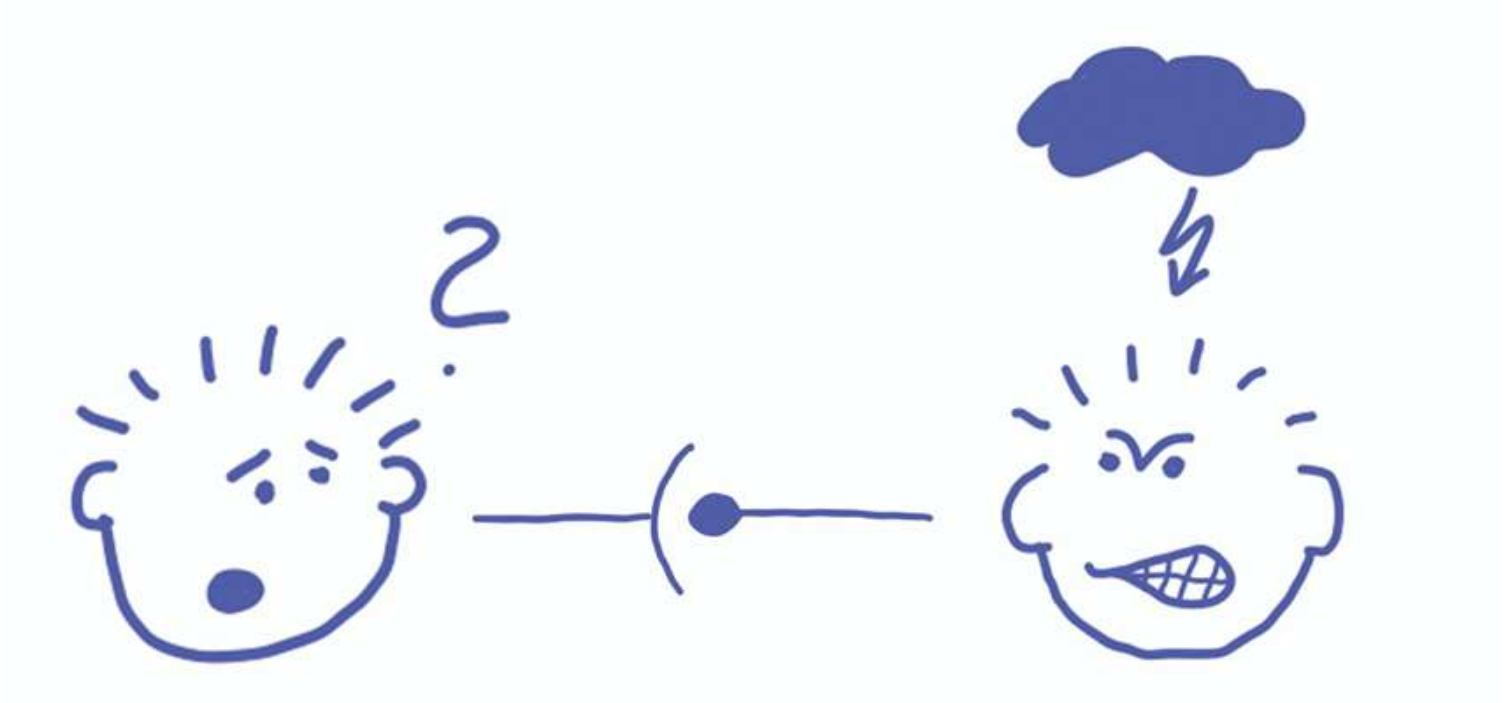


The best or nothing! (?)

What does it mean für your API product?

Katja Weisheit
Mike Horn

Softwerkskammer Chemnitz

PENTACOR
ARCHITECTING
YOUR
BUSINESS

Katja is an API enthusiast and works as an IT architect at pentacor

katja.weisheit@pentacor.de

Mike works as a Senior Architect at pentacor

mike.horn@pentacor.de

We love interactive sessions

- Raise questions directly or use the chat
- Discussion is always welcome
- Your experience is always welcome

Really a great product or isn't it?

Image Source: https://mercedes-benz.de

This product goes digital and provides a great UI

Image Source: https://mercedes-benz.de

Is one UI enough for building digital UseCases?

No! … A great digital product should provide APIs to make further digital experiences happen.

Image Sources: https://developer.mercedes-benz.com

Some digital experiences based on APIs

Predictive diagnostic
- Remote Diagnostic API

360° Vehicle View
- Vehicle Status API

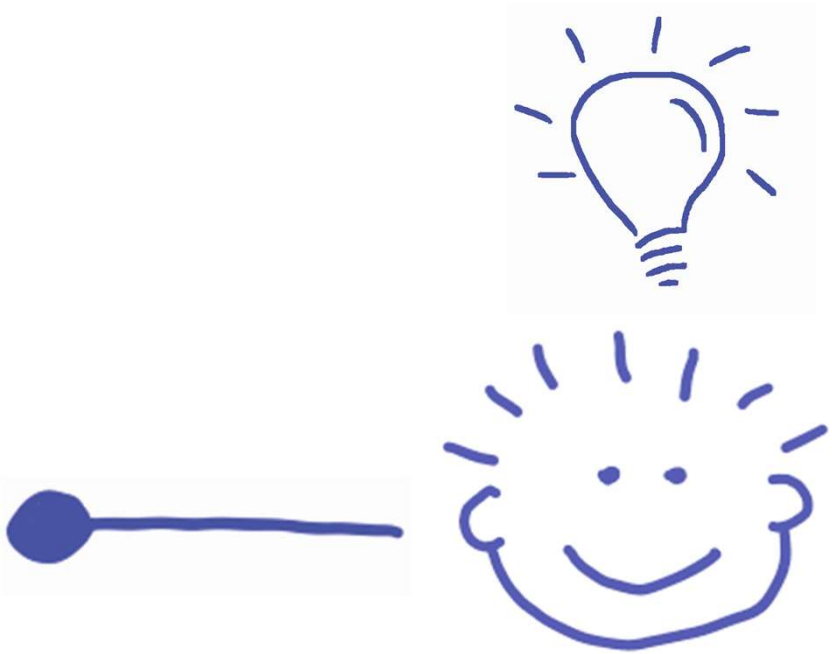
Carsharing
- Vehicle Lock API
- Vehicle Engine API
- Insurance API

Pay as you drive insurance
- Vehicle Odometer API

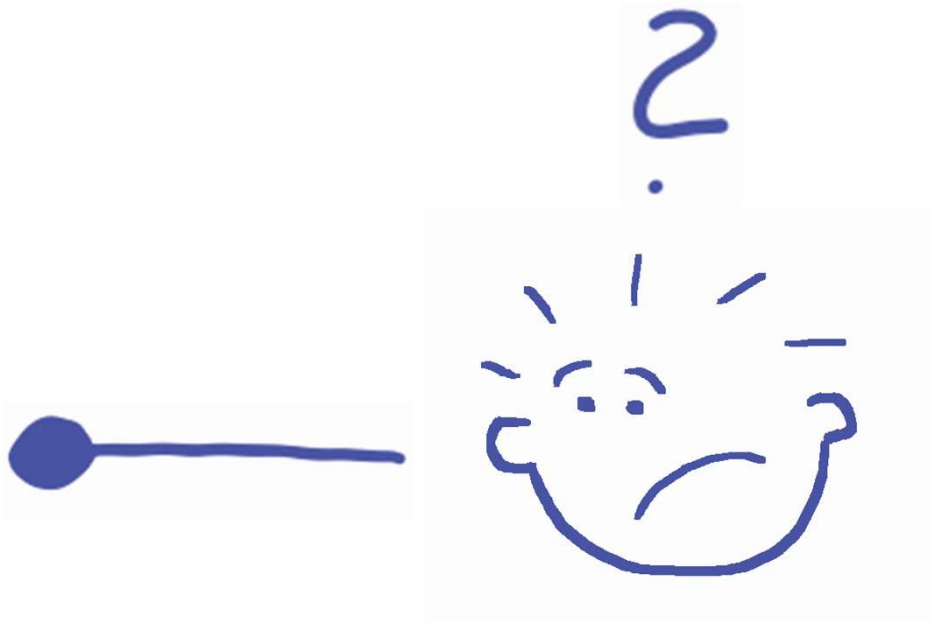
Delivery to trunk
- Vehicle Lock API

Image Sources Mercedes-Benz: https://developer.mercedes-benz.com
API Products: https://developer.mercedes-benz.com
Hint: Some APIs are not real, used for illustration purposed only

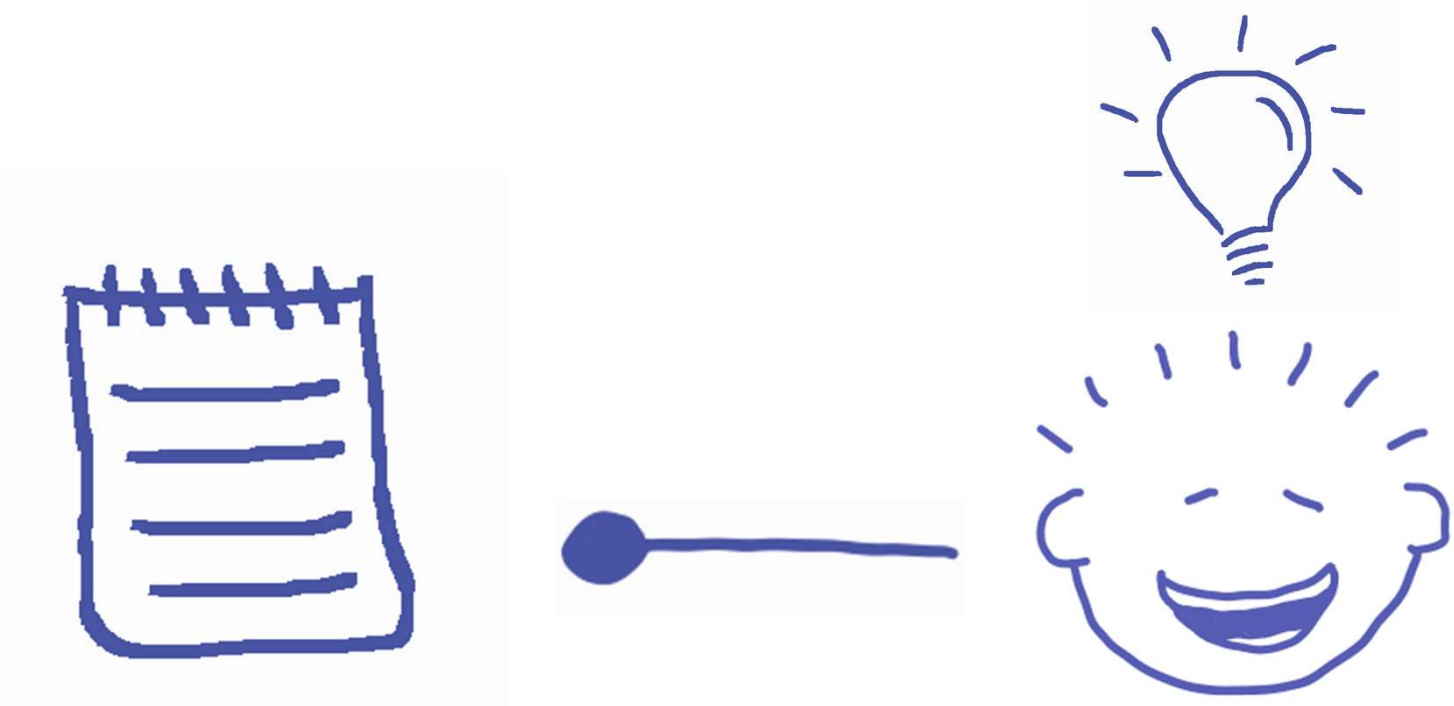
Hey – let´s expose an API to others



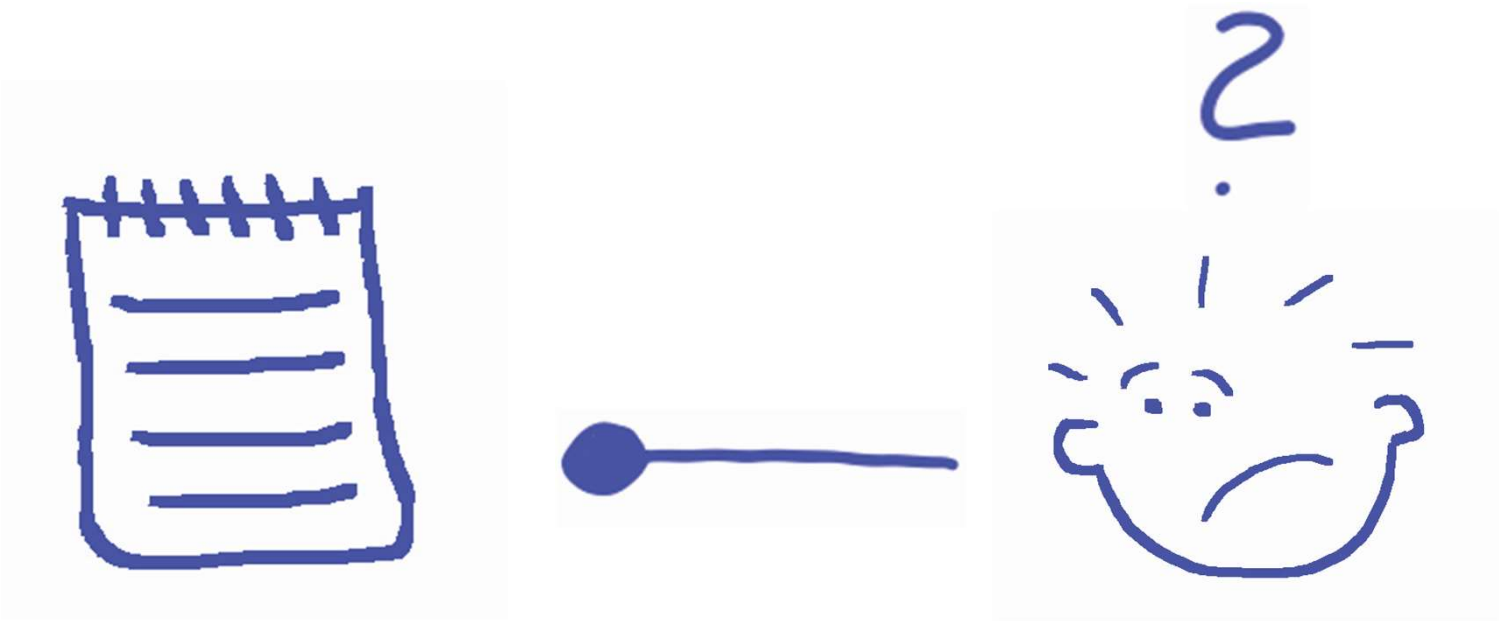
Nobody out there who uses it?

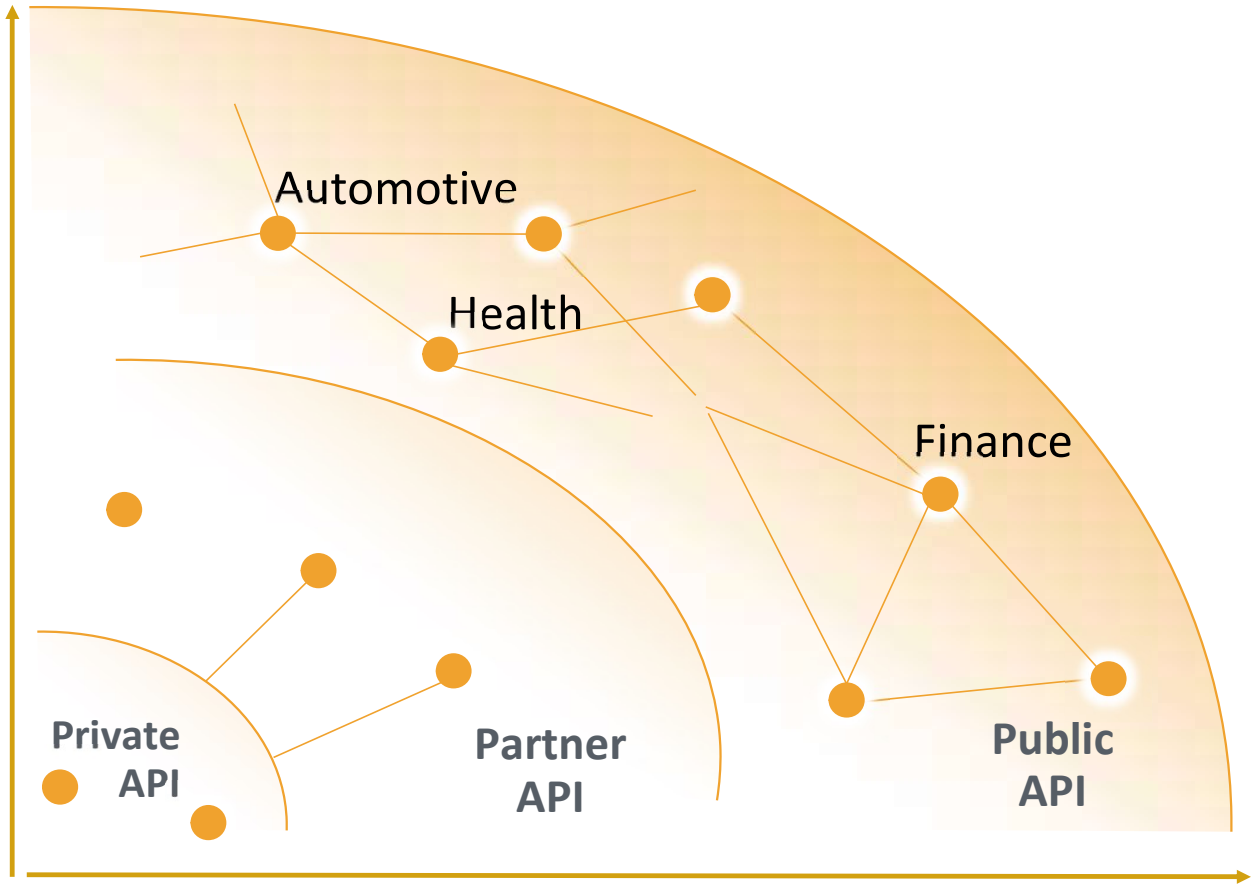


Might be a good idea to document it!

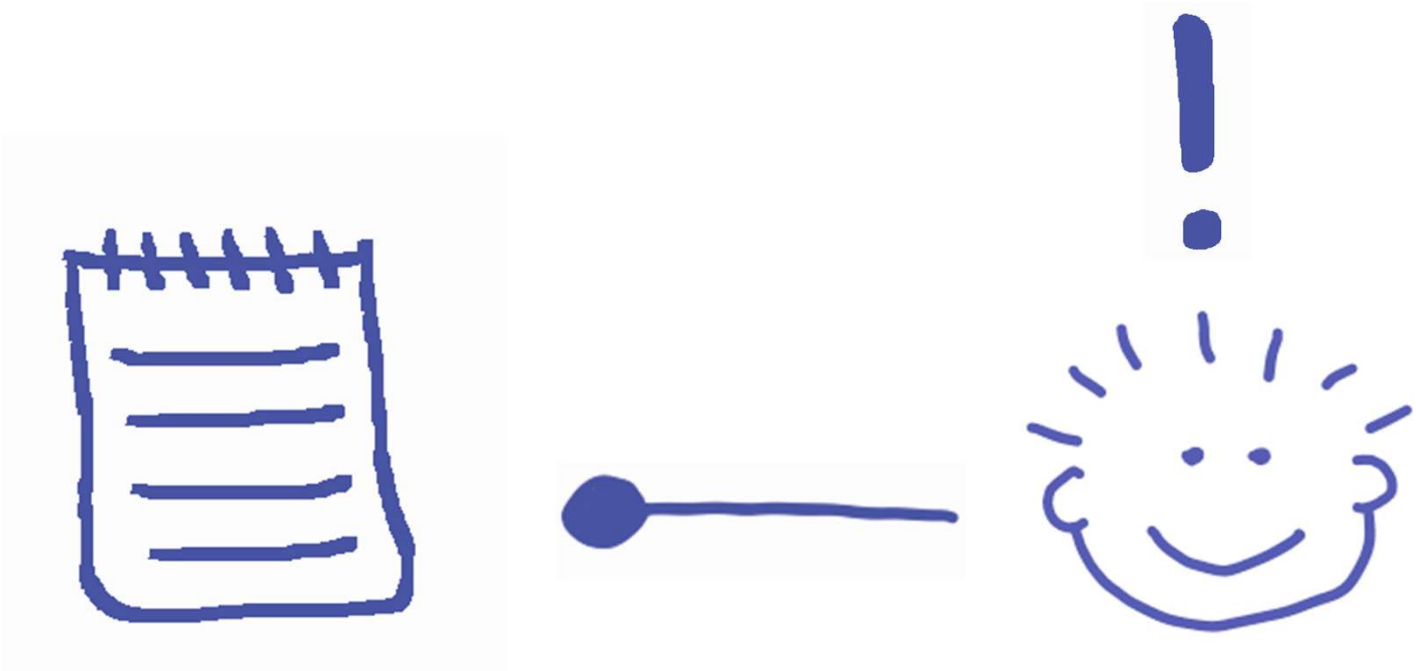


But where? And in / for which ecosystem?




Automotive
Health
Finance
Private API
Partner API
Public API

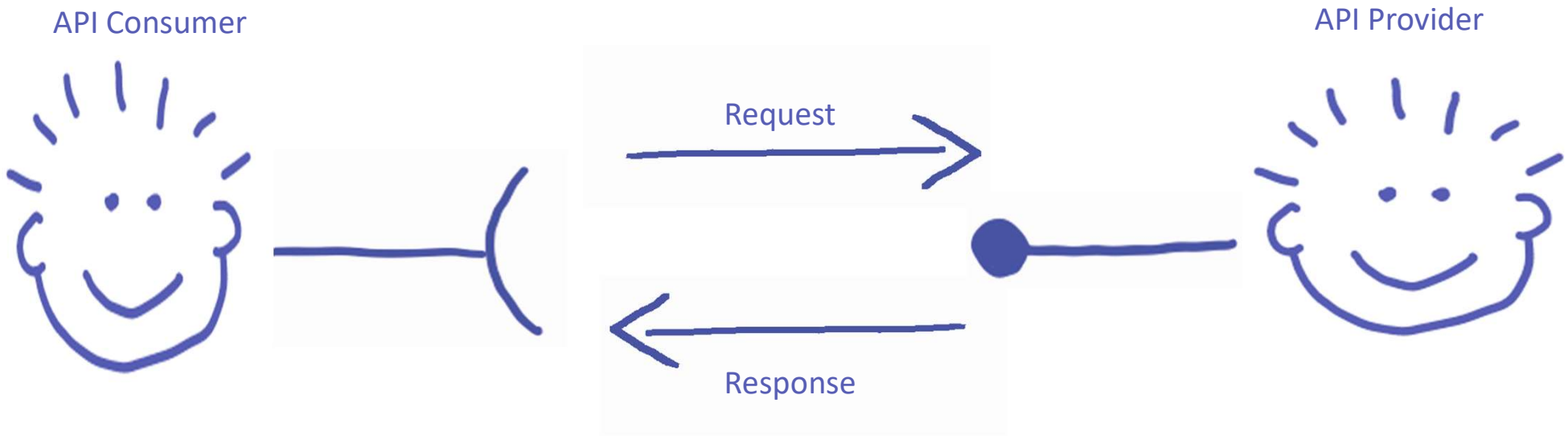
No matter what type of API - document it!



API Integration scenarios ??

How to make your digital function or data accessible?

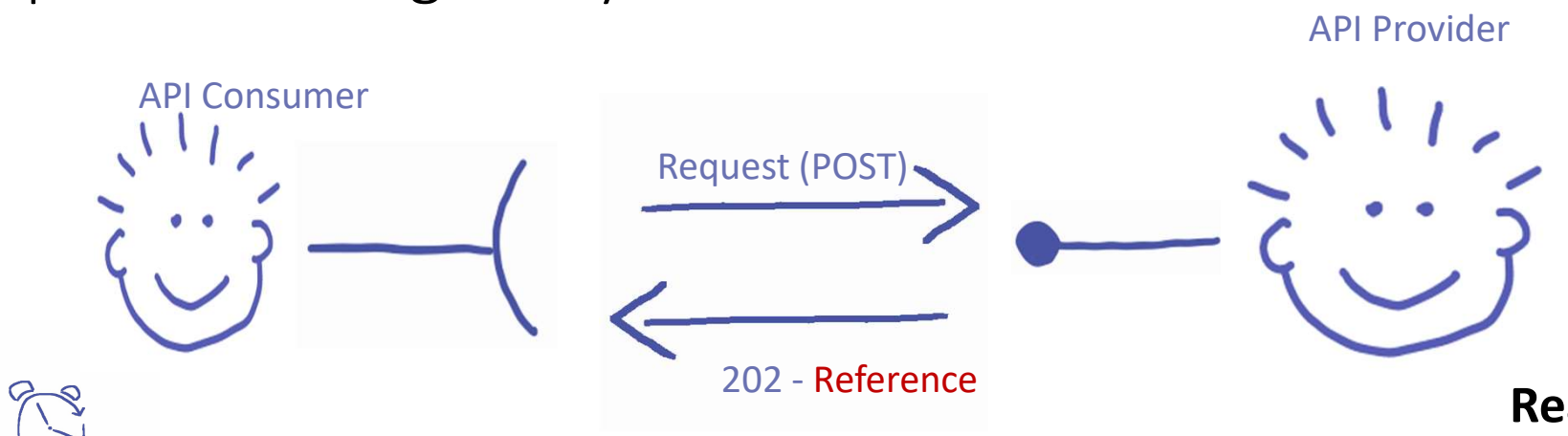
Synchronous Integration against your API



Realization options:

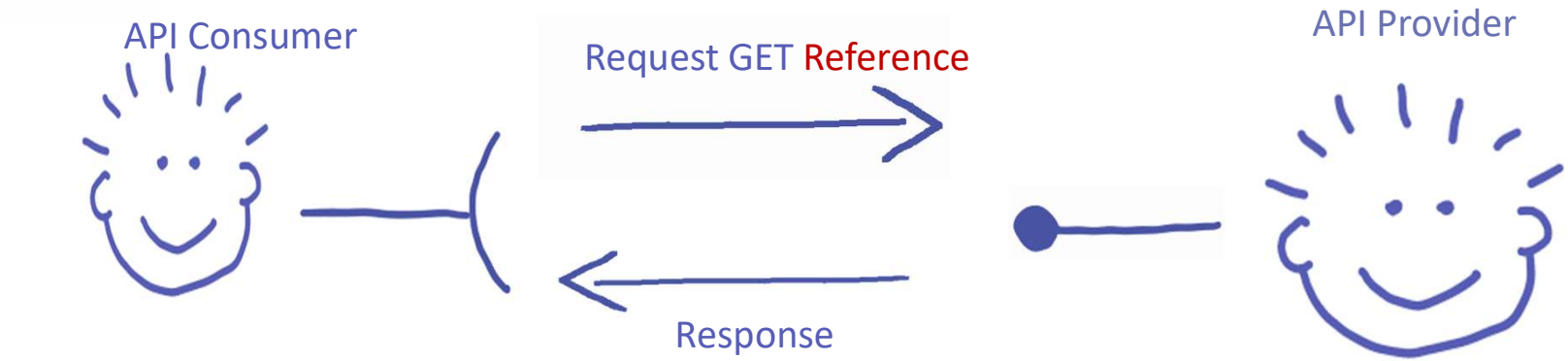
- SOAP/http
- REST/http

More decoupling necessary: Realize **asynchronous** integration possibilities against your API

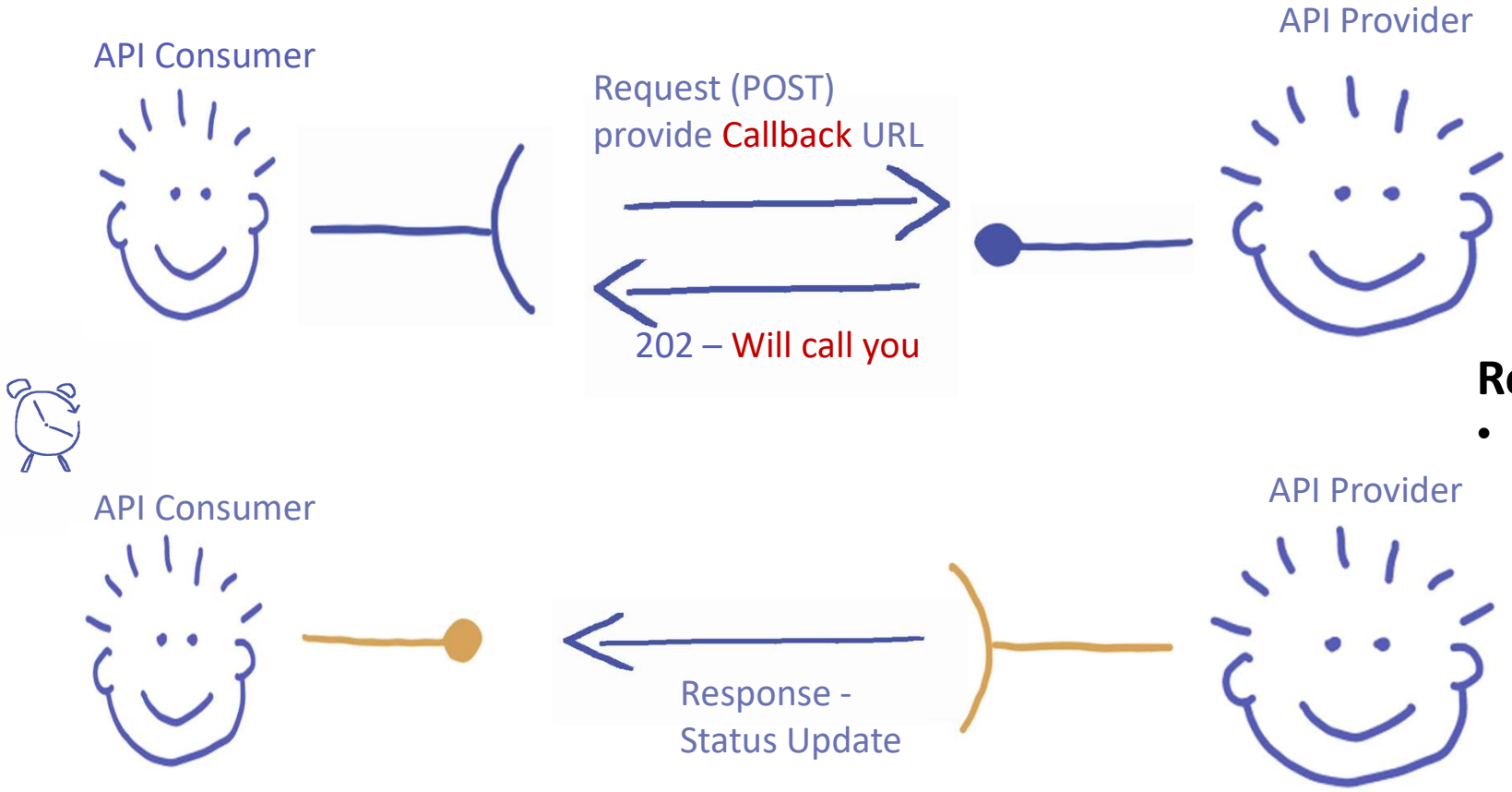


Realization:

- Poll by reference



More decoupling necessary: Realize **asynchronous** integration possibilities against your API



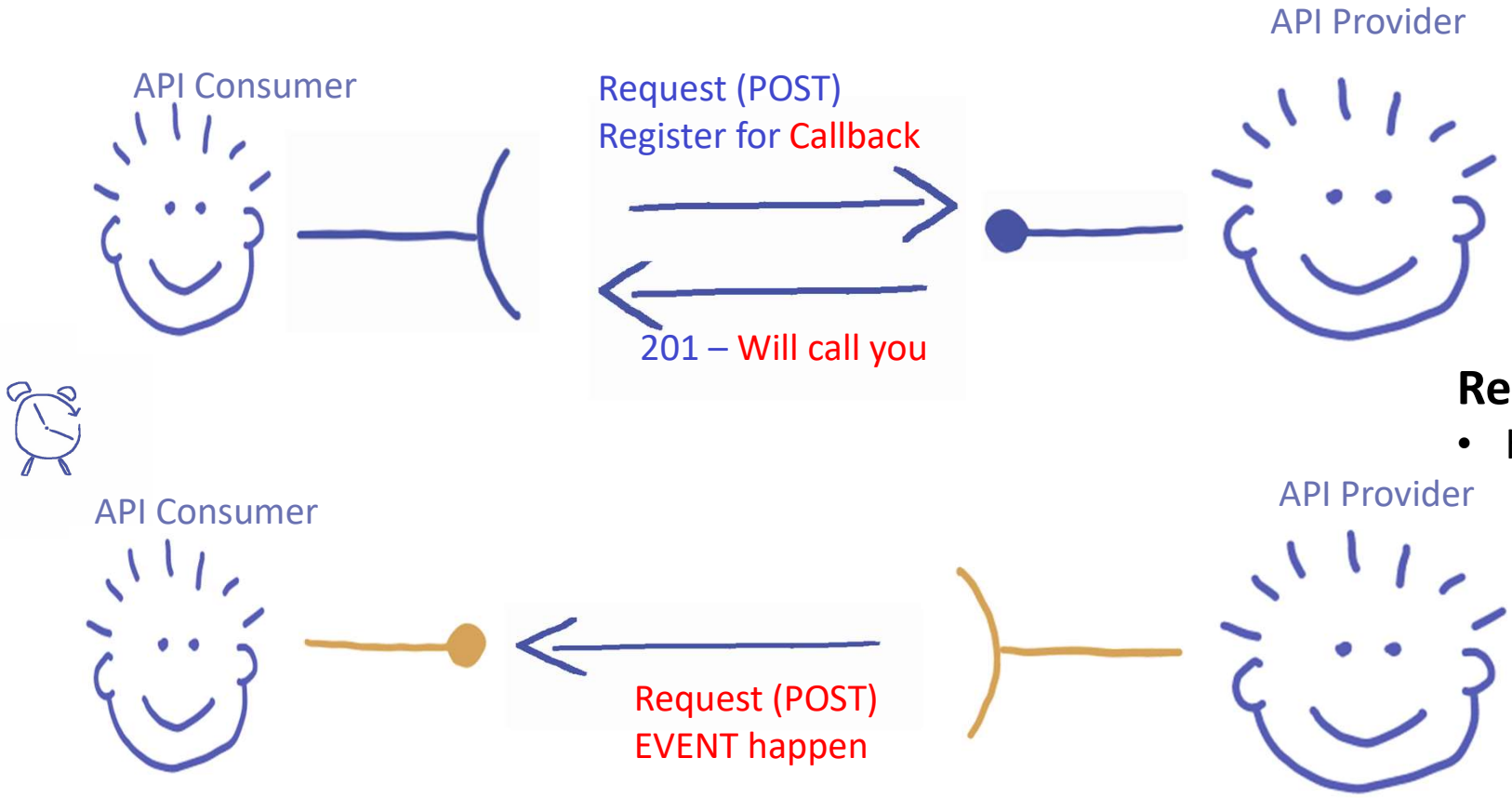
Realization:

- Provide Callbacks

Much more de-coupling necessary?

Provide your **events** and let the consumer decide how to work with them …

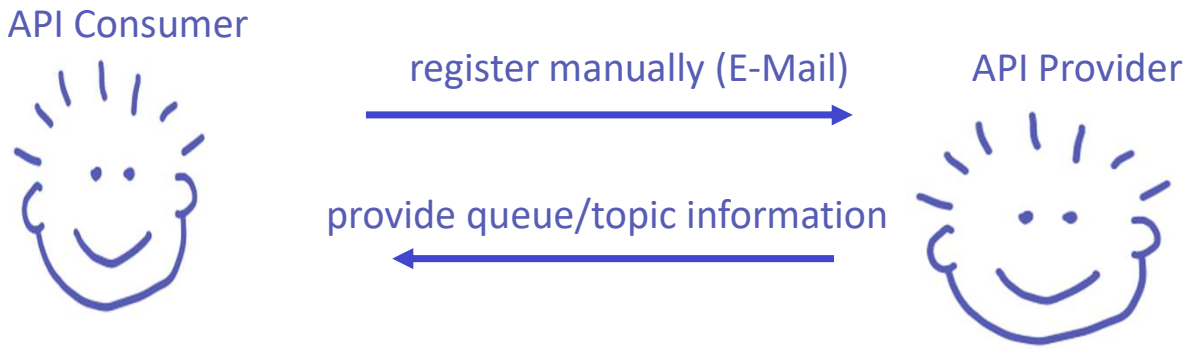
Event propagation: Publish/Subscribe

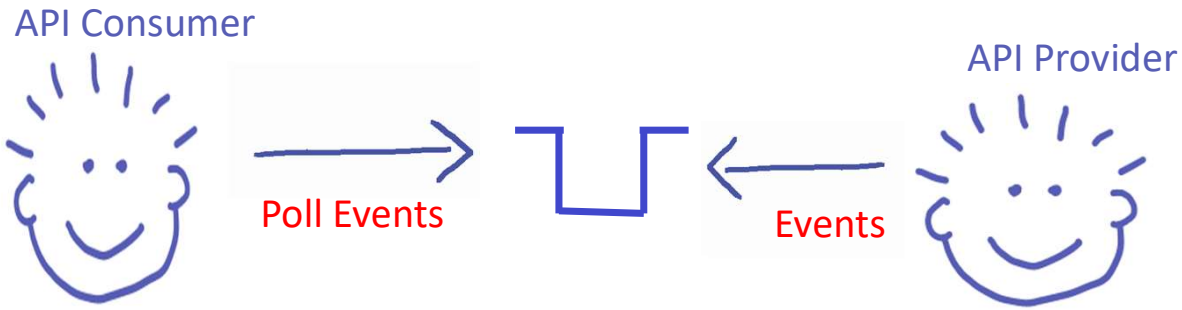


Realization:

- http based Pub/SUB

Event propagation: Publish/Subscribe





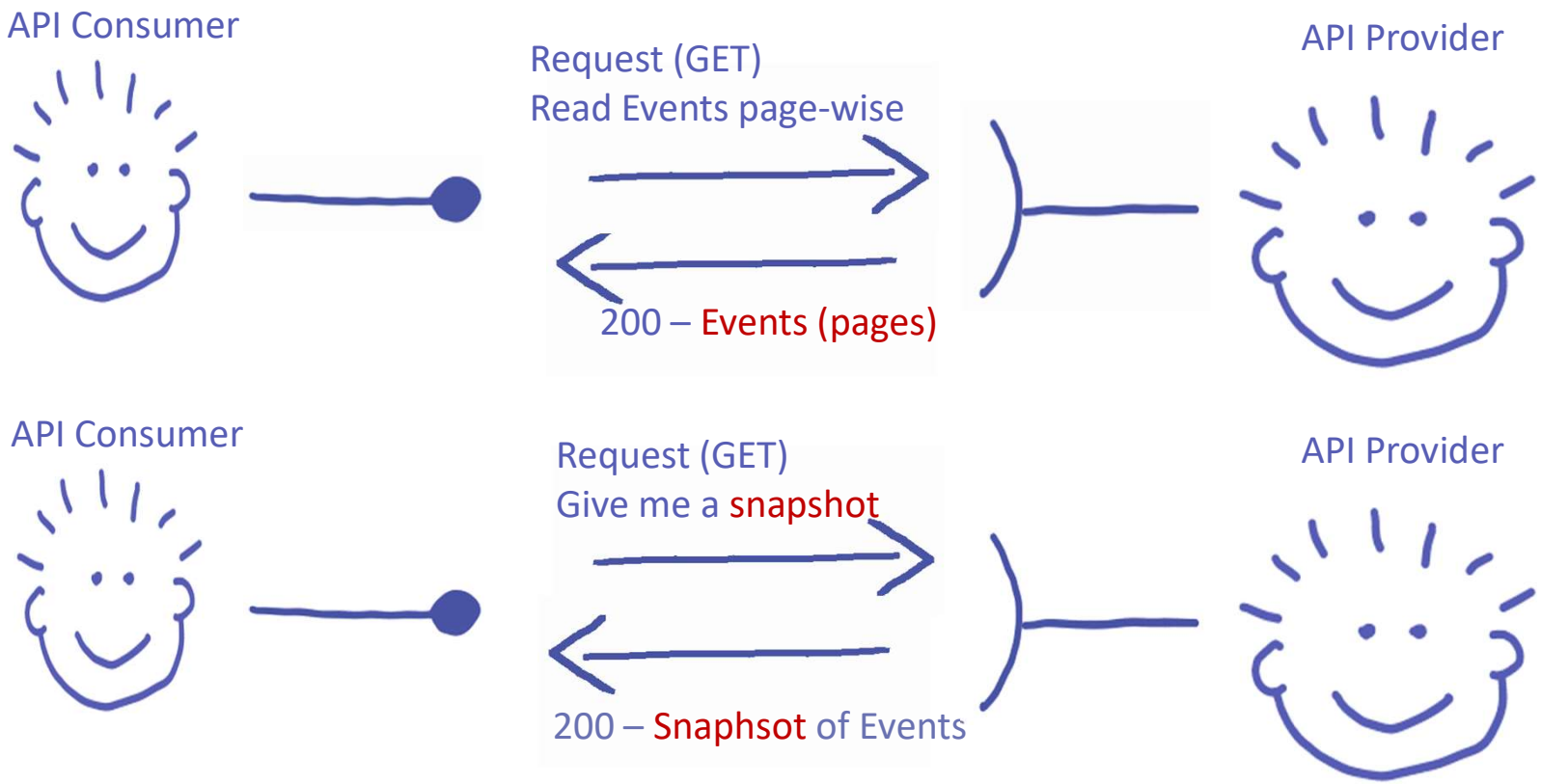
Realization:

- Queuing System based

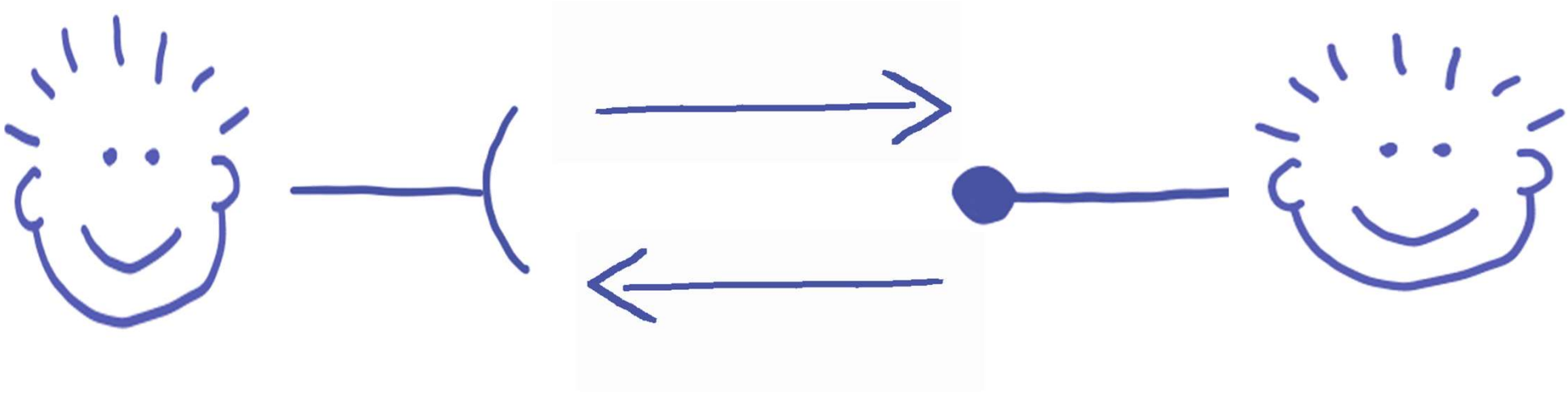
Event propagation: HTTP Web Feeds

Realization:

- HTTP Web Feeds
- Event Snapshots



Simple synchronous API – so everthing easy ☺

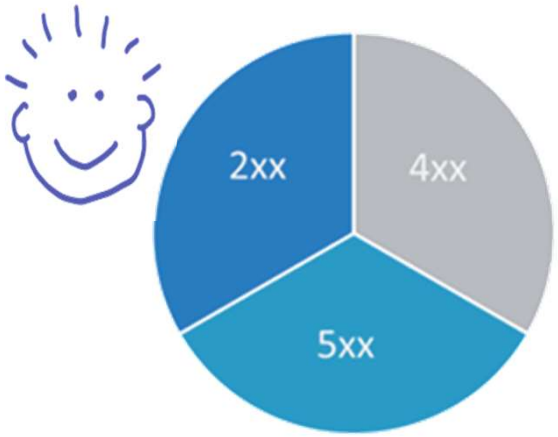


Really

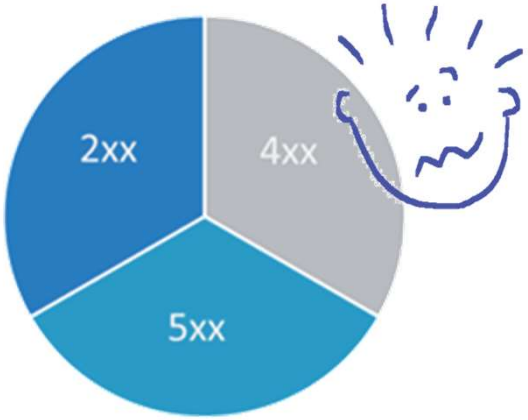
Let´s challenge it a little bit with a winking eye…

Start with the obvious things: HTTP response codes – as HTTP response codes are not rocket science

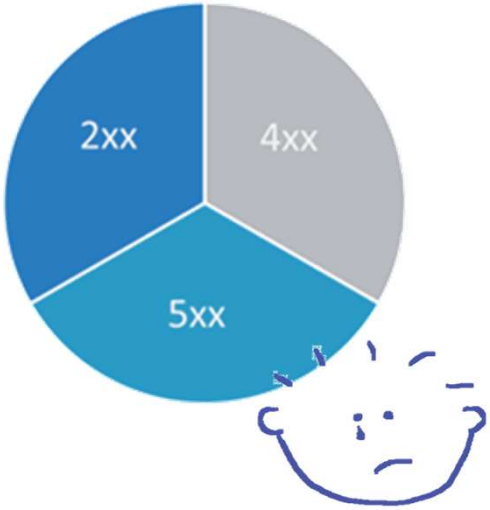
[Author's note: in detail sometimes it is in fact 😉]



200 (OK)	NO, this is not a possible response code in case of error (and you just send error details in the response body)!
202 (ACCEPTED)	OK… And now? How do I get an update / ack when you are finished?
204 (NO CONTENT)	It's OK, e.g. if I have sent a DELETE - but then please really without content (response body).

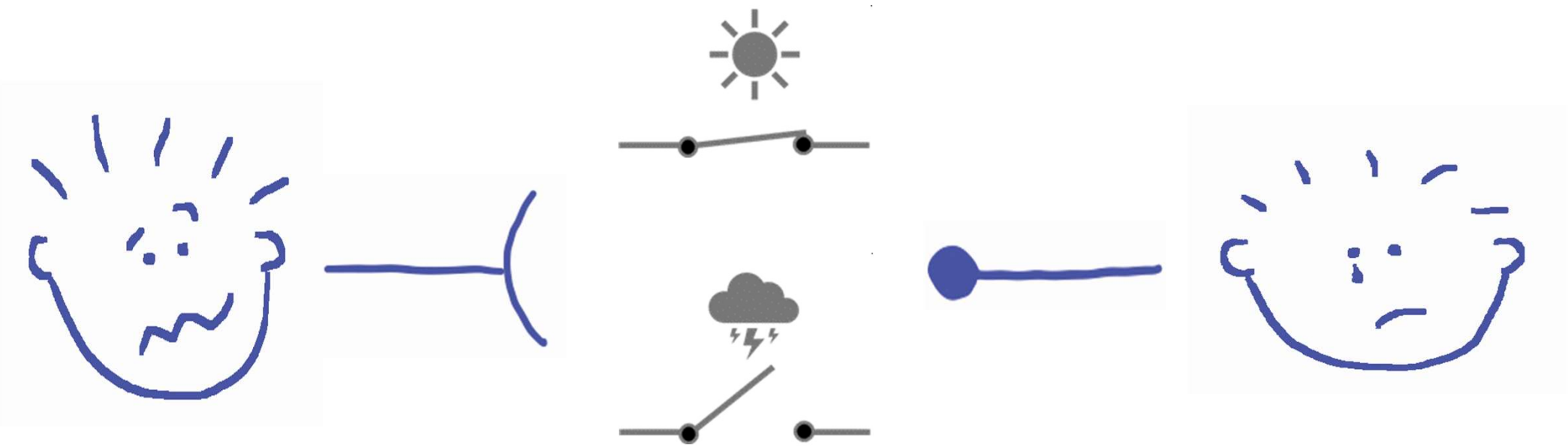


400 (BAD REQUEST)	So I have sent a request that does not match the interface description.
401 (NOT AUTHORIZED)	So I really have to check if I send the wrong API key or an expired access token.
403 (FORBIDDEN)	OK, I tried to do something I'm not supposed to do. But what?
418 (I´M A TEAPOT)	Ha-ha. Can you really afford jokes like that?



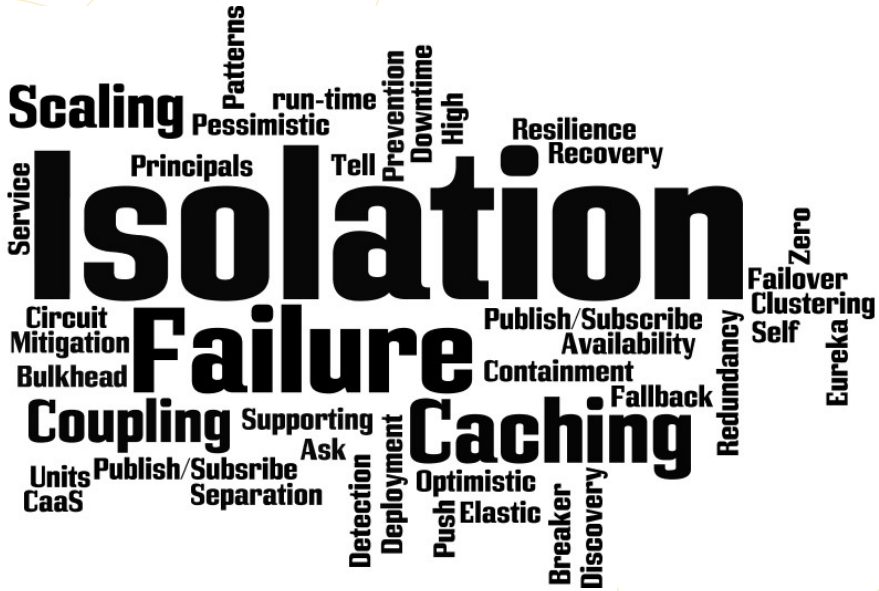
500 (INTERNAL SERVER ERROR)	Got it, you don't even know why it's not working right now.
503 (SERVICE UNAVAILABLE)	This indicates to me that you may not be able to answer due to overload or maintenance -that's a statement. I'll try again later.

… did I cause your 5xx ?



But how can I increase my overall API availability when a synchronous coupling is necessary ?

Source: http://smoobu.com/



Availability

$$\text{Availability} = \frac{\text{MTBF}}{\text{MTBF} + \text{MTTR}}$$

Availability calculation sample

$$\frac{720\text{h (30d)}}{720\text{h (30d)} + 2\text{h}} = 99{,}7\ \%$$

MTBF … Mean Time Between Failure → Uptime of a system
MTTR = Mean Time To Recovery → Downtime of a system

Resilience - Reliability

$$\text{Availability} = \frac{\text{MTBF}}{\text{MTBF} + \text{MTTR}}$$

Increase the *Uptime*:

- Failover-Clustering,
- Scalability,
- Zero Downtime Deployment,
- 2nd Site,
- ….

Traditional Approach

MTBF … Mean Time Between Failure → Uptime of a system
MTTR = Mean Time To Recovery → Downtime of a system

Resilience - Reliability

$$\text{Availability} = \frac{\text{MTBF}}{\text{MTBF} + \text{MTTR}}$$

Increase the *Uptime*:

- Failover-Clustering,
- Scalability,
- Zero Downtime Deployment,
- 2nd Site,
- ....

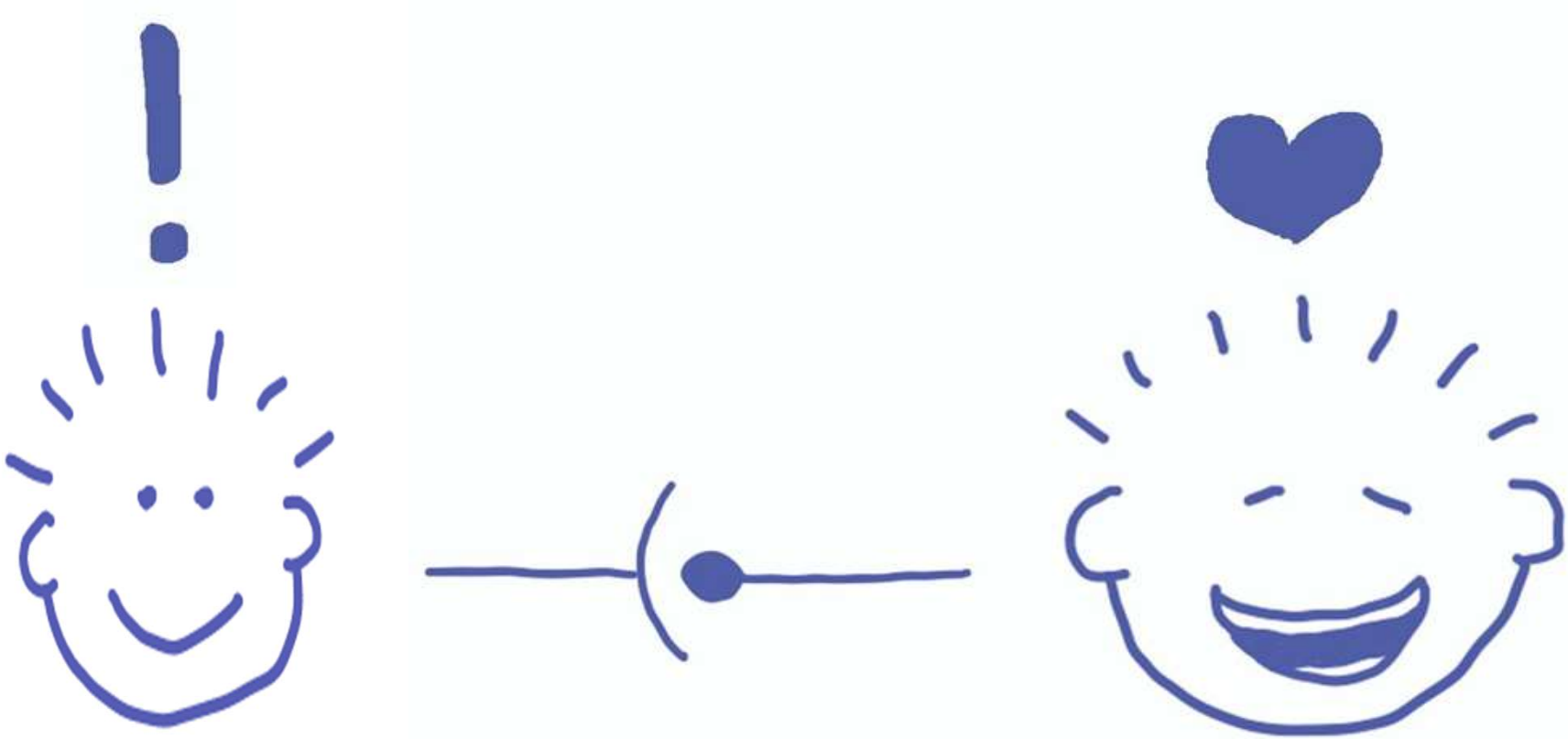
Traditional Approach

MTBF ... Mean Time Between Failure → Uptime of a system
MTTR = Mean Time To Recovery → Downtime of a system

Minimize the *Downtime* by designing stability principles in your architecture:

Failure Prevention • Separation of failure units • Prevent cascading failures by design (Event Sync, Async. Backend Calls)	**Failure Detection** • TimeOut Handling • Monitoring (Building Blocks, Connections)
Failure Mitigation • Fail-fast • Circuit Breaker (Fallback-Default value) • Shed Load	**Recovery** • Retry

Resilience Approach



Takeaways

- ❑ APIs open your digital products, they are **NOT** just an interface between different systems
- ❑ API documentation at the right place is important
- ❑ Think about the integration pattern (synchronous, asynchronous, event synchronization) based on your requirements
- ❑ Think about resilience when you have to choose synchronous integration.