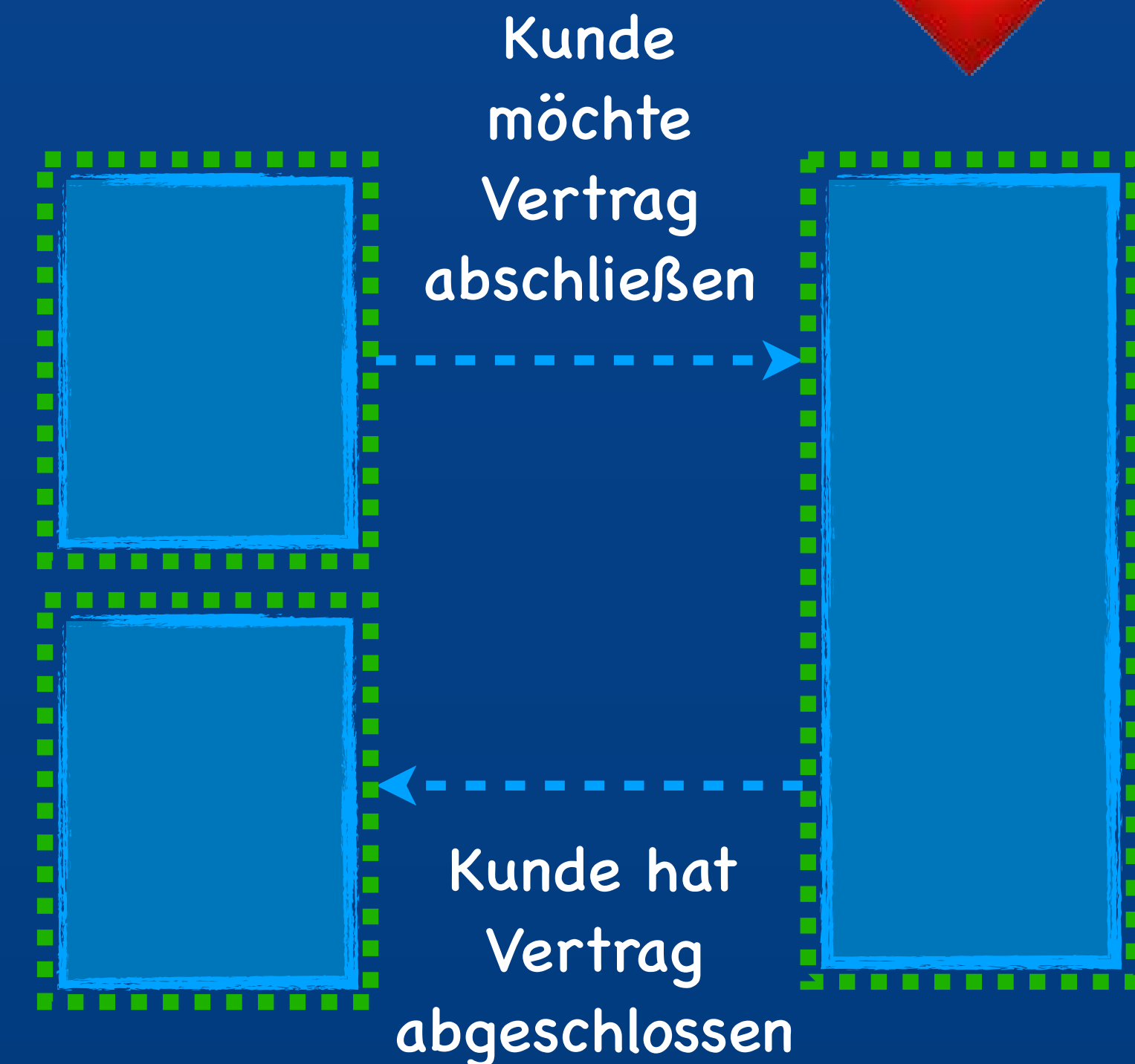
Ein paar
abschließende
Gedanken



Technische vs. fachliche APIs



evtl. Anzeichen für (Distributed)
„Big Ball of Mud (BBoM)“



„Self-contained Systems (SCS)“
Fachlich abgeschlossene (Teil-)Systeme

Testen Contract-Tests Fachlichkeit?



NEIN.

- ✓ Strukturelle Validität
- ✓ Statische Typsicherheit
- ✓ Beobachtbares Verhalten an einer Schnittstelle
- ✓ Integrations**kompatibilität**

- ✗ Geschäftslogik
- ✗ Datenrichtigkeit
- ✗ Fachliche Korrektheit



Fachliche Tests (Unit-Tests, Integrationstests)
Nur, wenn nicht über andere Testarten testbar: API-Tests

APIs und „Postel’s Law“

*“be conservative in what you do,
be liberal in what you accept from others”* *Sei präzise beim Senden und
tolerant beim Empfangen.*

– RFC 671 (TCP, 1974), „Robustheitsprinzip“, Jon Postel zugeschrieben

- ✓ Vorwärtskompatibilität: Unbekannte Felder ignorieren
- ✓ Optionale Felder nicht zwingend machen
- ✓ Toleranz bei Reihenfolge und Whitespace

- ✗ Falsche Datentypen akzeptieren
- ✗ Ungültige Werte stillschweigend korrigieren

etc.



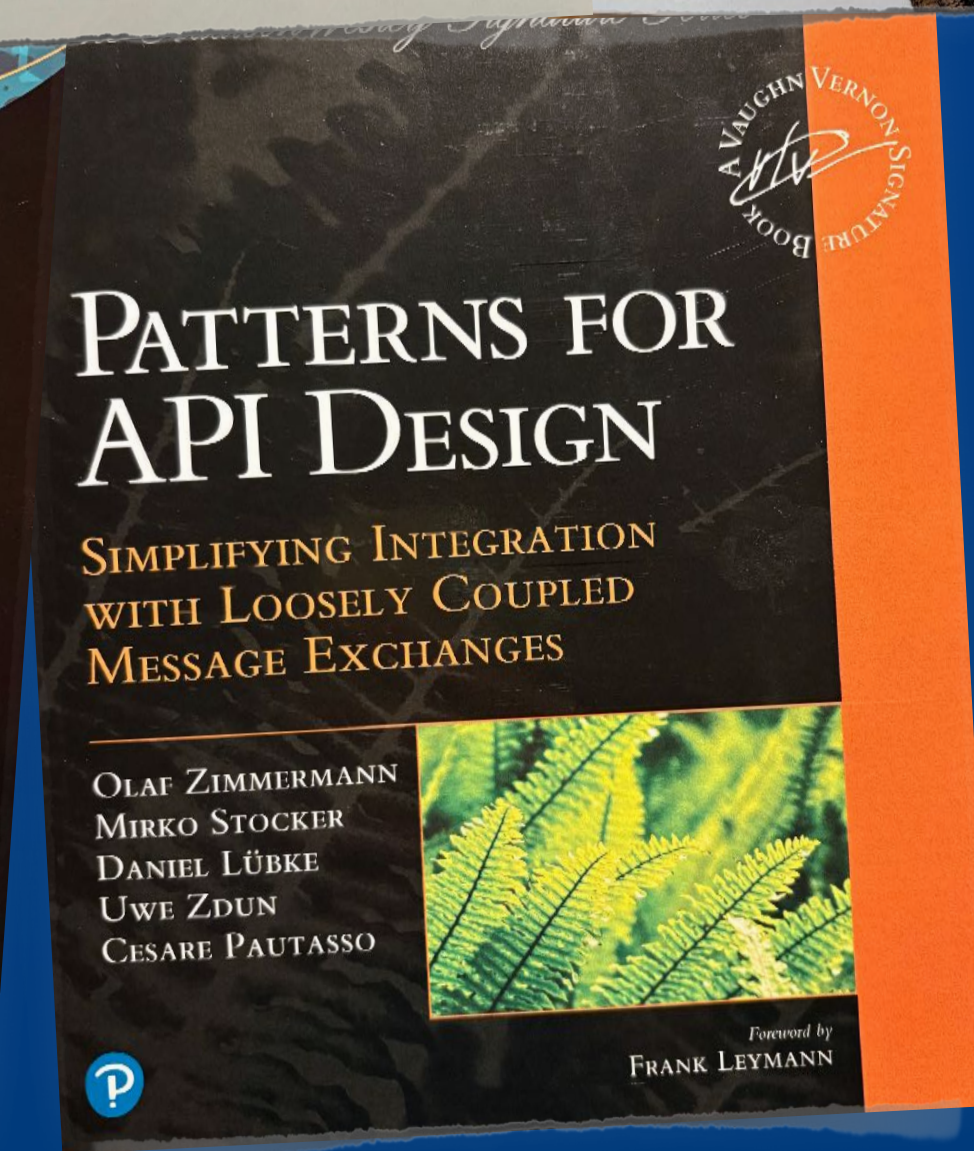
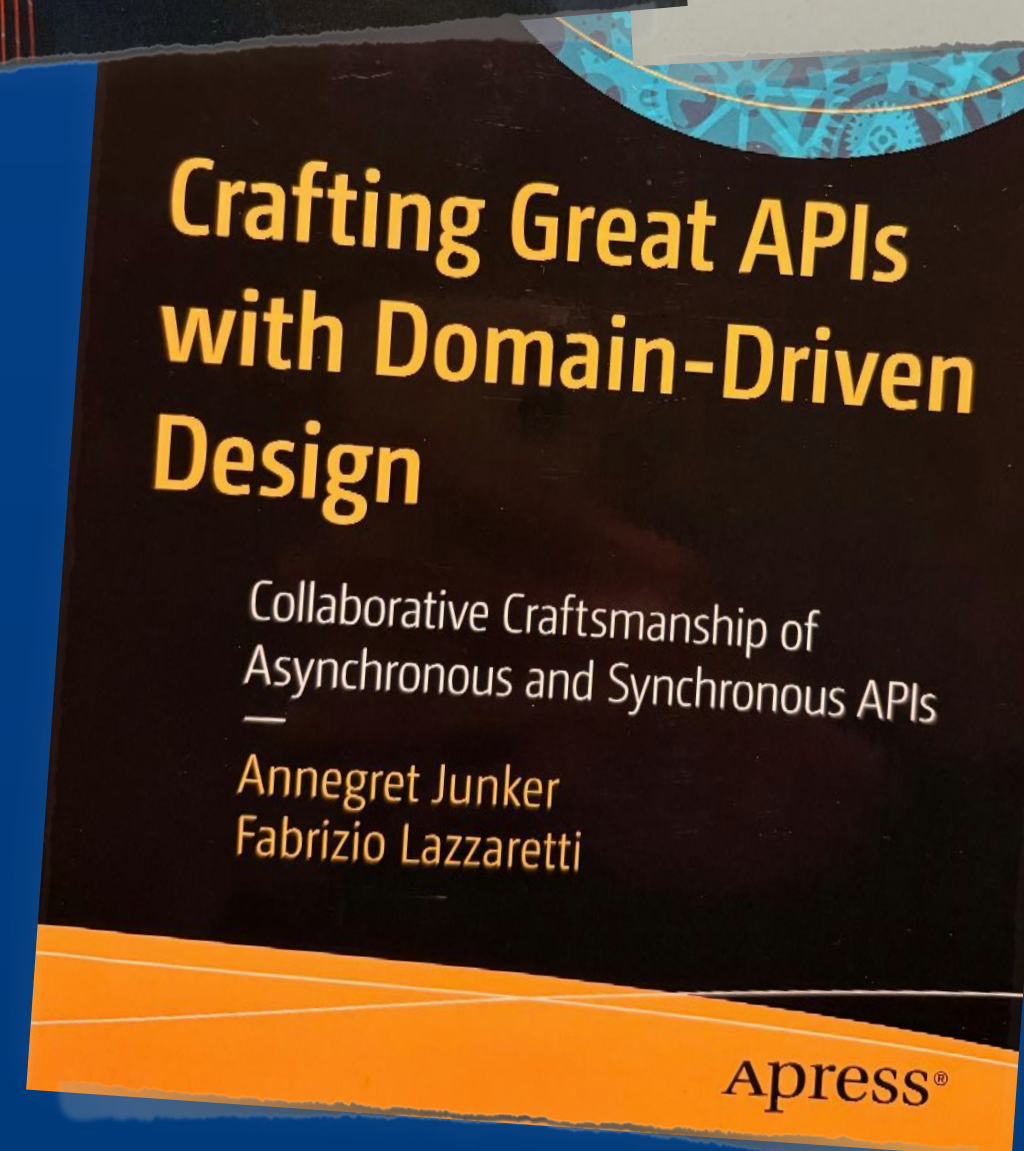
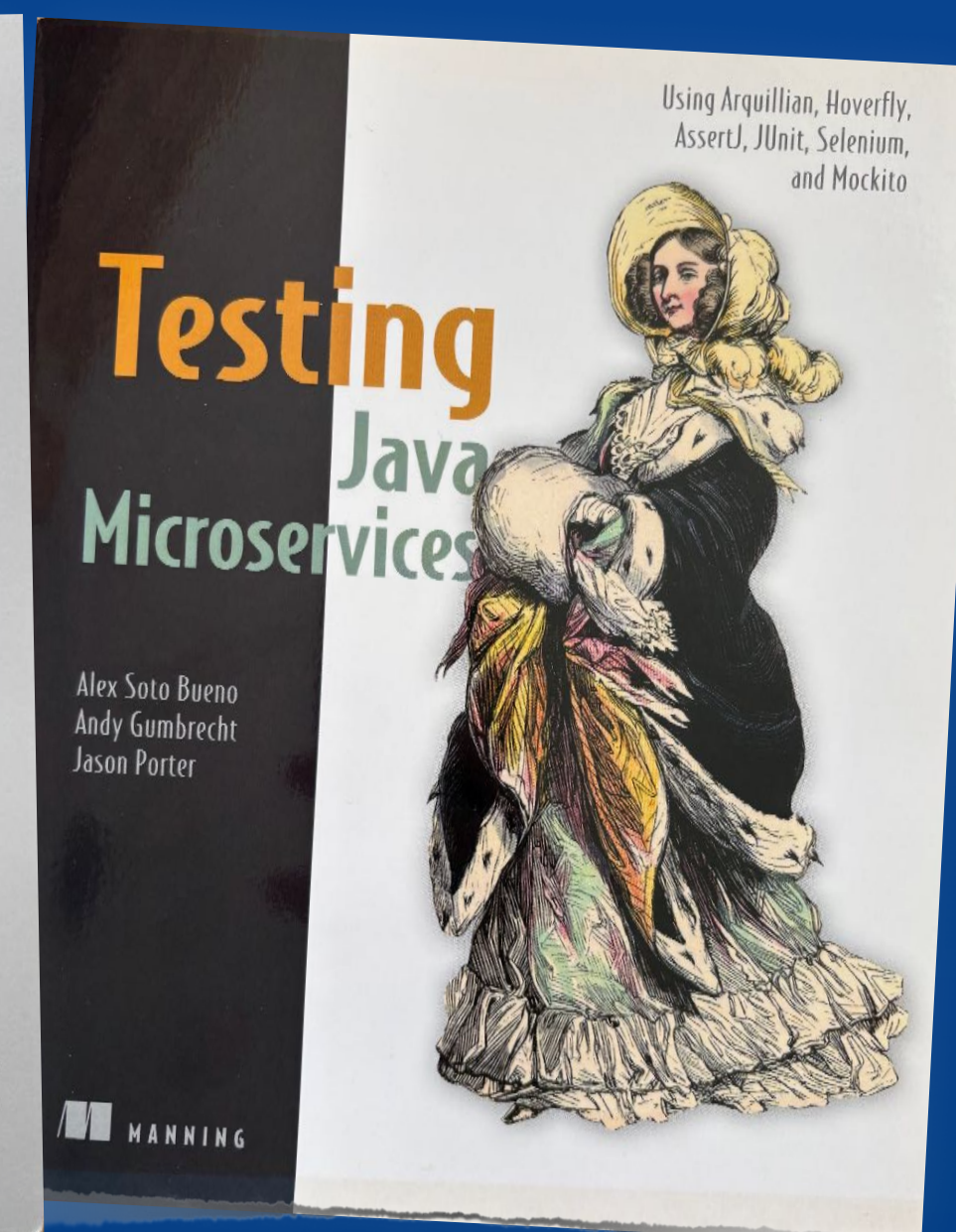
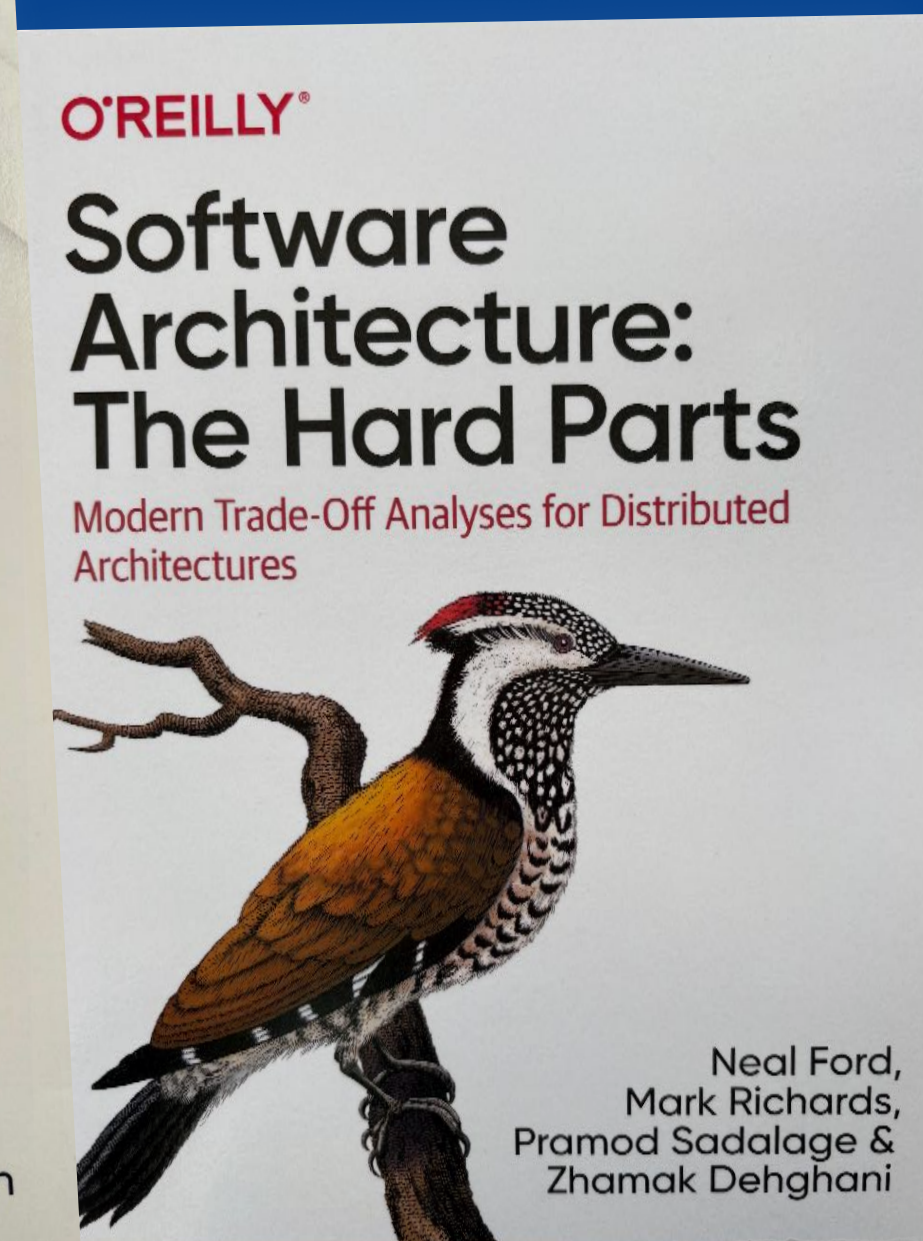
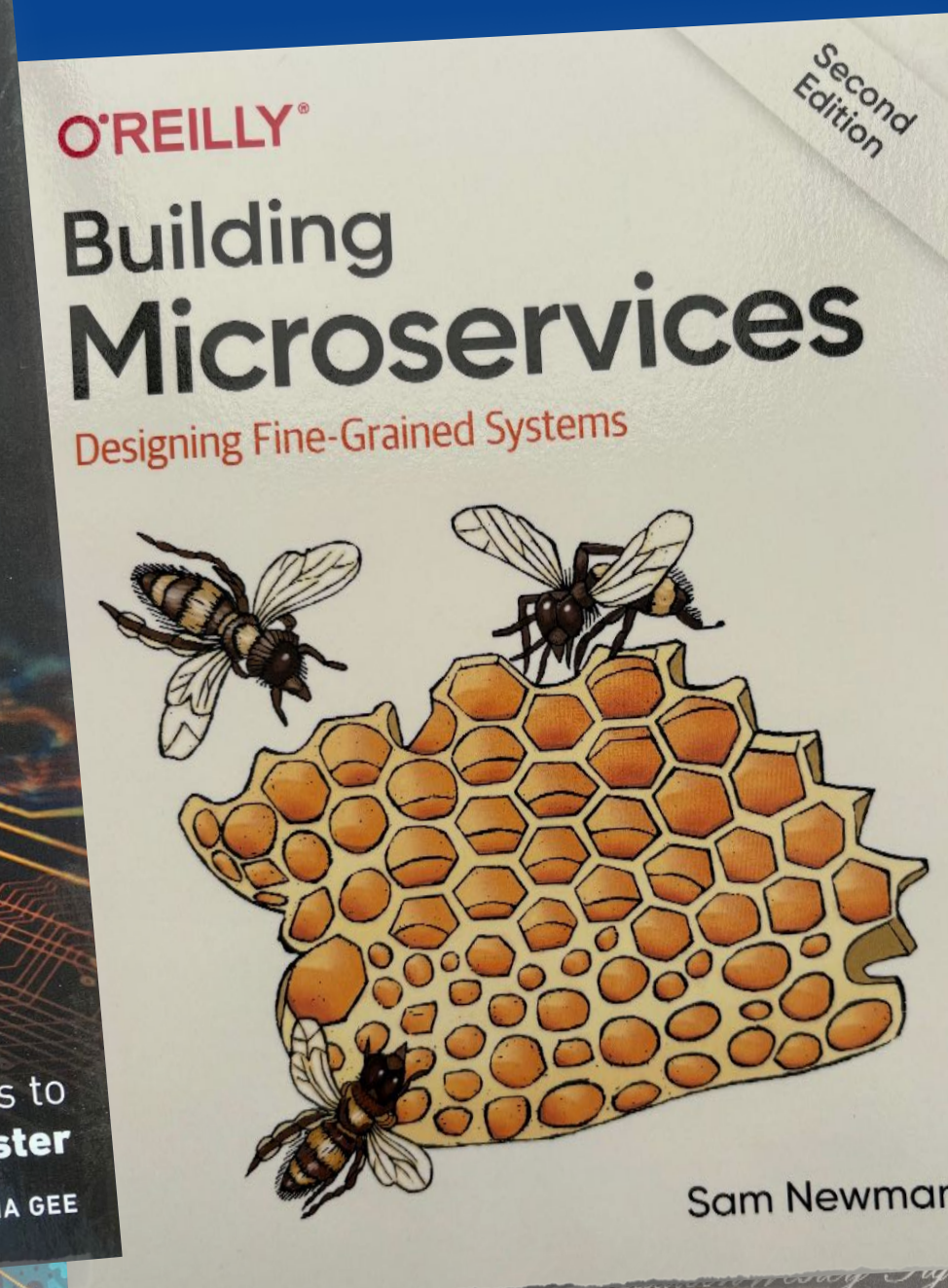
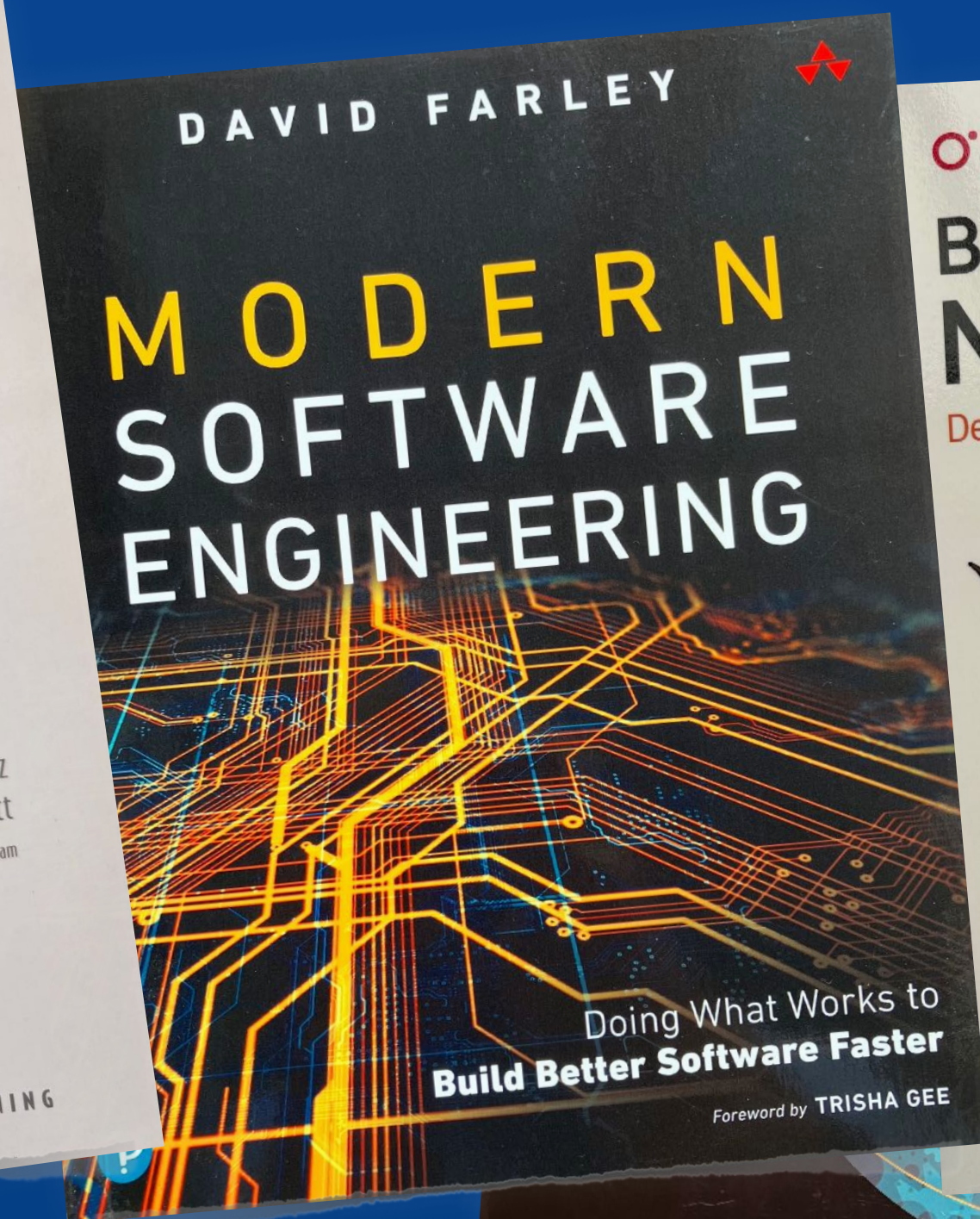
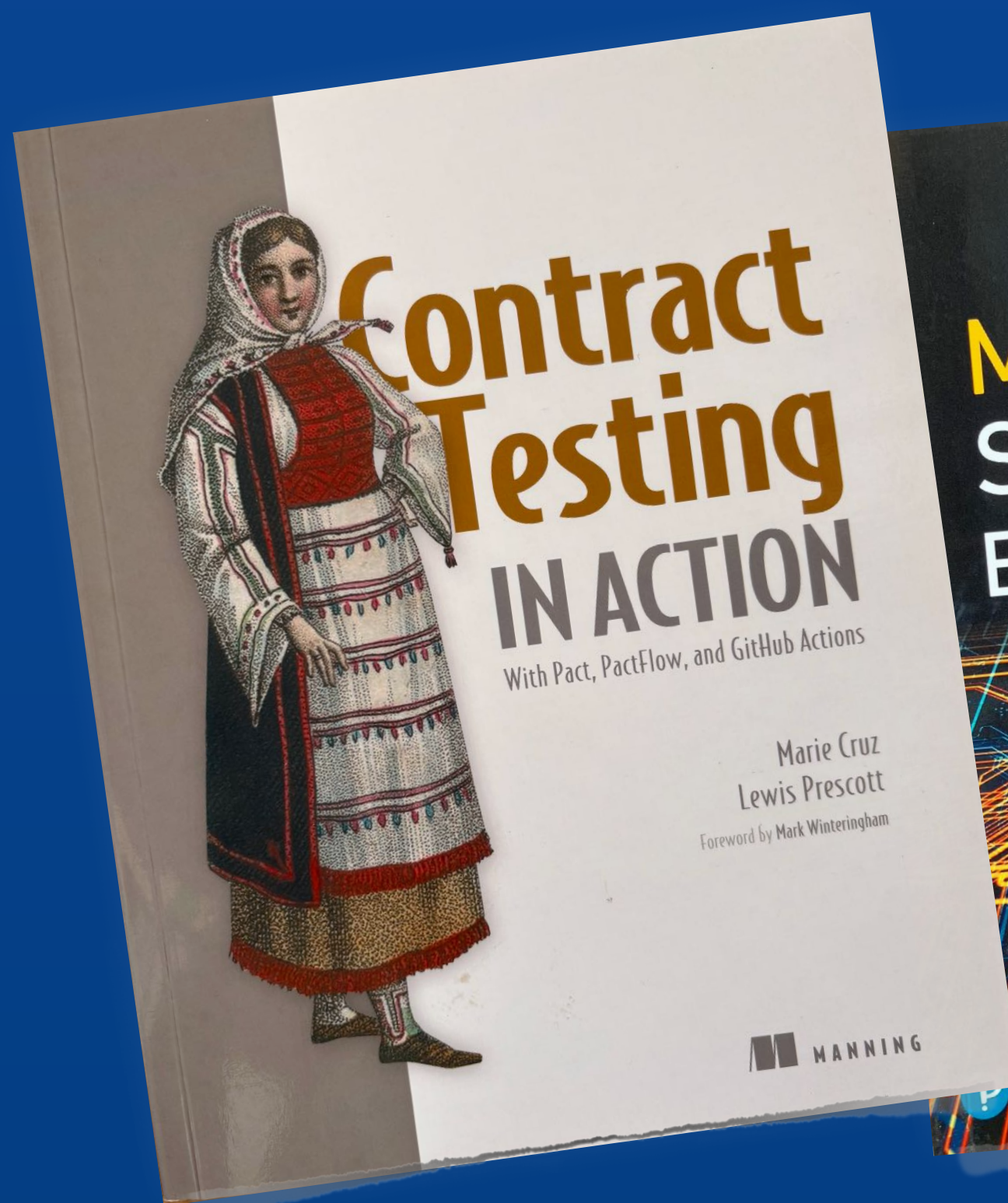
*Sei präzise beim Senden
und überlege gut,
wo du bei Empfangen tolerant sein kannst/solltest.*

Fazit: API-Tests, Contract-Tests etc.

Unterschiedliche Sichtweisen:
Wer möchte was sicherstellen?

(Consumer-driven) Contract-Testing:
Sicherheitsnetz für Integrationskompatibilität.
Weniger E2E-Tests nötig, aber kein kompletter Ersatz!

Wie gut sind eure APIs testbar?
Wie sind die Systeme ge-/entkoppelt?
HTTP/REST oder Events oder ... ?
Das sind Architekturentscheidungen!





www.tk.de/IT



[*'to:mas*] 🖐️

📧 🦋 📺 🐙 @thmuch



innoc.com/de/articles/2016/09/consumer-driven-contracts/

microsoft.github.io/code-with-engineering-playbook/automated-testing/cdc-testing/

martinfowler.com/articles/consumerDrivenContracts.html

pact.io

spring.io/projects/spring-cloud-contract