Lessons Learnt Building REST APIs

CD207 @ SAP TechEd 2013

Sascha Wenninger | **@sufw**
+
Custodio de Oliveira | **@zcust01**

About Us

Sascha Wenninger

- Technical Architect
- Focus: SAP Integration
- Co-founder of **Blue T**
- Wannabe Performance Engineer
- Enterprise IT != Boring

Custodio de Oliveira

Great ABAP Engineer

Focus: WDA, ABAP OO, BRF+, clean code

How We Got Here

Got curious about REST a few years ago.

Read some **books**/**blogs** and watched some **conference talks**

Designed some ~~simple~~ trivial ~~APIs~~ HTTP Handlers in ERP


BOSS, WE'VE JUST FINISHED THE WEB SERVICE
NO, IT'S JUST A WEB SERVICE
GREAT!! DO WE NOW HAVE A SOA?
OK! OK! WE'LL CALL "HIM" SOA
geek and poke
HOW TO GET A SOA

A New Project!

Customer Self-Service

Attributes

Seamless fit with existing online presence

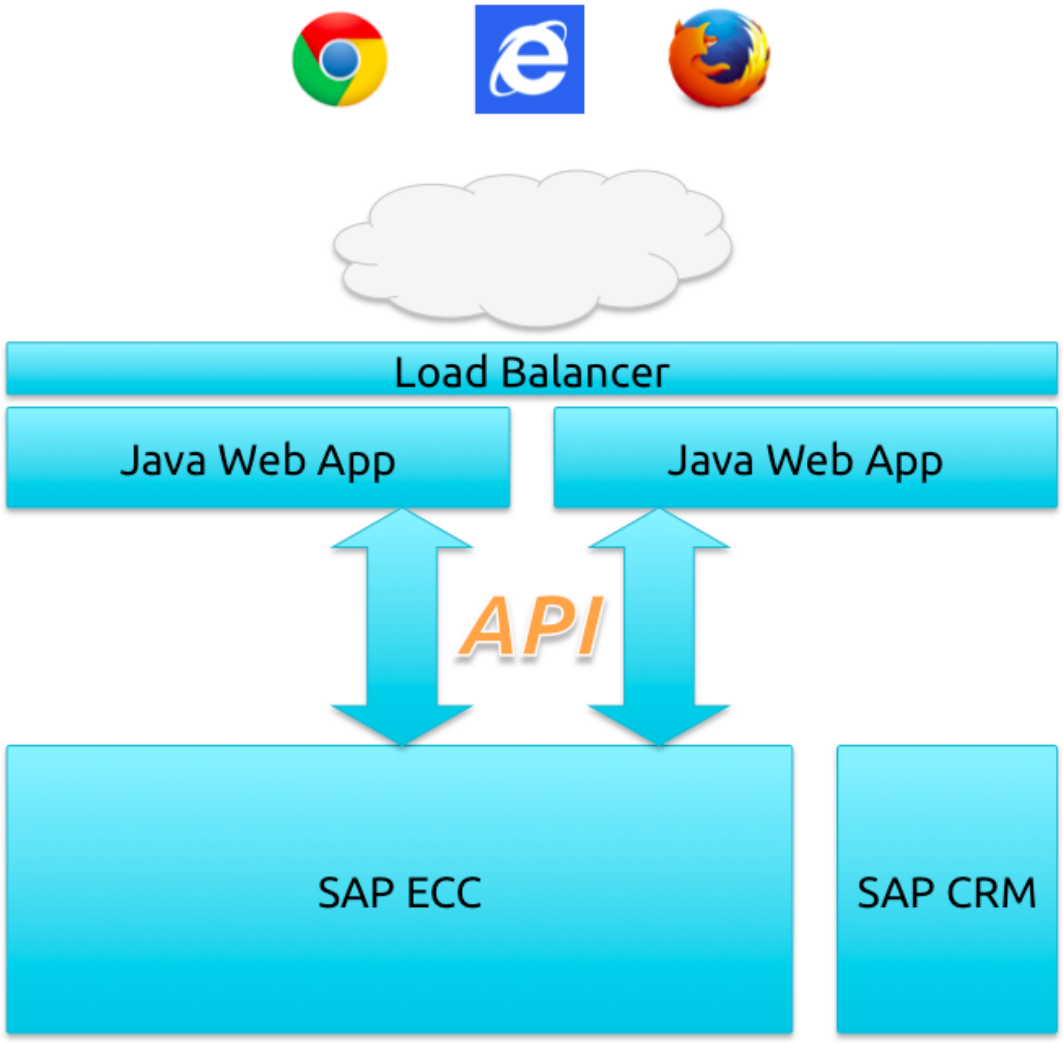
Easy to use for ~80,000 business customers

Integration to ERP and CRM for data & functions

Initially considered SAP's Biller Direct

Ultimately decided on a custom build

Stateless Java front-end built on Spring


Load Balancer
Java Web App
Java Web App
API
SAP ECC
SAP CRM

Welcome Cheryl APCN: 1000383785 | My details | Log out

My Business Account

- Dashboard
- Tax invoices
- Payments
- Transactions
- Account balance
- Enquiries
- Your account
- FAQs
- Help

Your account balance

VIEW ALL

Your balance as of 9 Mar 2012

$ 13,387.73*

*Payments processed today will appear the following business day.

12.5%

Credit limit:	$ 4,200.00
Exposure:	$ 525.00
Utilisation:	12.5%
Valid to:	31 Dec 2012
Pay terms:	Monthly

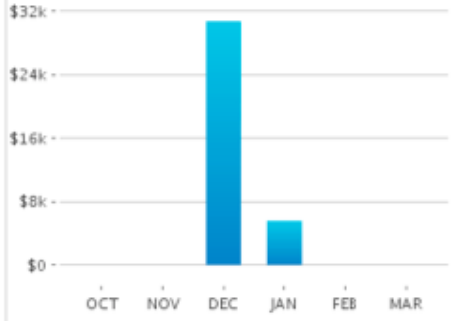
Account balance history

Unpaid tax invoices (0)

VIEW ALL

You have no outstanding tax invoices

Tax invoice history



"Why don't we build a **REST**ful API?"

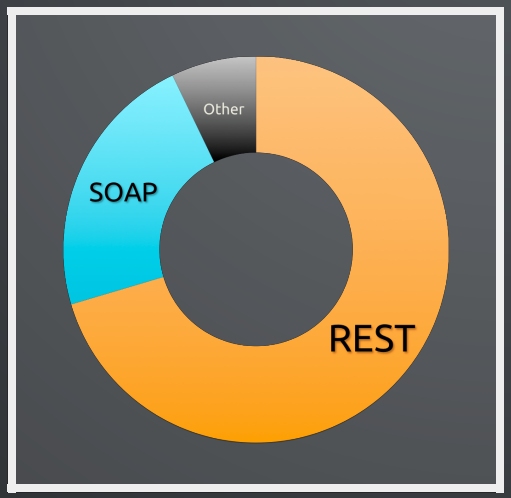
Why REST?

Cacheable
Some queries are expensive
We may have a lot of them

Not XML
Can render PDF or XLS in the backend
Keeps Java app lean

HATEOAS
Hyperlinks instead of hard-coded flow

Popular

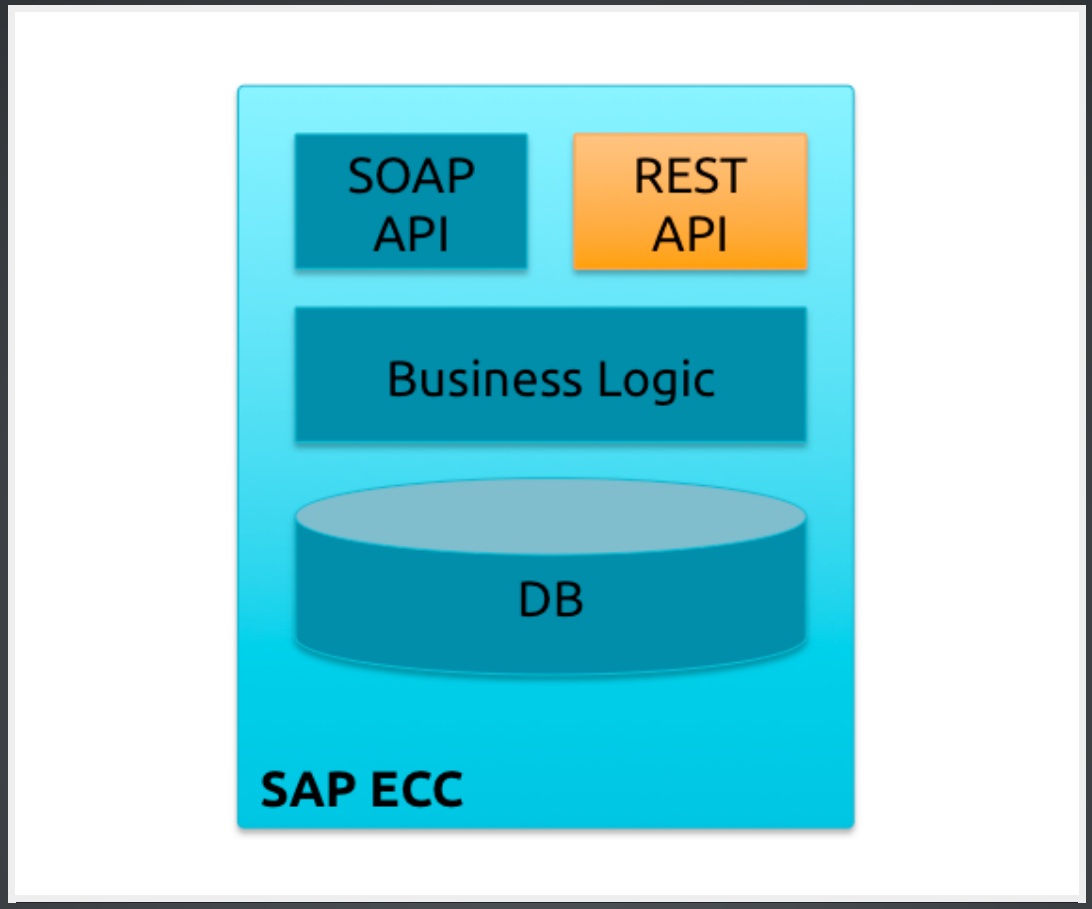


What did we Learn?

#0: Ease into it.

Nobody on the team had built a full RESTful API before.

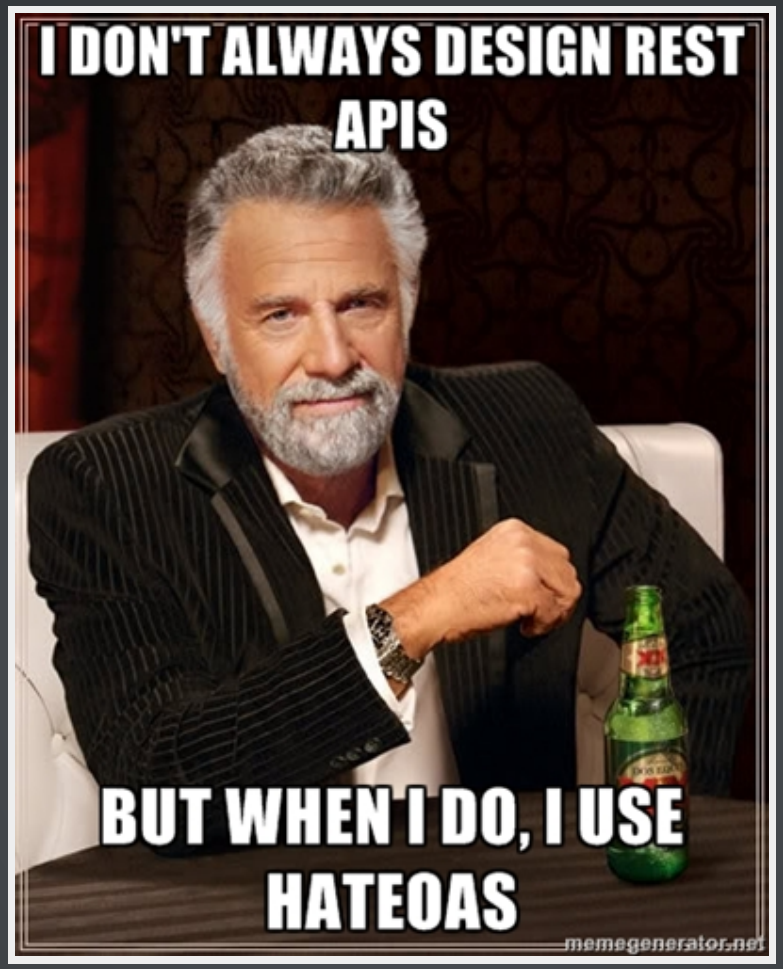
To sell the idea, we built a SOAP API in parallel.

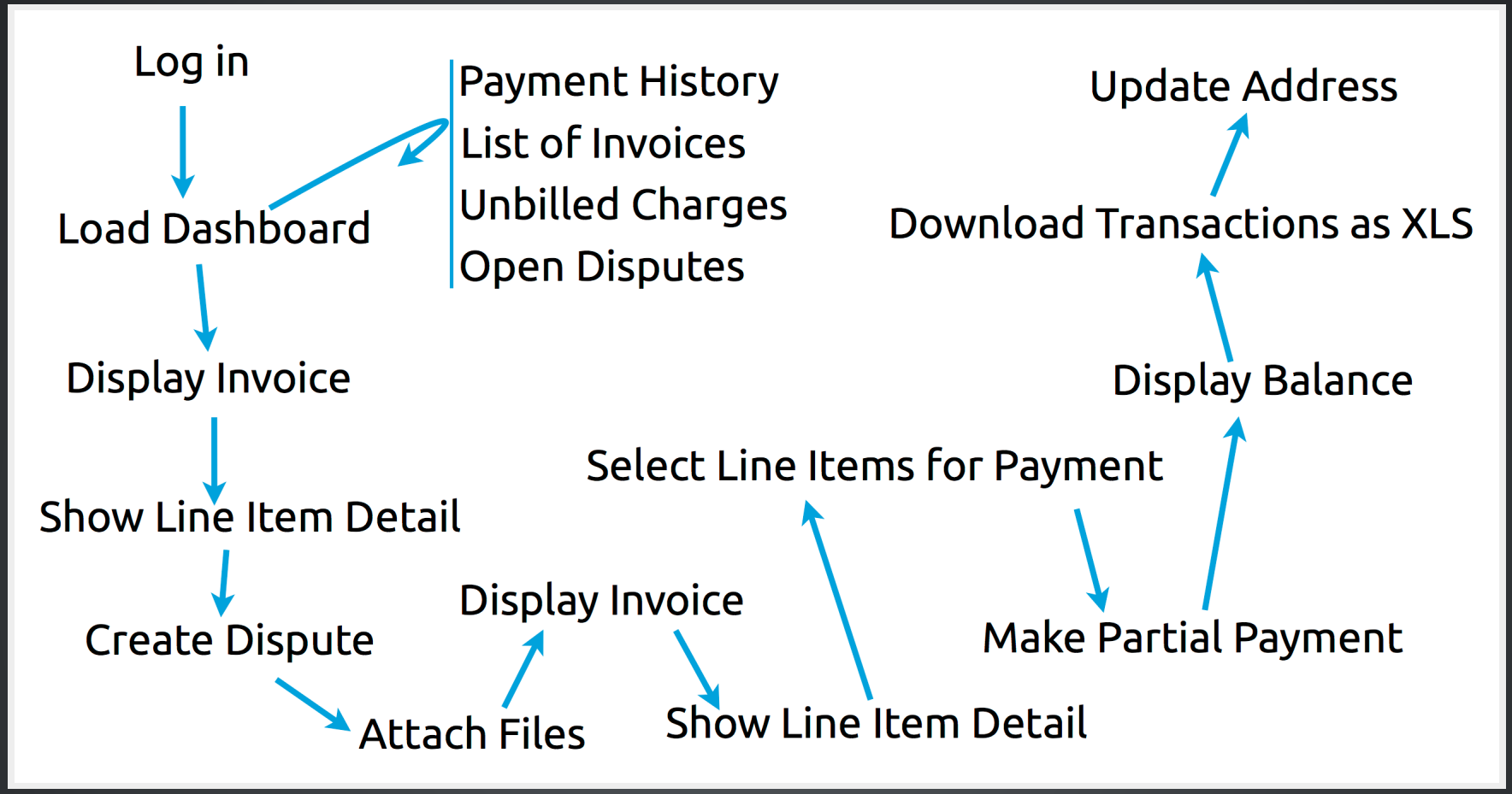

SOAP
API
REST
API
Business Logic
DB
SAP ECC

Abandoned this after ~4 weeks once comfort was built.

#1: HATEOAS

Hypertext As The Engine Of Application State


I DON'T ALWAYS DESIGN REST APIS
BUT WHEN I DO, I USE HATEOAS
memegenerator.net


Log in
Load Dashboard
Payment History
List of Invoices
Unbilled Charges
Open Disputes
Display Invoice
Show Line Item Detail
Create Dispute
Attach Files
Display Invoice
Select Line Items for Payment
Show Line Item Detail
Make Partial Payment
Display Balance
Download Transactions as XLS
Update Address

HATEOAS

Makes development and maintenance easier in the long run

- Server (ECC) is in control of logical flow
- Represents flow as hyperlinks
- Client (Java web app) follows the links

vs SOAP

Hard-code/configure:

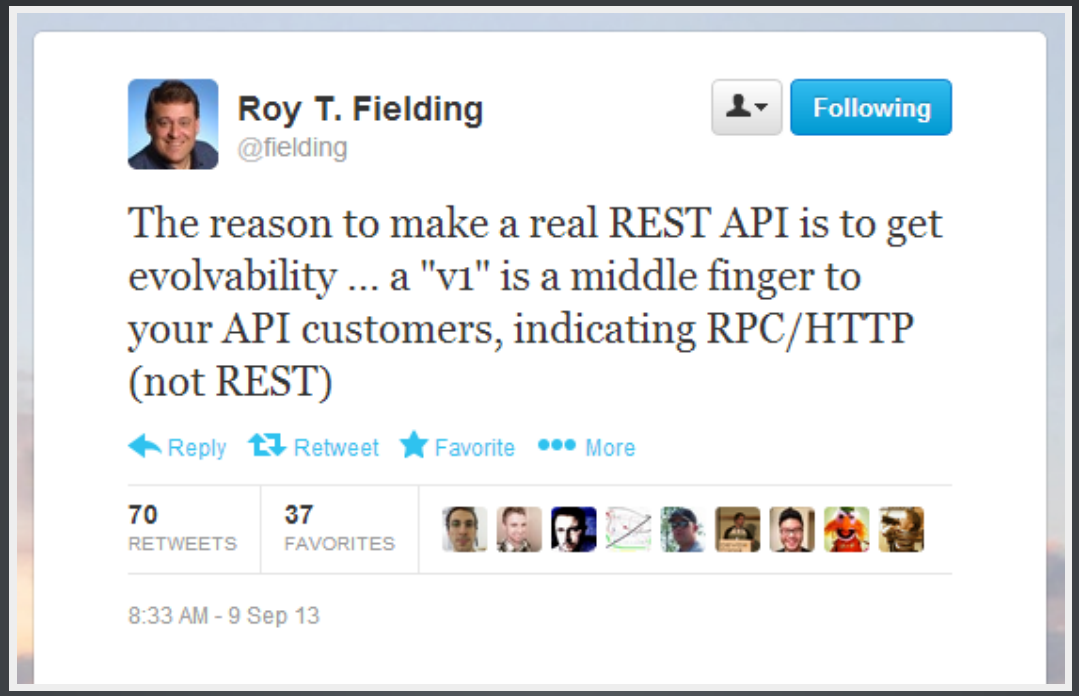
- all possible flows in the client
- data bindings from one response to next request
- ...and maintain in lock-step forever more :-(

#2: Versions

Put a version ID in the URL somewhere

http://myerp.acme.com/api/v1/customer/42

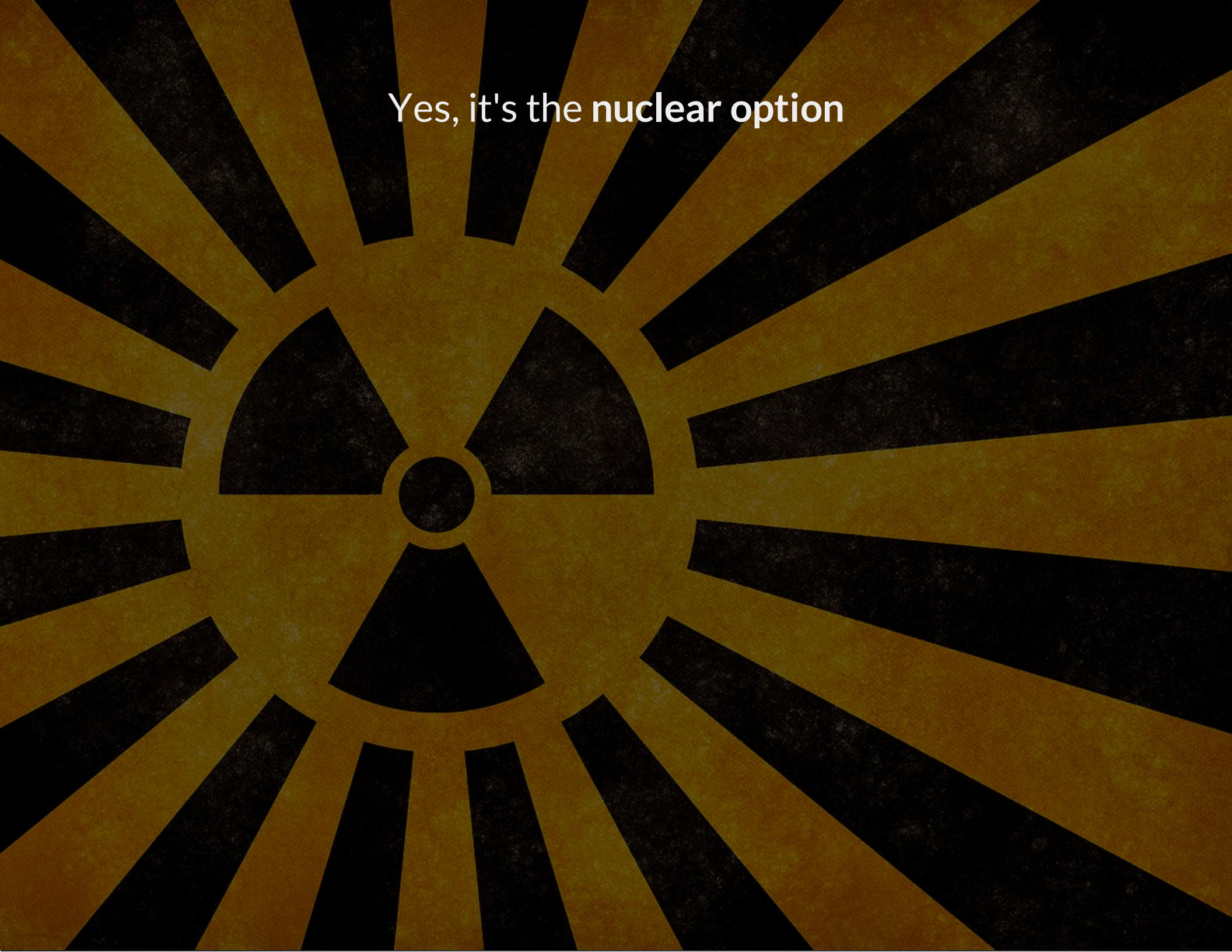
http://v1.api.myerp.acme.com/customer/42


Roy T. Fielding
@fielding
Following
The reason to make a real REST API is to get evolvability ... a "v1" is a middle finger to your API customers, indicating RPC/HTTP (not REST)
Reply
Retweet
Favorite
More
70
RETWEETS
37
FAVORITES
8:33 AM - 9 Sep 13

Purists might prefer header-based versioning...

```
>>> GET http://api.acme.com/customer/42 HTTP/1.1
>>> Accept: application/vnd.acme.com.customer+json;v=2,application/json;q

<<< 200 OK
<<< Content-Type: application/vnd.acme.com.customer;v=1
```


Yes, it's the **nuclear option**

However, in an enterprise we usually have different problems:

- Limited consumers
- Consumers we control
- Limited lifetime

K.I.S.S.

"Version in URL" is easier to do well

Works *with* the ICM

We didn't do this :-(

and regretted it before the project was finished


VERSION
ALL THE THINGS
memegenerator.net

Put a "revision" ID in every resource:

```
HTTP/1.1 200 OK
...
Etag: 2-020e25b0efb0ea57349fe2593a3c9cfe
Content-Type: application/vnd.acme.com.foobar+json
```

```
{
    "_id":  "e2593a3c925b0efb0ea57349fe2c4b2c",
    "_rev": "2-020e25b0efb0ea57349fe2593a3c9cfe",

    "foo":  "bar",
    "bar":  "baz"
}
```

Borrowed from **CouchDB** :-)

Why?

Certainty

Know when your cache is stale

```
>>> GET order/12345 HTTP/1.1
>>> If-None-Match: 2-020e25b0efb0ea57349fe2593a3c9cfe

<<< HTTP/1.1 304 Not Modified
```

or

```
<<< HTTP/1.1 200 OK
<<< Etag: 3-550e8400e29b41d4a716446655440000
<<< ... (and here is the new one)
```

Idempotent Updates

```
>>> PUT order/12345 HTTP/1.1
>>> If-Match: 2-020e25b0efb0ea57349fe2593a3c9cfe
>>> {updated Payload}

<<< HTTP/1.1 200 OK
<<< Etag: 3-550e8400e29b41d4a716446655440000
<<< ...
```

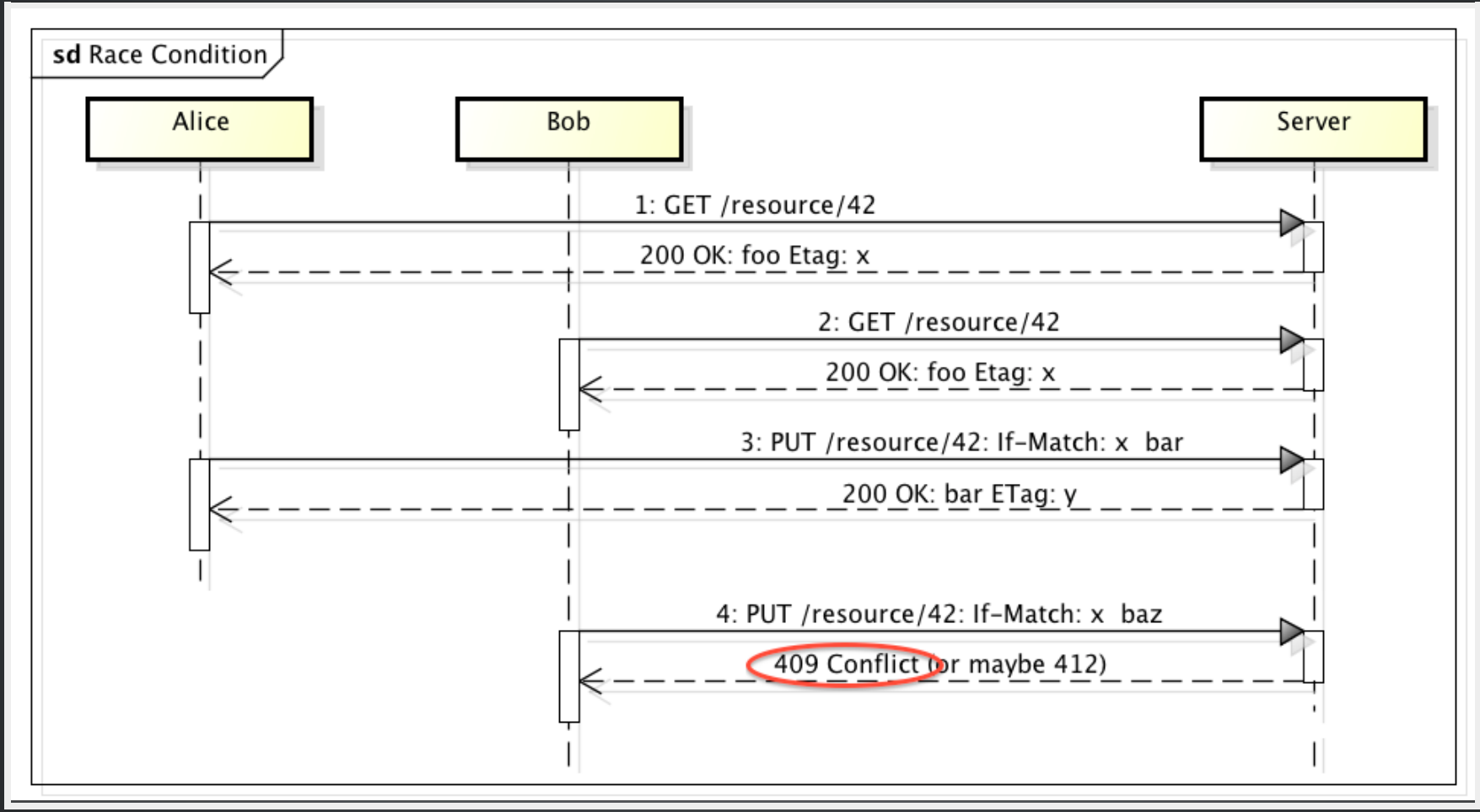
...later...

```
>>> PUT order/12345 HTTP/1.1
>>> If-Match: 2-020e25b0efb0ea57349fe2593a3c9cfe
>>> {updated Payload}

<<< HTTP/1.1 409 Conflict
<<< ...
```

Avoiding Race Conditions

aka Optimistic Locking



Slight Problem...

Many things in SAP are not explicitly versioned.

Cost to derive version ID may be significant

Use where it makes sense

#4: No Partial Updates

You're just asking for trouble.

What happens when you send a request twice?
add some caching?
have > 1 client?

K.I.S.S.

```
>>> GET customer/42 HTTP/1.1
```

```
<<< HTTP/1.1 200 OK
<<< Etag: 2-020e25b0efb0ea57349fe2593a3c9cfe
<<< {Payload}
```

```
>>> PUT customer/42 HTTP/1.1
>>> If-Match: 2-020e25b0efb0ea57349fe2593a3c9cfe
>>> {updated Payload}
```

```
<<< HTTP/1.1 200 OK
<<< Etag: 3-550e8400e29b41d4a716446655440000
<<< {updated Payload}
```

#5: Use Synthetic Resources

RESTful "Resource" !== DB Entity
!== BOR Object

Say we have a customer.

Want to see all about them?

```
GET customer/42 HTTP/1.1
```

```
{ "_id": "42", "_rev": "12-eee8f9c6b8a9477b9eb4954c2b00acf9",
  "name": { "firstname": "Joe", "lastname": "Bloggs",
  "_links": {"self": "/customer/42/name"}},
  "addresses": [
    { "_id": "1", "_rev": "...",
      "type": "mail", "city": "Richmond", "street": "...",
      "_links": {"self": "/customer/42/address/1"}
    },
    { "_id": "2", "_rev": "...",
      "type": "work", "city": "Melbourne", "street": "...",
      "_links": {"self": "/customer/42/address/2"}
    }],
  "_links": {
    "self": "/customer/42",
    "http://rel.acme.com/addresses": "/customer/42/addresses"
  }
}
```

Only their Addresses?

```
GET customer/42/addresses HTTP/1.1
```

```
{ "addresses": [
    { "_id": "1", "_rev": "...",
      "type": "mail", "city": "Richmond", "street": "...",
      "_links": {"self": "/customer/42/address/1"}
    },
    { "_id": "2", "_rev": "...",
      "type": "work", "city": "Melbourne", "street": "...",
      "_links": {"self": "/customer/42/address/2"}
    }],
  "_links": {
    "self": "/customer/42/addresses",
    "parent": "/customer/42"
  }
}
```

Update an Address?

```
PUT customer/42/address/1 HTTP/1.1
If-Match: 1-77ee3bf06b0f4616931870ed772842e5
```

```
{ "type": "mail",
  "city": "East Melbourne",
  "poBox": "PO Box 418", ... }
```

Server replies:

```
HTTP/1.1 200 OK
Etag: 2-5a7a1d9157544d668ddb1dae41401c87
```

Add an Address?

```
POST customer/42/addresses HTTP/1.1
```

```
{ "type": "home",
  "city": "Richmond",
  "street": "Bridge Road", ... }
```

Server replies:

```
HTTP/1.1 201 Created
Location: customer/42/address/3
Etag: 1-9d406e22d08a4d1187b584665ffd9c7b
```

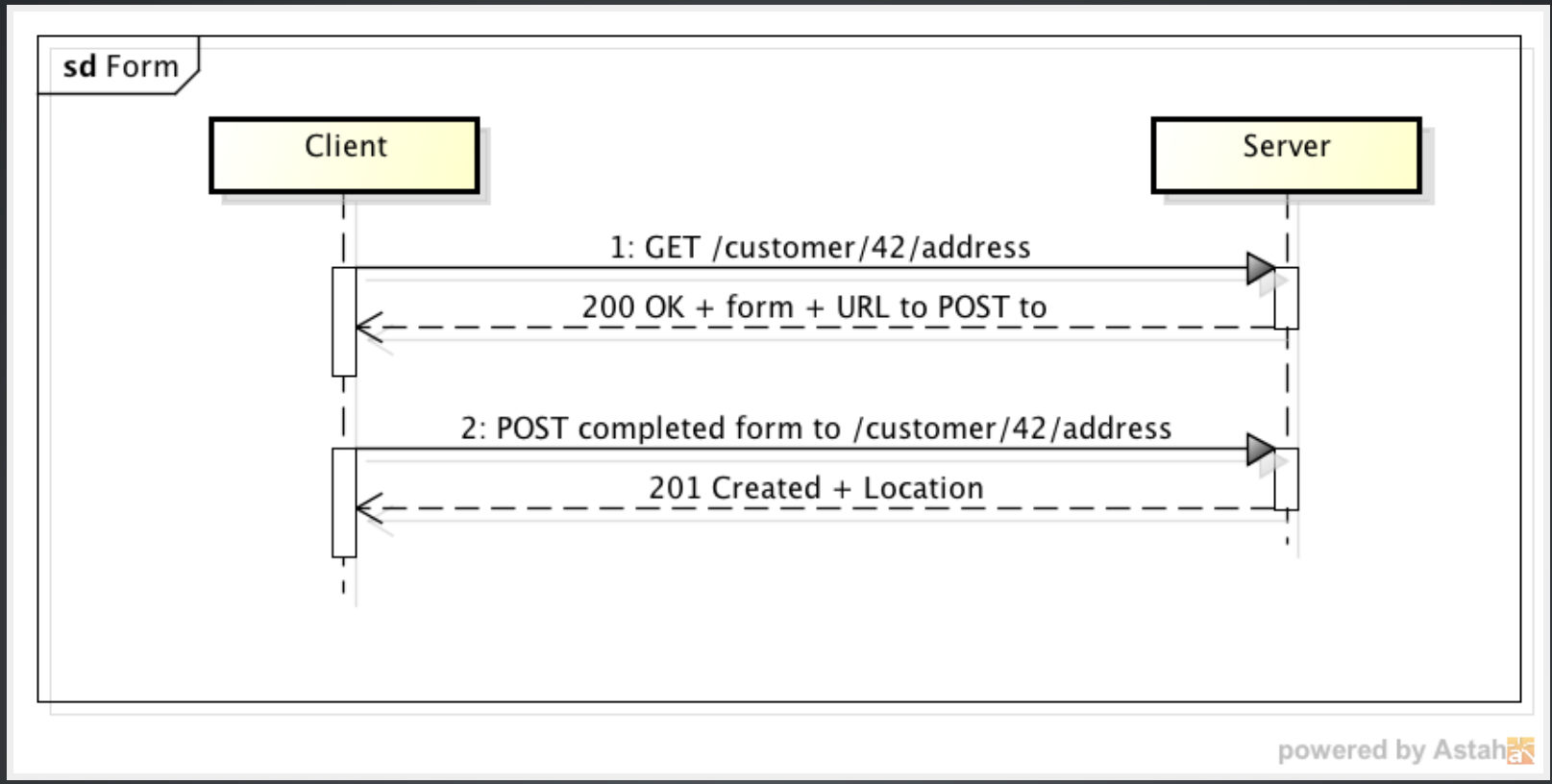
How to construct the POST request?

They could read docoumentation...

or:

Make the API more self-documenting

#6: Use Forms


sd Form
Client
Server
1: GET /customer/42/address
200 OK + form + URL to POST to
2: POST completed form to /customer/42/address
201 Created + Location
powered by Astah

Follow the links

```
GET customer/42 HTTP/1.1
```

```
{ "_links": {
    "self": "/customer/42",
    "http://rel.acme.com/addresses": "/customer/42/addresses"
  }
}
```

Get the Form

```
GET customer/42/addresses HTTP/1.1
```

```
{ "type": "",
  "city": "",
  "street": "",
  "poBox": "",
  "state": "", ...
}
```

Submit the Form

```
POST customer/42/addresses HTTP/1.1
```

```
{ "type": "home",
  "city": "Richmond",
  "street": "Bridge Road",
  "poBox": null,
  "state": "Victoria", ...
}
```

Or Use HTML?

```
<form method="post" action="/customer/42/addresses">
        <select name="type">
                <option value="home"></option>
                <option value="work"></option>
                <option value="mail"></option>
        </select>
        <input type="text" name="city">
        <input type="text" name="street">
        <input type="text" name="poBox">
        <input type="text" name="state">
</form>
```

HTML has some nice, universally understood form semantics

But can add complexity...

If in doubt, prefer simplicity

#1: HATEOAS!

e.g. for Atom

```
<link rel="edit" href="/customer/42/addresses/edit">
```

e.g. for JSON

```
{ "_links": {
  "edit": { "href": "/customer/42/addresses/edit",
            "name": "edit-or-add" }
}}
```

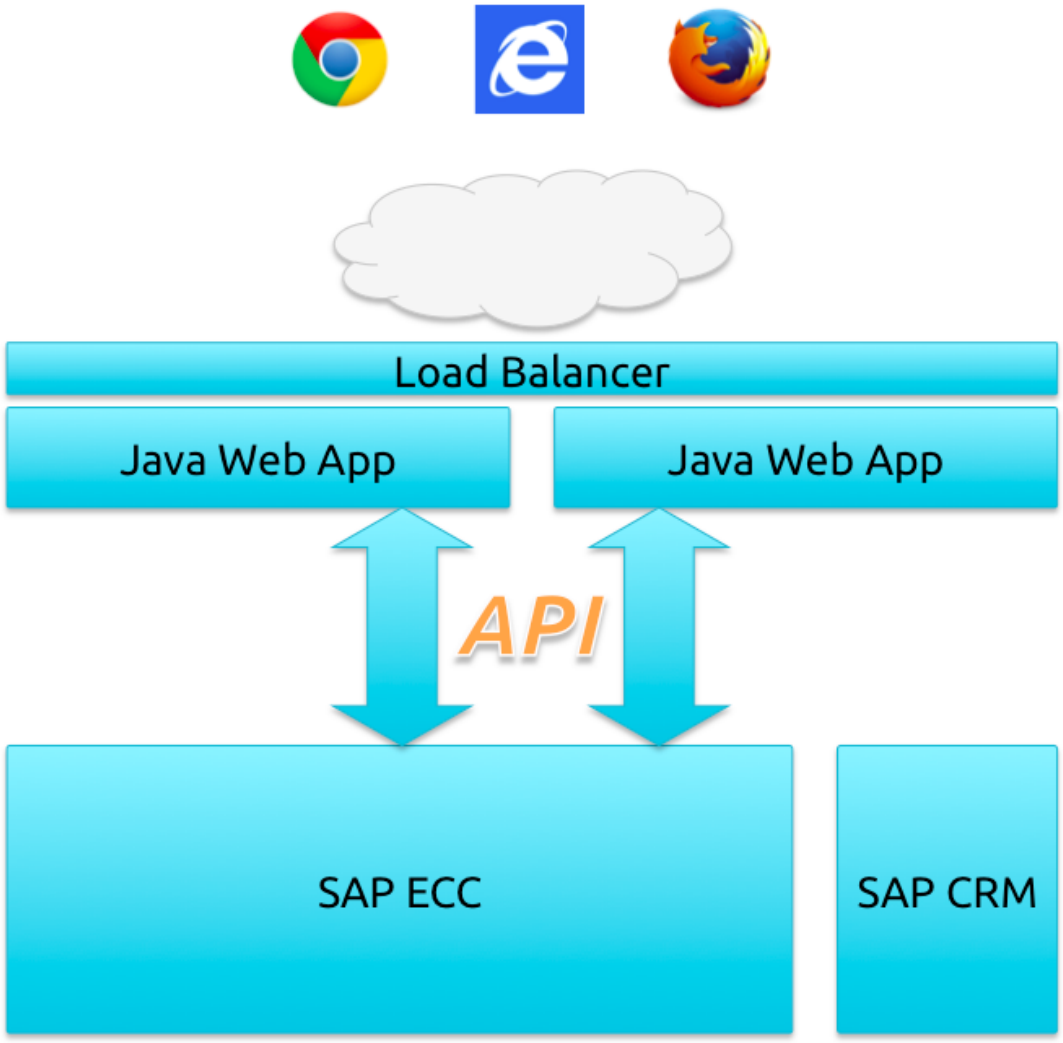
yes, it's **HAL**

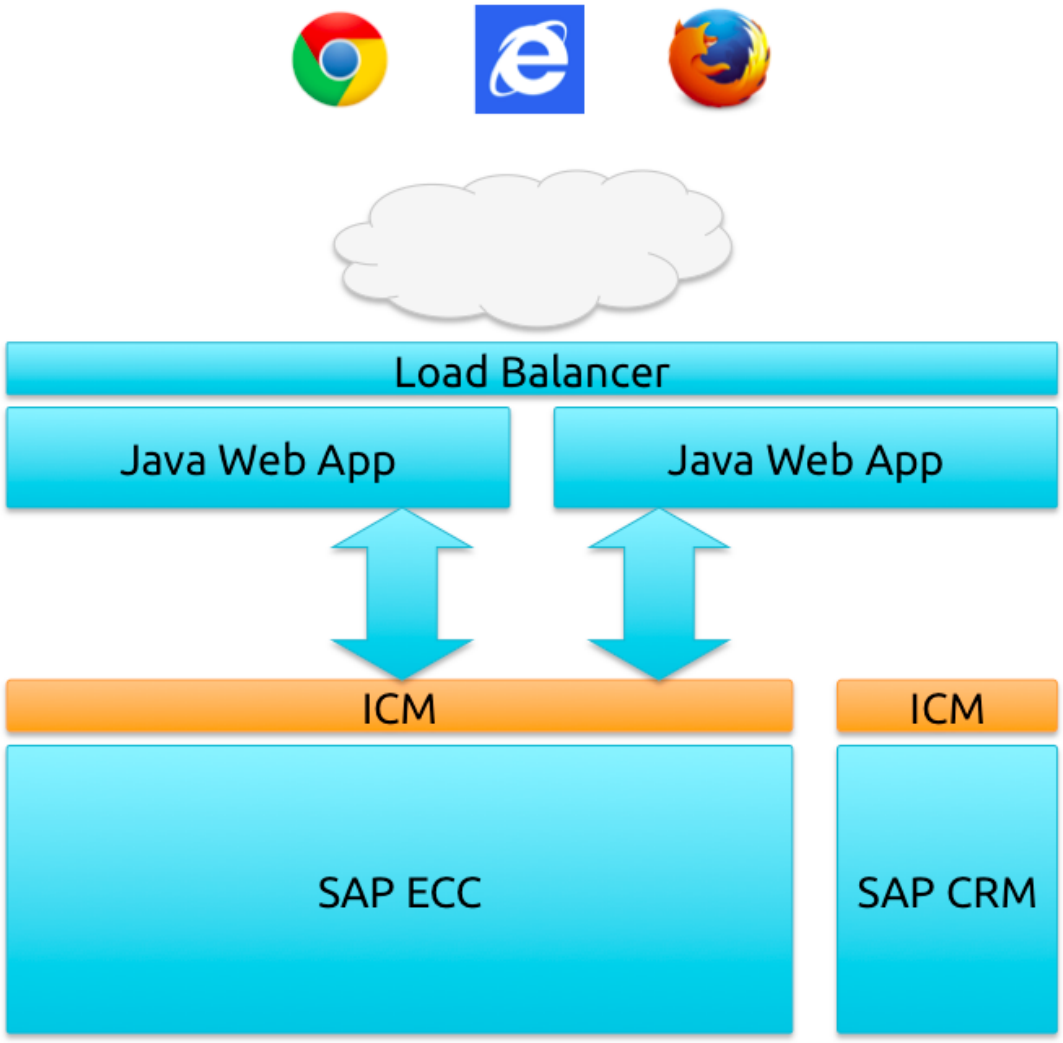
#7: Caching

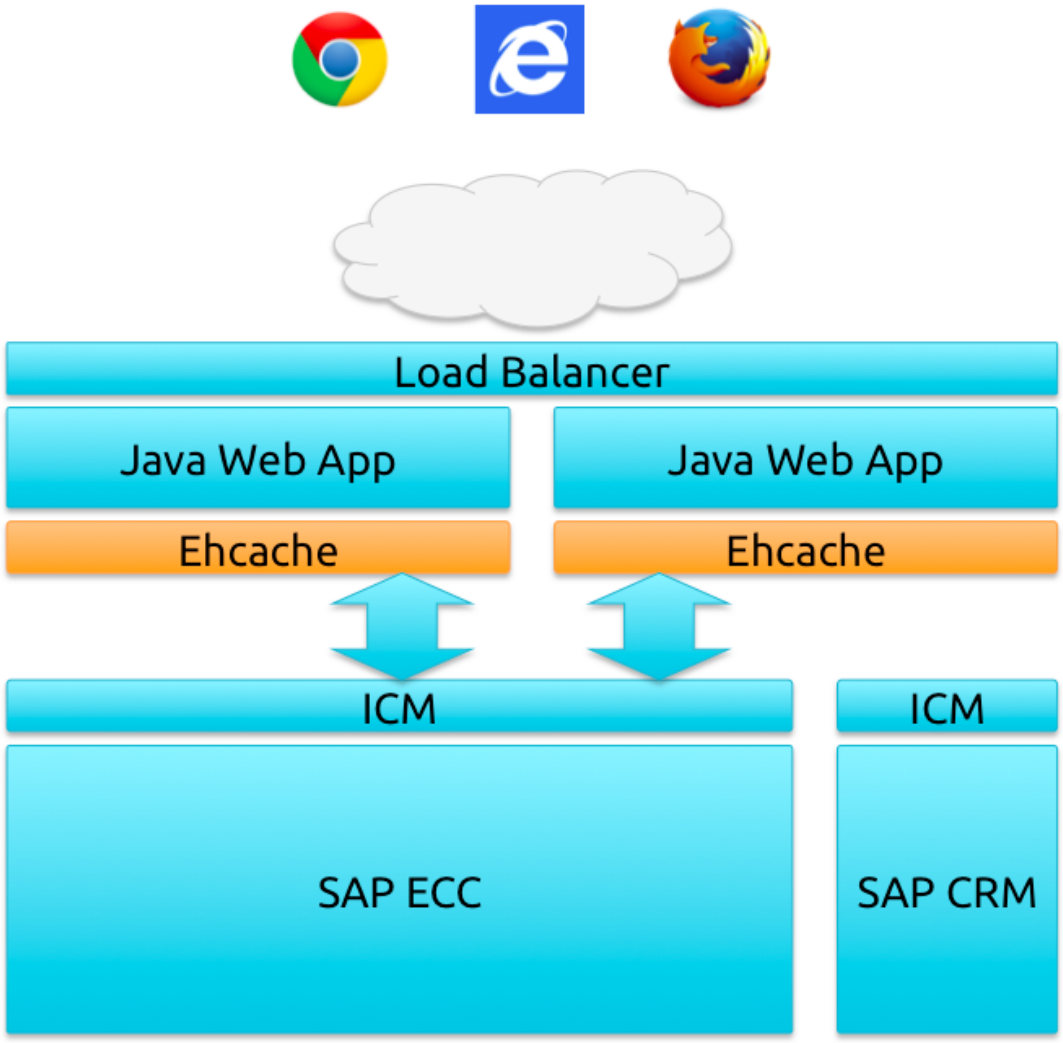
One of the most tangible benefits of the REST style

Caching is the magic sauce that makes the web run.

REST is the web, for machines


Load Balancer
Java Web App
Java Web App
API
SAP ECC
SAP CRM


Load Balancer
Java Web App
Java Web App
ICM
ICM
SAP ECC
SAP CRM


Load Balancer
Java Web App
Java Web App
Ehcache
Ehcache
ICM
ICM
SAP ECC
SAP CRM

Our Strategy

SAP ICM Cache for long-lived static content

Ehcache for session-based caching.

- Defined triggers in the app which flushed the session cache
- e.g. after payment of an invoice, etc.

Think about this from day # 1

Let it inform the design of your representations

Some Tips

1. Server-side caches
2. Segregate private and public data into different resources
3. Avoid multiple views of the same representation
4. Client-side caches
5. Work with the business to accept 'stale' content

Pagination

We decided not to do pagination

"GET items 250...300"-type queries are inefficient in ABAP

API returned full list of results; cached in Ehcache

Client app then paginated through cache content

#8: Meaningful Response Codes

HTTP is an Application Protocol.

Use the rich list of **Status Codes** for signalling

And then there's the **7xx** codes ;-)


200
OK

500

Internal Server Error


404
Not Found

409

Conflict

418

I'm a teapot

There's 78 of them on **Wikipedia**

Don't reinvent the wheel.

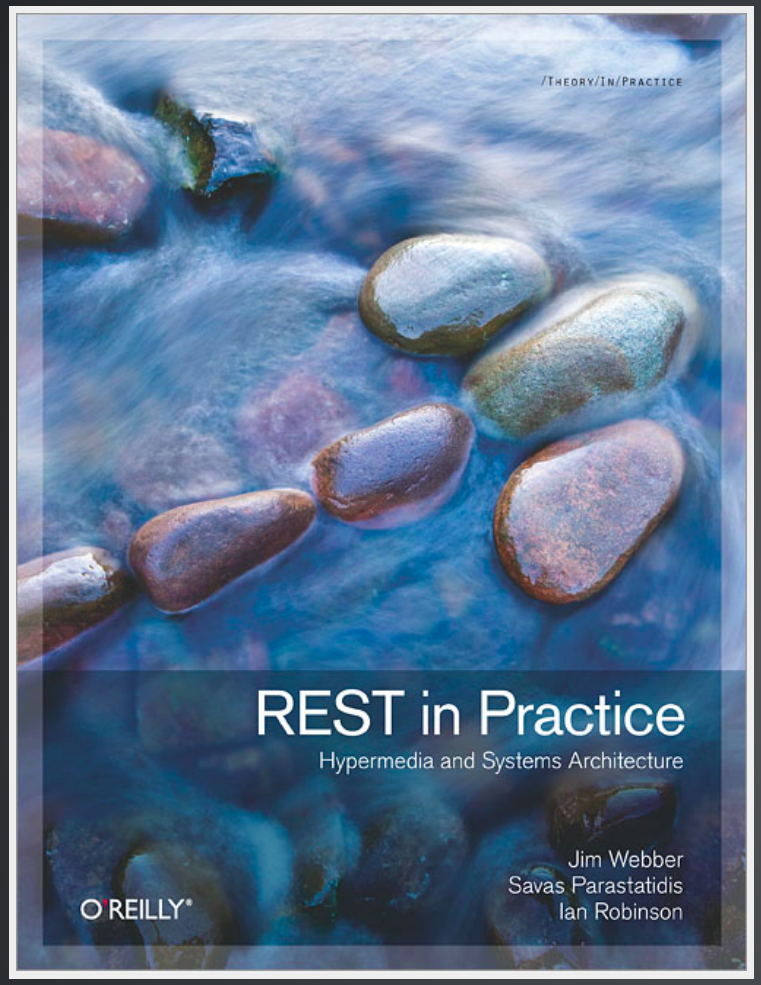
#9: Proper Headers

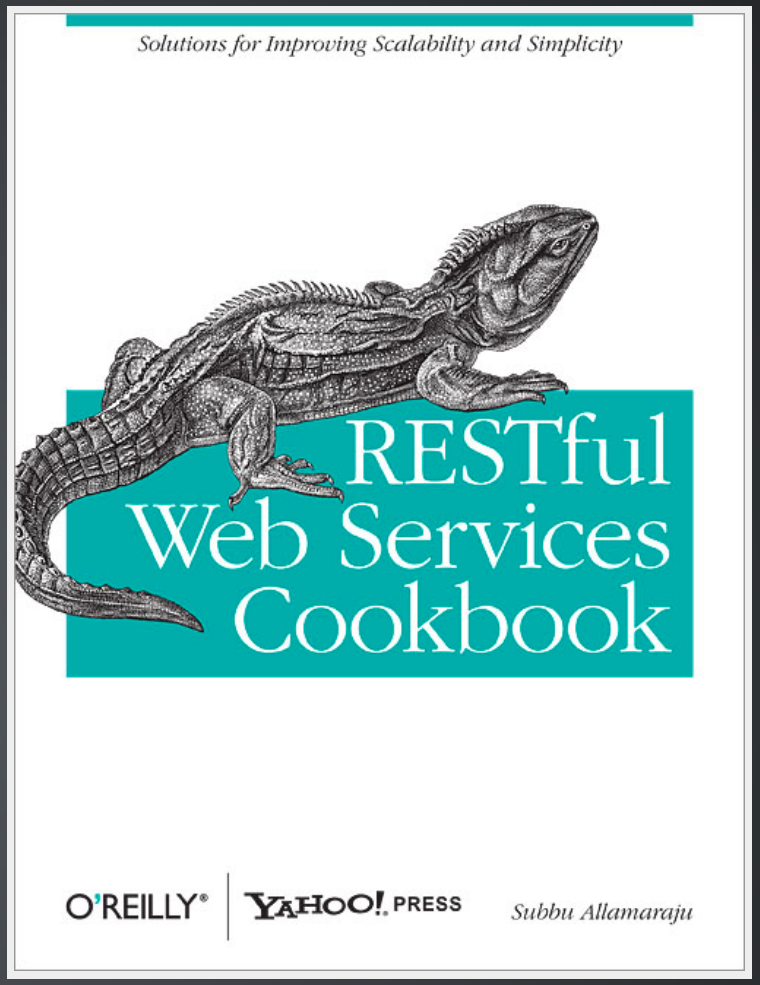
#10: RTFM

REST is not a Protocol

REST is an Architectural Style

There is no one way of doing things


/Theory/In/Practice
REST in Practice
Hypermedia and Systems Architecture
Jim Webber
Savas Parastatidis
Ian Robinson
O'REILLY®


Solutions for Improving Scalability and Simplicity
RESTful
Web Services
Cookbook
O'REILLY®
YAHOO! PRESS
Subbu Allamaraju

Read Applicable RFCs

RFC 2616 (HTTP/1.1)

RFC 5988 and **IANA Link Relations**

Roy's canonical thesis

IANA MIME Type Register

JSON.org, **JSON-LD**, **HAL**, etc...

Keep Reading

InfoQ

Apigee Best Practices **eBook** and **blog**

Blogs by **Erik Wilde**, **Mark Nottingham** or **Stu Charlton**

The Yahoo **rest-discuss** group

etc.

When implementing RESTful APIs,
a lot fewer choices are made *for* you.

Speed of implementation is not an aim of REST.

Long-term evolvability is.

#11: Keep Your Consumers Close

Get involved in client development

Helps you build a better API

In Our Case...

Automated Testing

Use stubs for isolation

Helps communication and coordination

Reality Checks

Build APIs which are usable

Avoid flights of fancy

"Why don't we implement _____?"

Abstraction

Working with non-SAP developers helps to avoid SAPinese

~~WERKS~~:	Warehouse | Store | Factory
~~KUNNR~~:	Customer ID
~~000000000042~~:	42
~~X~~:	true


#12: This is
Difficult!

It's non-trivial to build a decent RESTful API.

It will take time, and probably more than 1 attempt.

Your project might not need a full-blown RESTful API.

And that's okay!

1. Build a HTTP Handler in ABAP
2. Do it well.
3. Learn from it.
4. YAGNI

Just don't call it REST ;-)

Q&A

Thank you very much!

Tweet us at **@sufw** and **@zcust01**

Thank you for the Photos!

Red Letter Box, by Les Haines

"It's a SOA", by Oliver Widder (Geek and Poke)

Nuclear Burst Grunge Flags, by Nicolas Raymond

HTTP Status Cats, by GirlieMac

Go Player, by Peter Dahlgren