


Grundlagen und Aspekte der API Kompatibilität

Und wie man diese
nachhaltig beherrscht



 @pfleidfn



 @nik101010

API

noun COMPUTING

a set of functions and procedures that allow the creation of applications which access the features or data of an operating system, application, or other service.

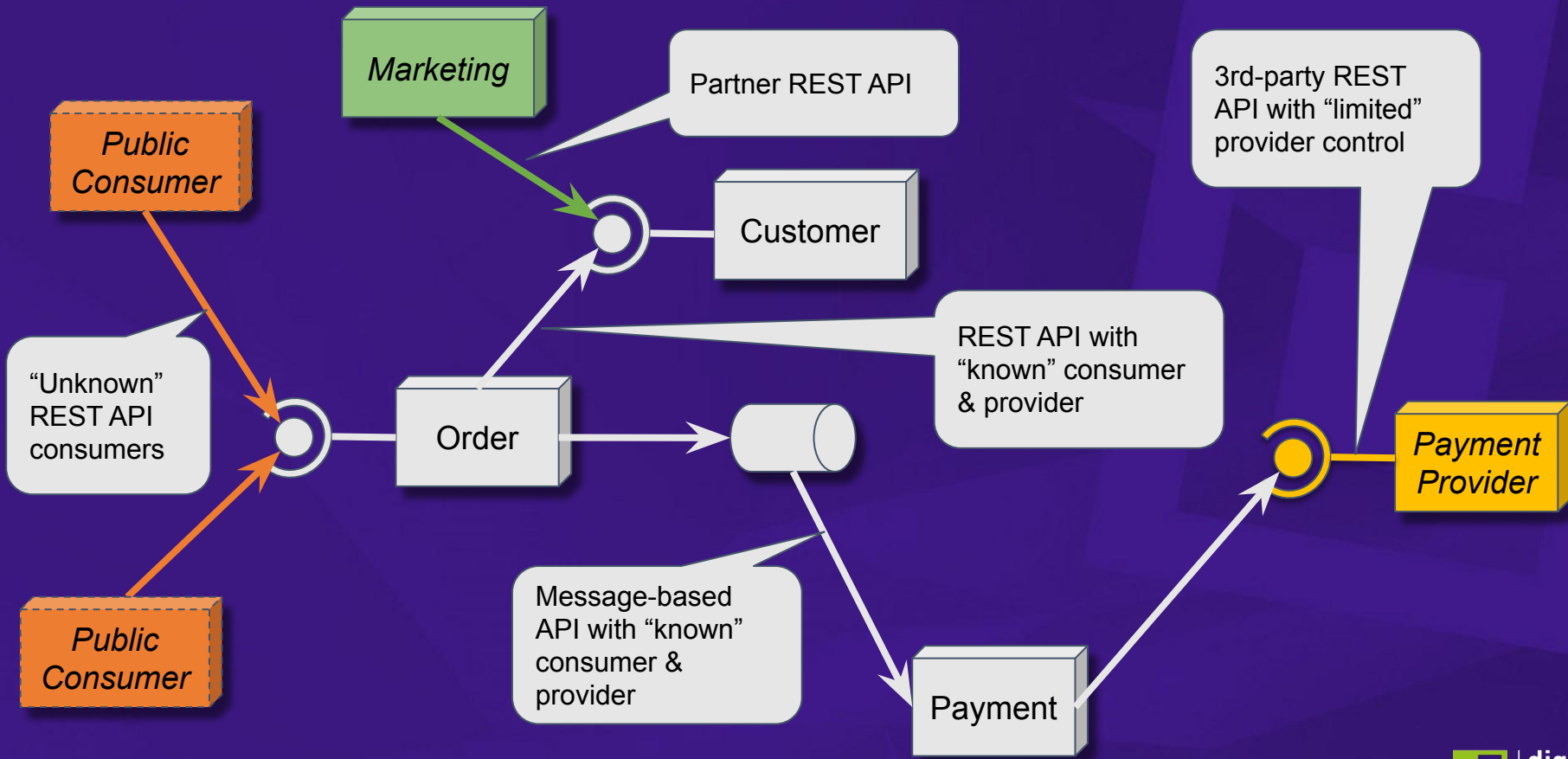
data of an operating system, application, or other service.

a set of functions and procedures that allow the creation of applications which access the features or

noun COMPUTING

Quelle: <http://www.google.de>

API Types

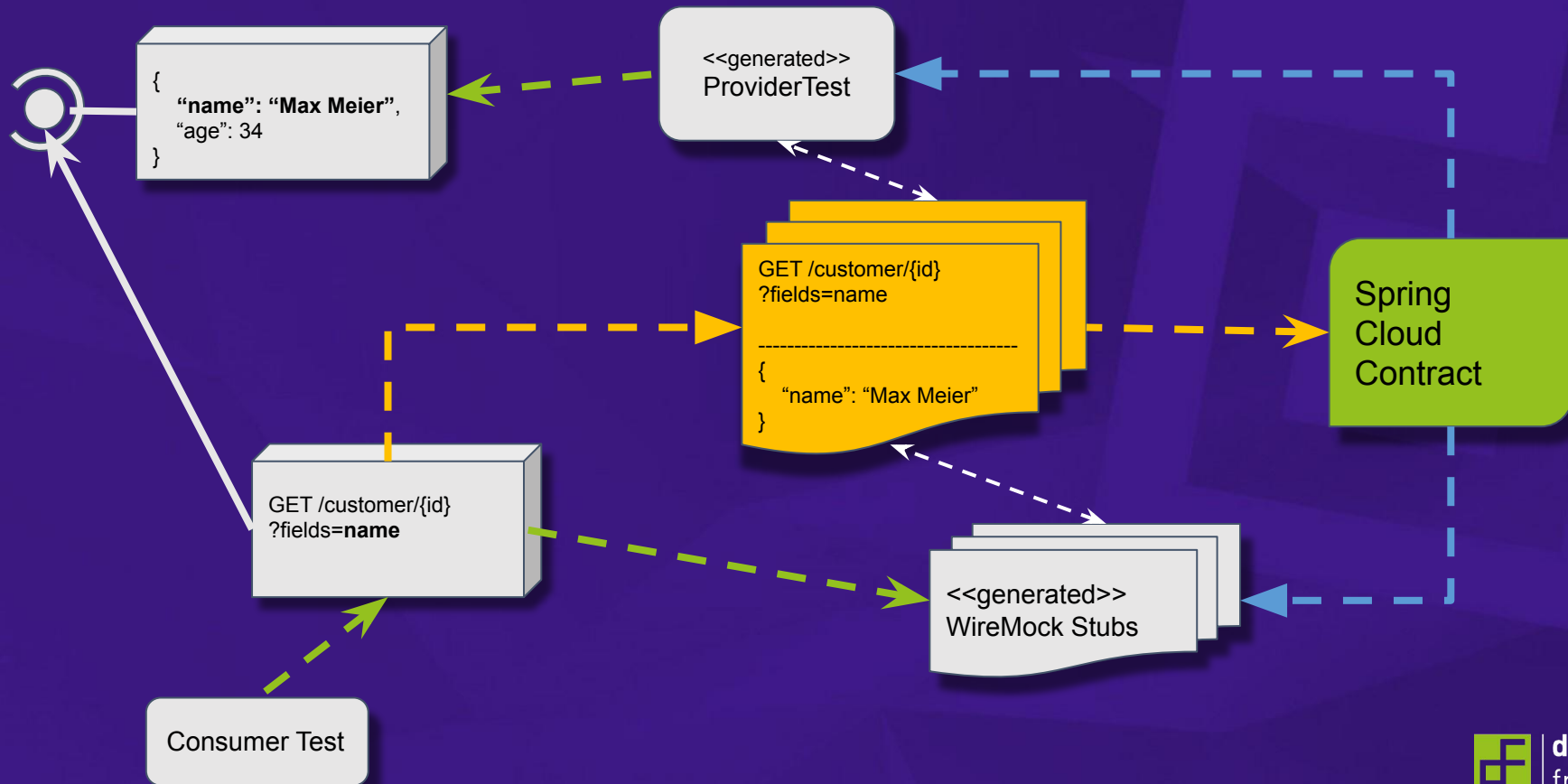


API evolution is the concept of striving to maintain the "I" in API, the request/response body, query parameters, general functionality, etc., only breaking them when you absolutely, *absolutely*, have to. It's the idea that API developers bending over backwards to maintain a contract, no matter how annoying that might be, is often more financially and logistically viable than dumping the workload onto a wide array of clients.

array of clients.

ally and logistically viable than dumping the workload onto a wide ar-

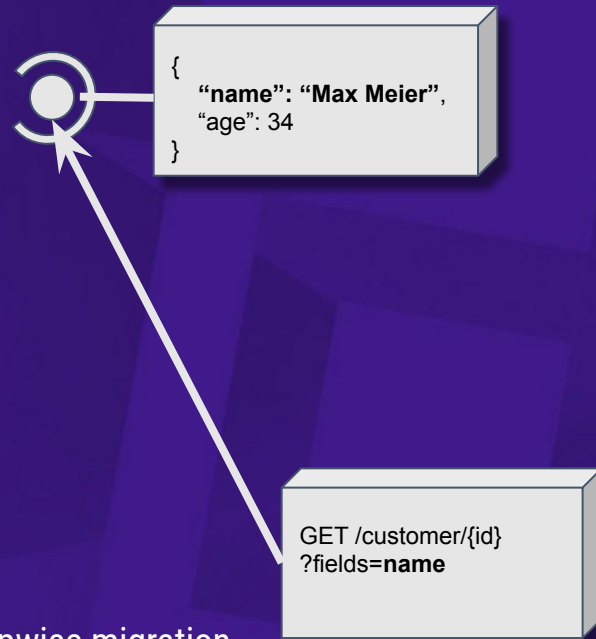
Consumer Driven Contract Testing



API Evolution

“Things have changed since WSDL back then ...”

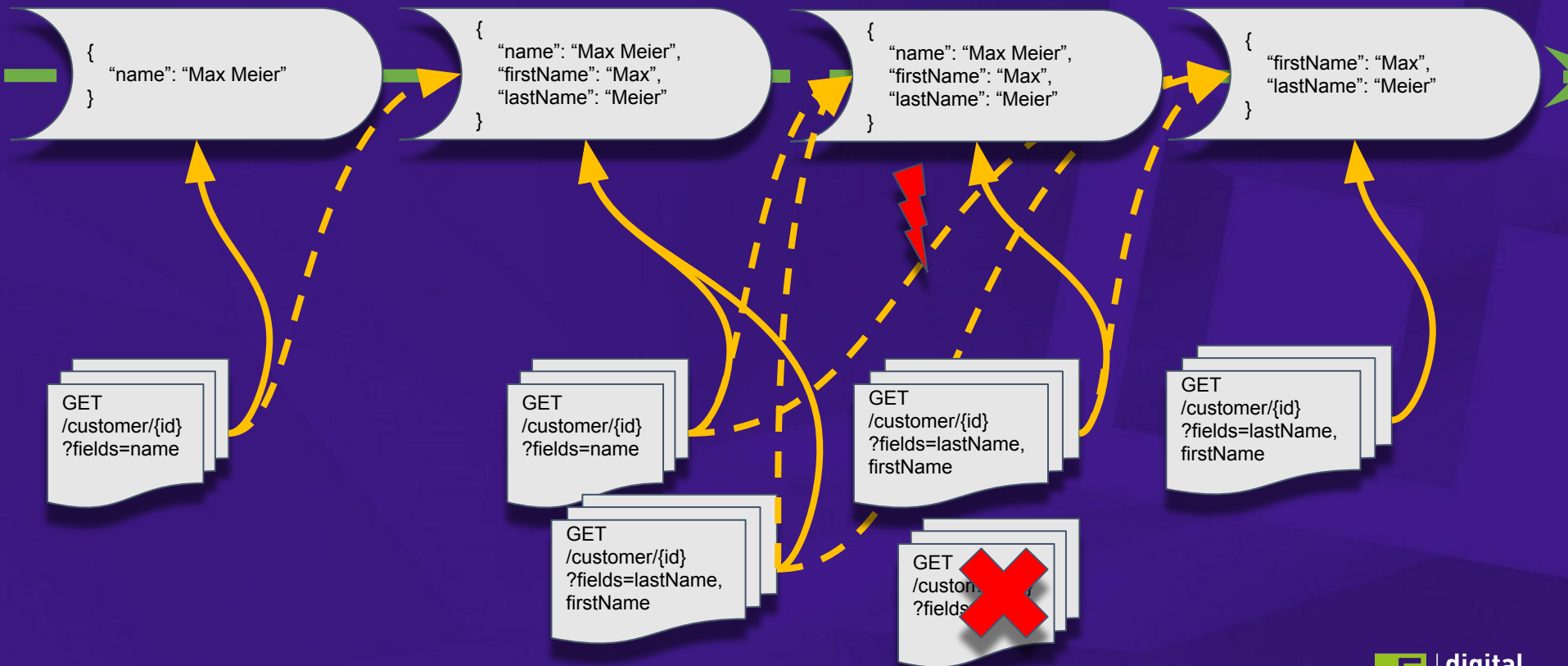
- Relaxed Schema
 - Resource based
 - Tolerant Reader
 - Relaxed type system based on JSON
 - No strict schema validation
- Maintenance
 - Evolution ideally consumer “driven”
 - Number of consumers outweighing providers
 - Effort for maintaining compatibility at provider
 - Avoid coupling, especially lock-step deployments, using stepwise migration
- Versioning
 - Last resort



Consumer-Provider Relationships



Managing API Compatibility



Live Demo

Spring Cloud Contract

Dimensions of compatibility

“But wait, there’s more ...”

- Schema compatibility
 - Can be automatically tested with CDCT
 - Tolerant Reader: No strict schema validation
 - Beware of cumulated changes when deploying
- Semantic compatibility
 - Fields can change their meaning over time
 - Expected content for fields can vary
 - Might lead to consumer or producer malfunctions
- Compatibility of state
 - Even when APIs syntactically match, services might make differing assumptions for system state

Summary

1. Use Consumer Driven Contract Testing to decouple your development life-cycles
 - a. Ideal for known consumers/provider
 - b. Regularly check 3rd-party APIs using “proxy” contracts
 - c. Offer contracts to public consumers?
2. Preserve API compatibility by avoiding breaking changes
 - a. Consolidate consumer driven API changes
 - b. Verify backward compatibility using Cloud Pipelines
3. If necessary, use stepwise migration
 - a. Verify backward compatibility using Cloud Pipelines
 - b. Let consumers drive the breaking change
 - c. Keep track of consumers/providers still relying on older versions

How are you dealing with maintaining API compatibility?



 @pfleidfn



 @nik101010

<https://blog.digitalfrontiers.de>

<https://www.digitalfrontiers.de>

#WeAreHiring



digital
frontiers