

TOP REST API DESIGN PITFALLS

BY VICTOR RENTEA





Victor Rentea in 

► **18y of Coding, Java Champion** 

► **10y of Training at 150 companies** **for hire**

► **100+ Talks on YouTube**

► **Free Webinars for my Community**

► **Life +=**  +  +  

REST API



Postel's Law

(author of TCP/IP)

Be **conservative** in what you do,
but liberal in what you accept from others

Backwards Compatible

Exercise

Cause a **Breaking Change**

that would force clients to update

Content-Type: application/xml
or application/json; charset=utf-32

fullName: a request:

"firstName": "John"

"lastName": "DOE"

"email": "Address.com",

"phone": ["+407129"],

"currency": "EUR"

One of
EUR | USD | GBP

response: 2023-03-01

"date": "03/01/2023",

"age": "37" ← number

~~"country": "ROU"~~

"more": ...

One of
BEL | ROU | FRA

fullName:

a request:

+required

+regex valid, min length

make it an array 🐱

2023-03-01T00:00:00Z

One of

EUR | USD | GBP

no longer
supported

← number

One of

BEL | ROU | FRA

Legend:

breaking change=>MAJOR++ (v2.0)

non-breaking (backwards-compatible)=>MINOR++ (v1.1)

Backwards-Compatible Forever!

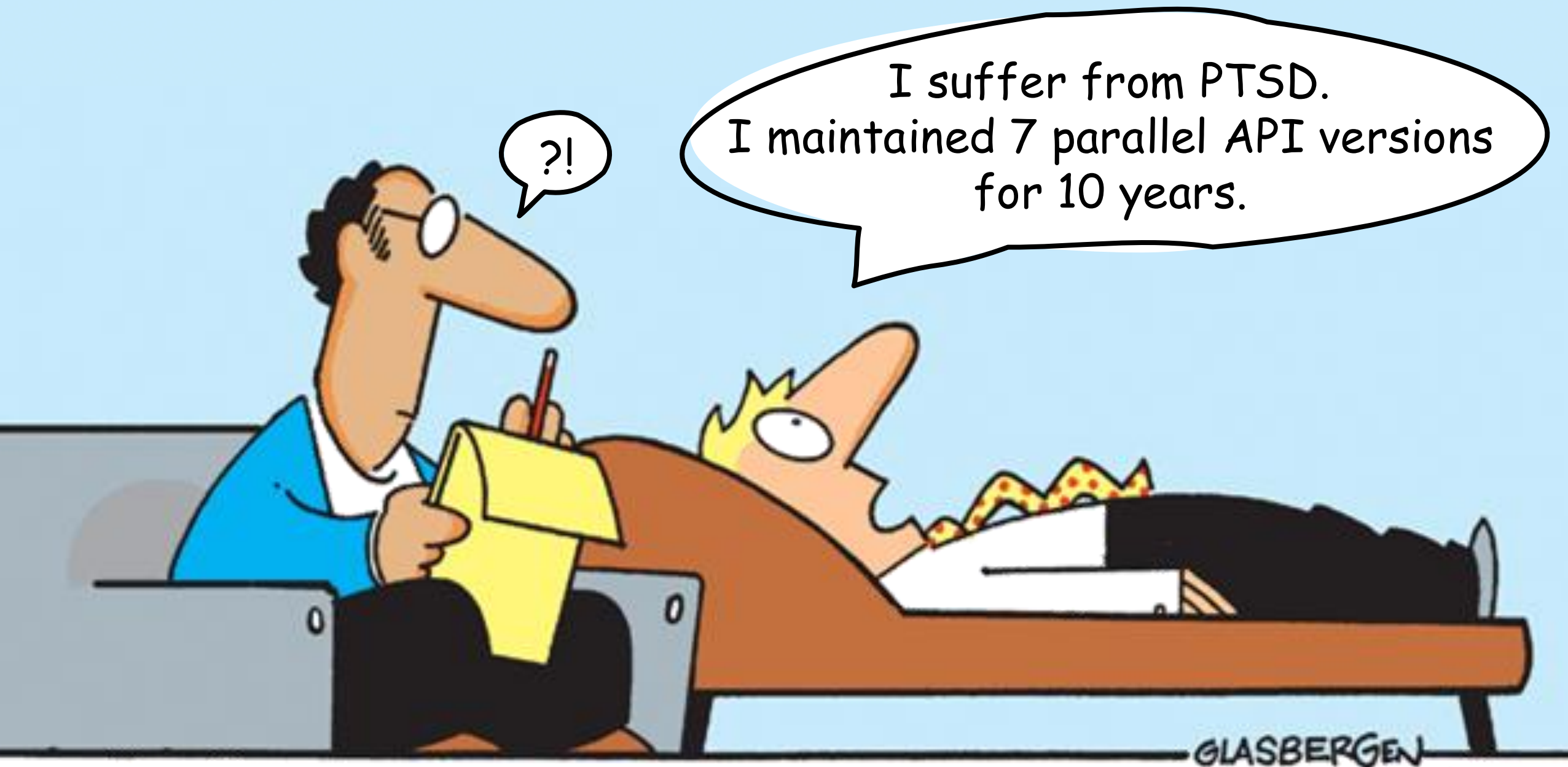
- We keep adding non-breaking changes:
 - + support new input fields and values
 - + return more output fields
- + return / accept alternative representations:

```
{  phoneString: "+40720..",  
  phoneAreaCode: "+40",  
  phoneLocal: "720.." }
```
- When accumulated **API Debt** grows to high → v2



V2 is Here! 🤪

- Ideal: have max 2 parallel versions for a limited timeframe
- Logic gets polluted with copy-paste and if (version==7)
- **Find your clients** via Auth-tokens or TracelD💖
 - To trace field-level coupling, see pact.io
- **Convince Clients to Upgrade™**
 - V1: No New Features, Rate limited, 📺 Fee\$, 📞, 🙏, 🤡, 🗨️, 🤖 Can I help?
 - ~~if (random<0.01) throw("Surprise💣: Please upgrade to v2"!)~~



To avoid future breaking changes,
I'll make my API "more generic"



Future-Proof API

```
{  
  emails: ["a@b.com"],
```

in case tomorrow we start supporting more 😊

```
  phones: [{phone: "ABC", type: "work"}],
```

more types tomorrow?

```
  metadata: [  
    {name: "age", value: 12},  
    {name: "gender", value: "M"}  
  ]
```



more keys tomorrow

```
  ... 100 more added over 7 years 🤯
```

To avoid future breaking changes,
I'll make my API "more generic"

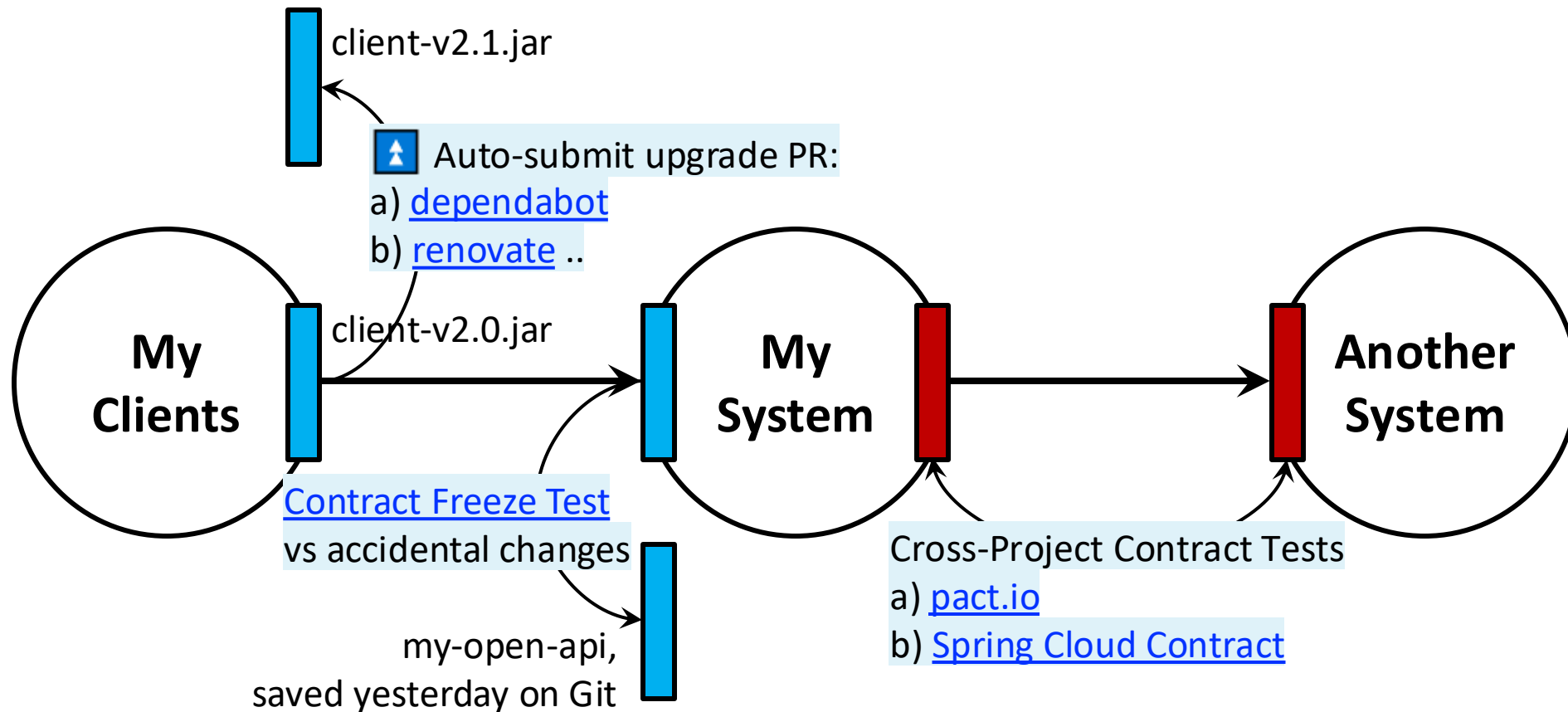
Hurts API Clarity 🤯

Versioning Strategies

- **new service**: order-service-**v2**.intra + separate Git+DB?
- **per-service** : order-service.intra/**v2**/order/{id}
 - "v1" is a middle finger  to your API customers

[Roy Fielding, creator of REST](#)
- **per-endpoint**: /order/**v2**/id --- is this a monolith?
- **Content-Type + Accept**: application/json+v3

How **fast** can you detect a **contract mismatch**?



Performance Hits

Get One of Many

You have many records in your DB.
The only way you expose them is via:

`GET /products/{id}`

 Misplaced Responsibility?

What can go wrong?

Clients could **network-call-in-a-loop** (=performance massacre 💀):

`for (id in listOfIds) {... yourApi.getById(id)}`

Expose a batch API



sequential IDs ?

■ `GET /products?id=1,2,3,...`

■ **POST** `/products/get-many` + `[1,2...1000]`

■ `GET /products` + `[1,2,3..]`

URL can get truncated at 2000 characters

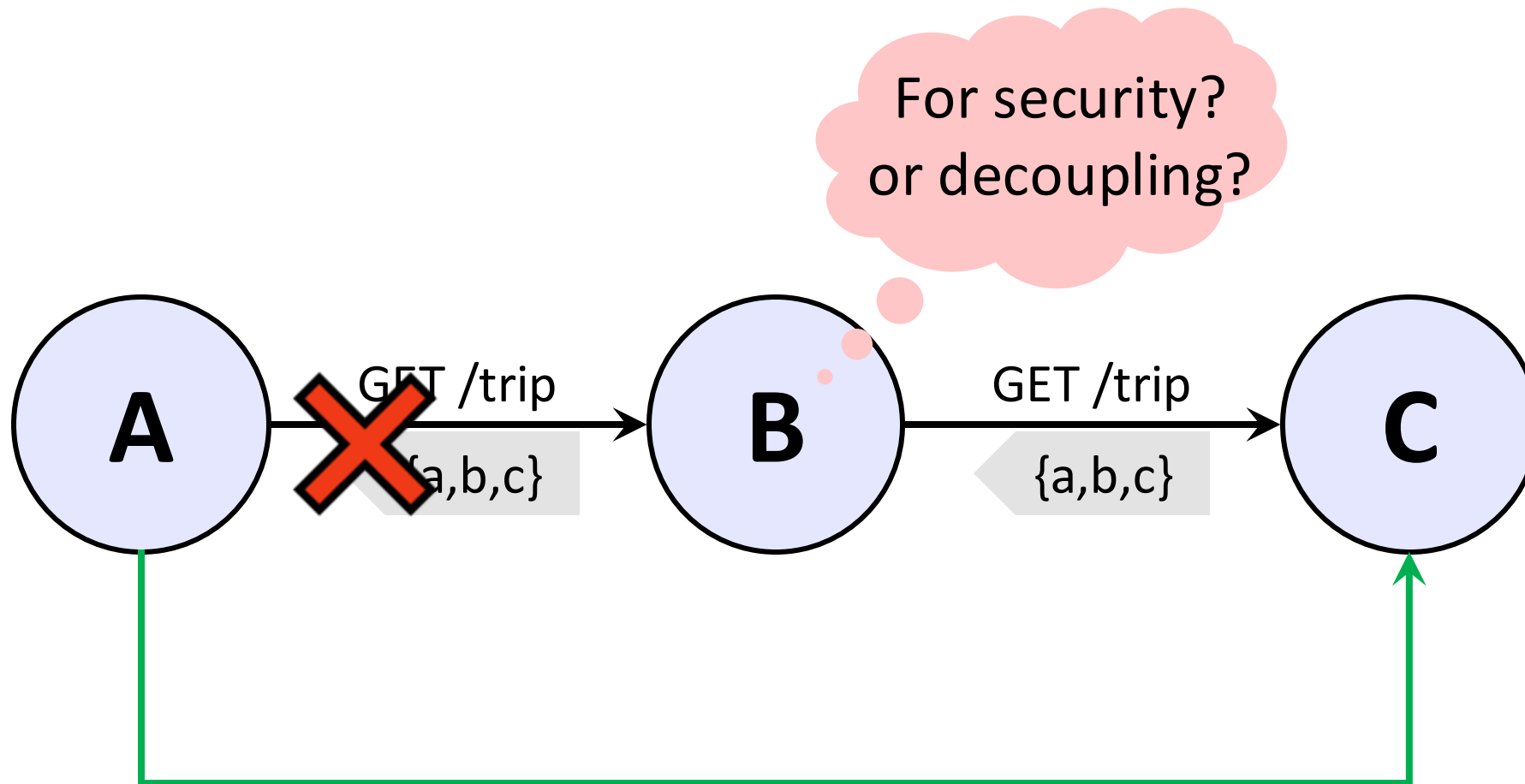
Keep response decent, size <= 1000

GET can have a body since 2021 – too recent?

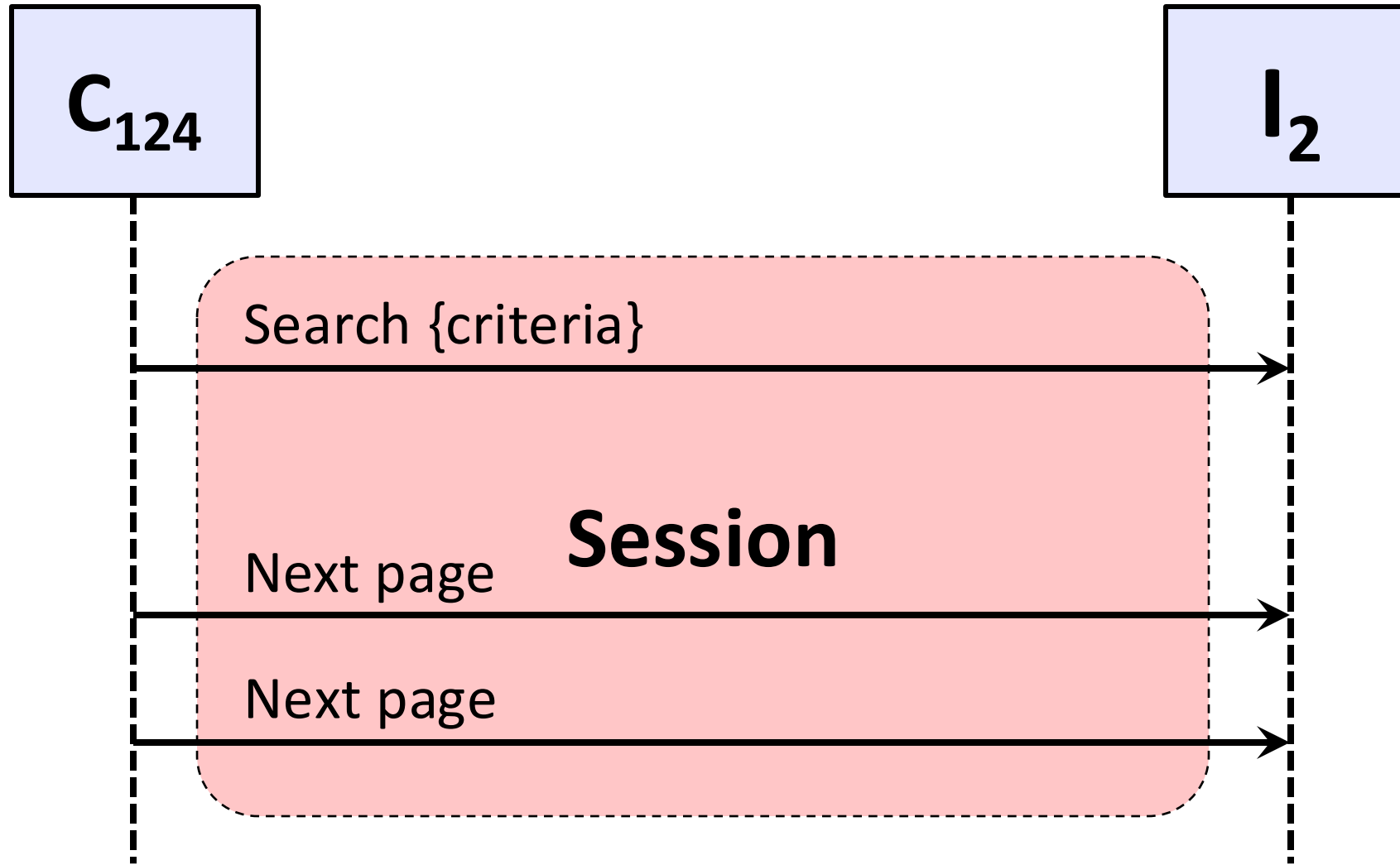
■  Rate-limit client: 429 Too Many Requests 👉 Please use the batch API

KISS: implement the bulk API when alarms 🚨 in production metrics show it's needed.

Request Proxying



Stateful Endpoints



API design that can Hurt Performance

Network-call-in-a-loop

Large Monolithical Payloads

Request Proxy (accidental)

Stateful Endpoints

ORM Lazy Loading (goof 😊)



Performance

Logic

Scalability

Don't Leak Domain Model in your API

Opinions?

```
@GetMapping( )  
public Customer findById(  
    CustomerDto
```

```
// Domain Model  
public class Customer {
```

- **Your Domain Model Freezes** as clients grow coupled to your API
- **Security Risk** to expose sensitive data [[API-SEC](#) 🗝️]
- **Pollutes Domain** with presentation: @JsonIgnore, @JsonFormat...
- **Performance Risk** if ORM lazy-loads extra data (goof!)

Expose **Data Transfer Objects (DTOs)** in your API

clear

documented

Stable API for clients to rely on

Separate **Contract** (DTO)

from **Implementation** (Domain Model)

Evolving to simplify implementation

Data Transfer Objects **←Mapping→** Domain Model

= the price to pay for decoupling the two models

Automatic Mappers

```
Dto  
  
var dto = new Dto();  
dto.setName(entity.getName());  
dto.setEmail(entity.getEmail());  
..... <20 more>  
return dto;
```

```
}
```

Boilerplate Code

```
var dto = Dto.builder()  
    .name(entity.getName())  
    .email(entity.getEmail())  
    ..  
    .build(); // immutable DTO
```

```
val dto = Dto(//named params  
    name = entity.name,  
    email = entity.email,  
    ...  
);
```



Automatic Mappers

<< *My API Model* >>

```
class CustomerDto {  
    firstName:string  
    lastName:string  
    ...  
}
```

<< *Domain Model* >>

```
class Customer {  
    firstName:string  
    lastName:string  
    ...  
}
```



mapping happens automatically
if field names match

Libraries:

- MapStruct 🏰
- ModelMapper
- Dozer
- orika
- AutoMapper

Best: it generates mapping code:
1) can see/debug generated source
2) fastest (JIT-compiled)
3) simple to give up on it

The Dark Side of Automatic Mappers



**Frozen
for Clients**

<< My API Model >>

```
class CustomerDto {  
  firstName:string  
  lastName:string  
  ...  
}
```

<< Domain Model >>

```
class Customer {  
  firstName:string  
  1 fullName:FullName  
  ...  
}
```

I'll group them in
a new Value Object

**Ever
evolving**

- Build/Tests should fail 🙅
- Complex Mapping Customization...
- Tempting to keep the models in sync - Why then DTOs?!

NEAH...

I'll do it next sprint

The Fallacy of Auto-Mappers: Tempting to keep the models in sync.

When Mapping grows cumbersome => Switch to manual

Abusing the same DTO for GET + POST/PUT

```
@GetMapping("/{id}")  
InventoryItemDto get(id) {
```

```
@PostMapping  
void create(InventoryItemDto) {
```

InventoryItemDto

```
{  
  "id": null, ← always null in create flow  
  "name": "Chair",  
  "supplierName": null,  
  "supplierId": 78,  
  "description": "Soft friend",  
  "stock": 10,  
  "status": null,  
  "deactivationReason": null,  
  "creationDate": null,  
  "createdBy": null  
}
```

What's wrong?

The Contract becomes:

– misleading for **clients**



"Why should I provide the id?"

– confusing to **implement**

"Why is that field always null?"

– **coupling** of endpoints

get change ==> create changes

Dedicated Request/Response Structures

= **CQRS** at the API Level

```
@GetMapping("/{id}")
GetItemResponse get(id) {
```

GetItemResponse

```
{
  "id": 13,
  "name": "Chair",
  "supplierName": "ACME",
  "supplierId": 78,
  "description": "Soft friend",
  "stock": 10,
  "status": "ACTIVE",
  "deactivationReason": null,
  "creationDate": "2021-10-01",
  "createdBy": "Wonder Woman"
}
```

```
@PostMapping
void create(CreateItemRequest){
```

CreateItemRequest

```
{
  "name": "Chair",
  "supplierId": 78,
  "description": "Soft friend",
  "stock": 10
}
```

+validations
@NotNull..

in a 'dto' package

A shared 'dto' package could encourage
reusing DTOs between endpoints = BAD PRACTICE

Keep request/response objects next to Use-Cases
and separated from each other = VSA

CQRS

≠ SQL

Command/Query Responsibility Segregation

Update Data

Read Data

= Separation

Command/Query Responsibility Segregation

Most people perceive software systems as *stores of records*: that they **Create**, **Read**, **Update**, **Delete** and **Search** (CRUDS)



As system grows **complex**:

READ aggregates (SUM..) or enriches the data (JOIN, API calls..)



latency & availability

WRITE store additional metadata: createdBy=, ...



data consistency

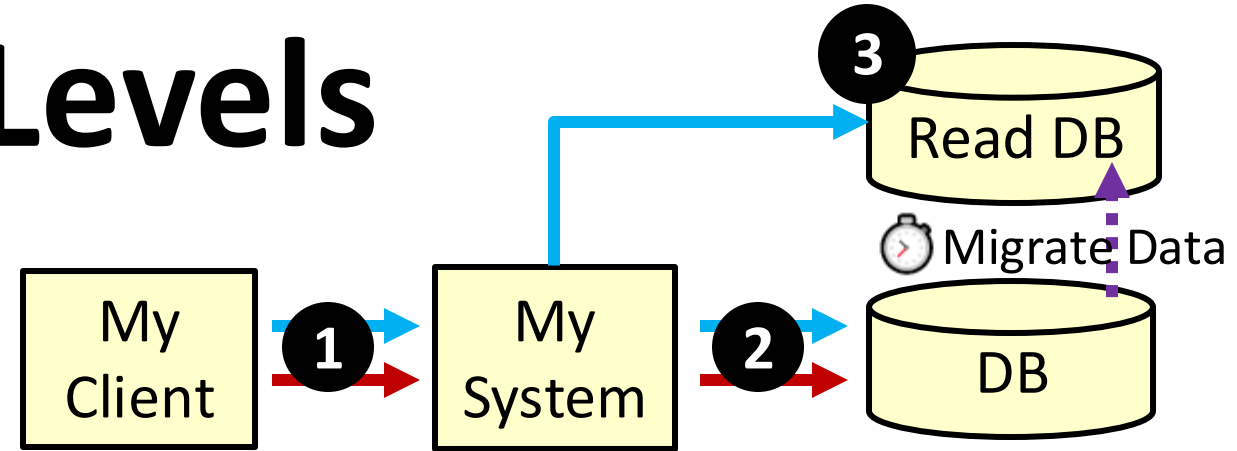
CQRS = use separate **WRITE** / **READ** data models

CQRS Levels

1. CQRS at API level to Clarify Contract

GetItemResponse -- query

CreateItemRequest -- command



2. CQRS at SQL Interaction to Optimize Read when using an ORM

Search via `SELECT new dto.SearchResult(e.id, e.name...) FROM User e` -- query

Modify data via Domain Model Entities -- command

 **Strong Consistency**

3. Distributed CQRS

Write to one storage: SQL, Event Store ..., but

Query a Read Projection in another storage, async updated 

- **Materialized Views** (SQL age) can pre-aggregated data
- **Redis**,... < 1ms, in-memory
- **Elastic Search**... for full-text search
- **Mongo**... for dynamic structure

 **Eventual** 

Consistency

low latency +
high availability,
under high load

CQRS

Separate **Commands** (Updates)

from **Queries** (Reads)

Generated ID is OK

Should a POST or PUT -- Command
return data back? -- Query

NO!*

- Couples GET and PUT responses.
- Can lead to security breaches.
- Can waste resources.

Please approach me for Debates>



What can
go wrong?

A Large Edit Screen

Edit Inventory Item id:13

Name	<input type="text" value="Chair"/>
Description	<input type="text" value="Soft Friend"/>
Supplier	<input type="text" value="ACME"/> ▼
Supplier Cost (EUR)	<input type="text" value="120"/>
Stock	<input type="text" value="10"/>
Status	<input type="text" value="ACTIVE"/> ▼
Deactivation Reason	<input type="text"/>



PUT

A Large Edit Screen

Server must DIFF new state vs DB

Edit Inventory Item id:13

Name	Chair
Description	Soft Friend <input checked="" type="radio"/>
Supplier	ACME ▾
Supplier Cost (EUR)	120
Stock	10 <input checked="" type="radio"/>
Status	ACTIVE ▾
Deactivation Reason	

v=7

If you change status ACTIVE→INACTIVE,
user must provide a reason

BAD UX: not obvious rule

CONCURRENT UPDATES

User edits the description in but meanwhile minutes..

ItemSoldEvent arrived that decreased the stock

This can cause **data loss** by blind overwriting stock=10.

Solutions:

A) **optimistic locking:** **frustrating users** on click Update:

UNABLE TO SAVE

Someone else already changed this item.
Refresh the page and re-do your updates.

OK 

👉 +VERSION column in DB sent/received from clients

B) **pessimistic locking** on click  Edit: **bottleneck risk**

CANNOT OPEN EDIT SCREEN

Item under edit by **vrentea** since 3h ago.

OK 

👉 +LOCKED_BY, LOCKED_AT columns in DB +

watchdog 

Task-Based UI

"Prefer Action Buttons over Edit Screens"

What tasks
do my users 🥰
usually do?

Edit Inventory Item id:13

Name	Chair
Description	Soft Friend
Supplier	ACME ▾
Supplier Cost (EUR)	120
Stock	10
Status	ACTIVE ▾
Deactivation Reason	

Cancel

Update

Edit some text for more
FOMO

Adjust price & supplier

Sell over the phone

Re/Deactivate a product

Name ^	Supplier	Active	Supplier Cost	Stock
Chair	ACME	<input checked="" type="checkbox"/>	120	10
Armchair	ACME	<input type="checkbox"/>	160	12
Table	ACME	<input checked="" type="checkbox"/>	255	5
Sofa	ACME	<input checked="" type="checkbox"/>	980	4

Task-Based UI

- ✓ Intentional API
- ✓ Lower concurrency risk
- ✓ Simpler server implem.

- ✗ Requires User Research
- ✗ More APIs
- ✗ More Screens

sub-resource ✓

PUT /item/13/details

```
{
  name:
  supplier:
  description: 🦄
}
```

or via

PUT/PATCH /item/13

Deactivate Inventory Item

Reason*:

stopped manufacturing|

Cancel Deactivate

PUT /item/13/deactivate

```
{ reason: .. }
```

action ✓

Name	Supplier	Active	Supplier Cost	Stock
Chair	ACME	<input checked="" type="checkbox"/>	120	10
Armchair	ACME	<input type="checkbox"/>	160	12
Table	ACME	<input checked="" type="checkbox"/>	255	5

action ✓

Update Supplier Cost

New cost*:

120

Cancel OK

PUT /item/13/cost

```
{ newCost:120 }
```

sub-resource ✓

POST /item/13/sell

```
{ quantity:2 }
```

POST is not idempotent (not retryable)

Religious REST Fallacy

URLs shall never contain verbs (actions)

```
PUT /item/13/deactivate  
{ reason:"..." }
```

action 

Religious REST Fallacy

- CRUD can limit your API's semantics
- PUT endpoint attracts most complexity 🤯

In complex domains, introduce:

- **Sub-resources**: GET & PUT /item/13/cost
- **Actions (verbs)**: PUT or POST /item/13/deactivate

**Segregate unrelated client actions
in different screens/endpoints**



PATCH

PATCH

PATCH /item/1 = partial update

```
{
  "status": "INACTIVE",           => set
  "deactivationReason": "reason", => set
  "description": null             => remove
  // missing fields               => unchanged
}
```

=> hard to parse

Manual

Why did status, deactivationReason and description change at once??

[jsonpatch.com

```
{op:"set", path:"/status", value:"INACTIVE"},
{op:"set", path:"/deactivationReason", value:"reason"},
{op:"rem", path:"/description"}
{op:"set", "path":"/authors[id='jdoe']/phone"} ❌
]
```

PATCH

Lacks Semantics

Why did **status**,
deactivationReason and
description change at once??



```
PUT /item/1/deactivate  
{reason}
```

500

Internal Server Error

{...}

Security Breach 🚨:

Hackers can determine frameworks versions and exploit their known CVE vulnerabilities

500 Internal Server Error

```
java.lang.IllegalArgumentException: Missing email
at victor.training....CustomerApplicationService.register(CustomerApplicationService.java:77)
at java.base/java.lang.reflect.Method.invoke(Method.java:569)
at org.springframework.transaction.interceptor.TransactionInterceptor.invoke(TransactionInterceptor.java:119)
at org.springframework.web.filter.RequestContextFilter.doFilterInternal(RequestContextFilter.java:100)
at org.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run(TaskThread.java:61)
at java.base/java.lang.Thread.run(Thread.java:840)
```

400 Bad Request / 422 Unprocessable

errorRef:<UUID>

{}

Missing 'email'

{email}

Missing 'phone'

{email,phone}

Missing 'age' x 10 more times



Missing ['phone','email','age'..]

[RFC 9457](#) = standard error response schema

RFC 9457 = standard error response schema

```
HTTP/1.1 422 Unprocessable Content
Content-Type: application/problem+json
Content-Language: en
```

```
{
  "type": "https://example.net/validation-error",
  "title": "Your request is not valid.",
  "errors": [
    {
      "detail": "must be a positive integer",
      "pointer": "#/age"
    },
    {
      "detail": "must be 'green', 'red' or 'blue'",
      "pointer": "#/profile/color"
    }
  ]
}
```

REST API Design Pitfalls

- ✓ **Backwards Compatibility?** What's that!?
- ✓ **Expose your internal Domain Model.** GDPR if for legal dept.
- ✓ **Encourage your clients to call your GET {id} in a loop**
- ✓ **Proxy as many requests you can**
- ✓ **Reuse the same DTO class in POST/PUT/GET!**
- ✓ **All PUT must return data back to client**
- ✓ **CRUD is all you need! Ban any verbs from your URLs!**
- ✓ **One PATCH to rule them all!**
- ✓ **Errors don't happen! YOLO**

Thank You!

DEVOXX™
... UK ...

victorrentea.ro



It all depends

Almost everything I said has exceptions.
I flew here to debate them with you.
Please approach me.

