How to Embrace the Browser, Architect's Edition

enhanceconf London

4 March, 2016

Stefan Tilkov, innoQ

@stilkov

innoQ

Lessons from the past: CORBA

- Once, there was this thing called CORBA
- It was supposed to rule the world!
- The browser came along, using some stupid text-based protocol
- Thank God it got CORBA support!
- Everyone was supposed to switch to it …

Lessons from the past: WS-*

- “We can’t ignore this Web thing anymore”
- “Let’s just do RCP over XML and HTTP”
- “Port 80 is open, so let’s use it!”
- Re-invent IIOP, IDL, CORBA Services as SOAP, WSDL, WS-*
- Proprietary integration
- Strong vendor support

Anatomy of a SOAP “Web service”

- Doesn’t expose individual resources with URIs
- No links, no forms, no hypermedia
- Uses HTTP as a transport
- Can’t use advanced HTTP features (e.g. caching)
- Specific instead of generic
- “Tunnels” through the Web

That “REST” thing

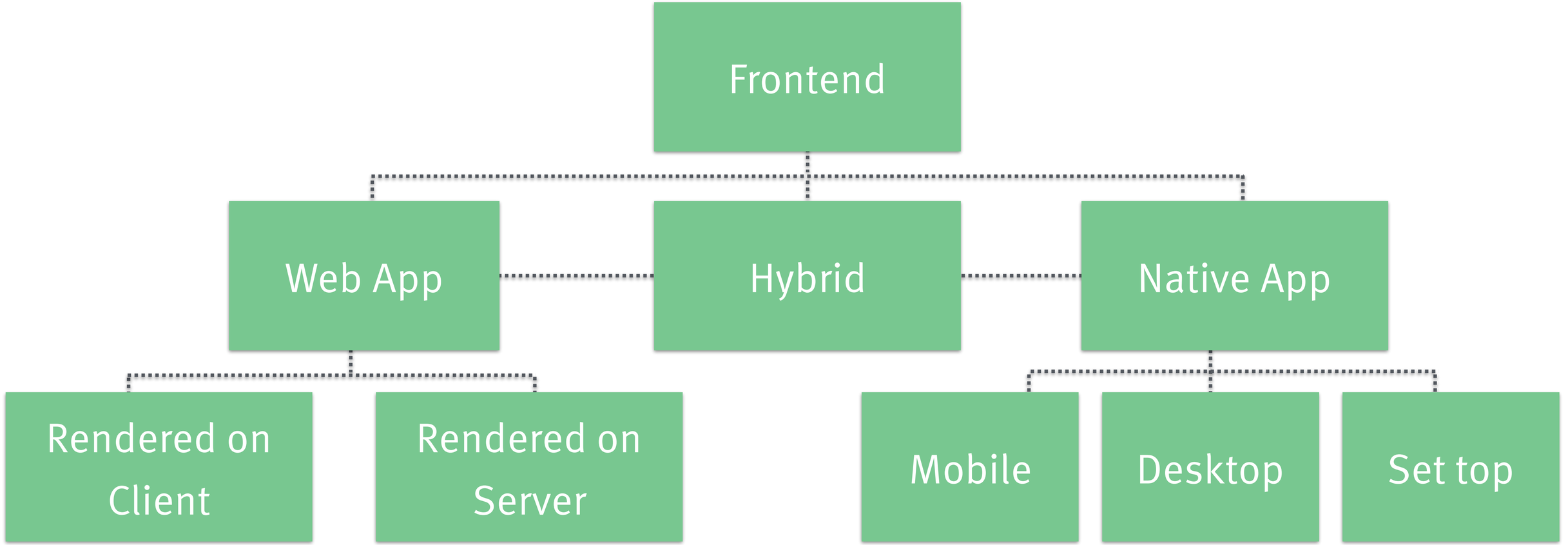
- Architectural style, defined after the fact
- Identification of resources, HatEoAS, self-descriptive messages, representations
- Highlighted what contstraints need to be adhered to gain benefits, and what tradeoffs involved are

RESTful Web services

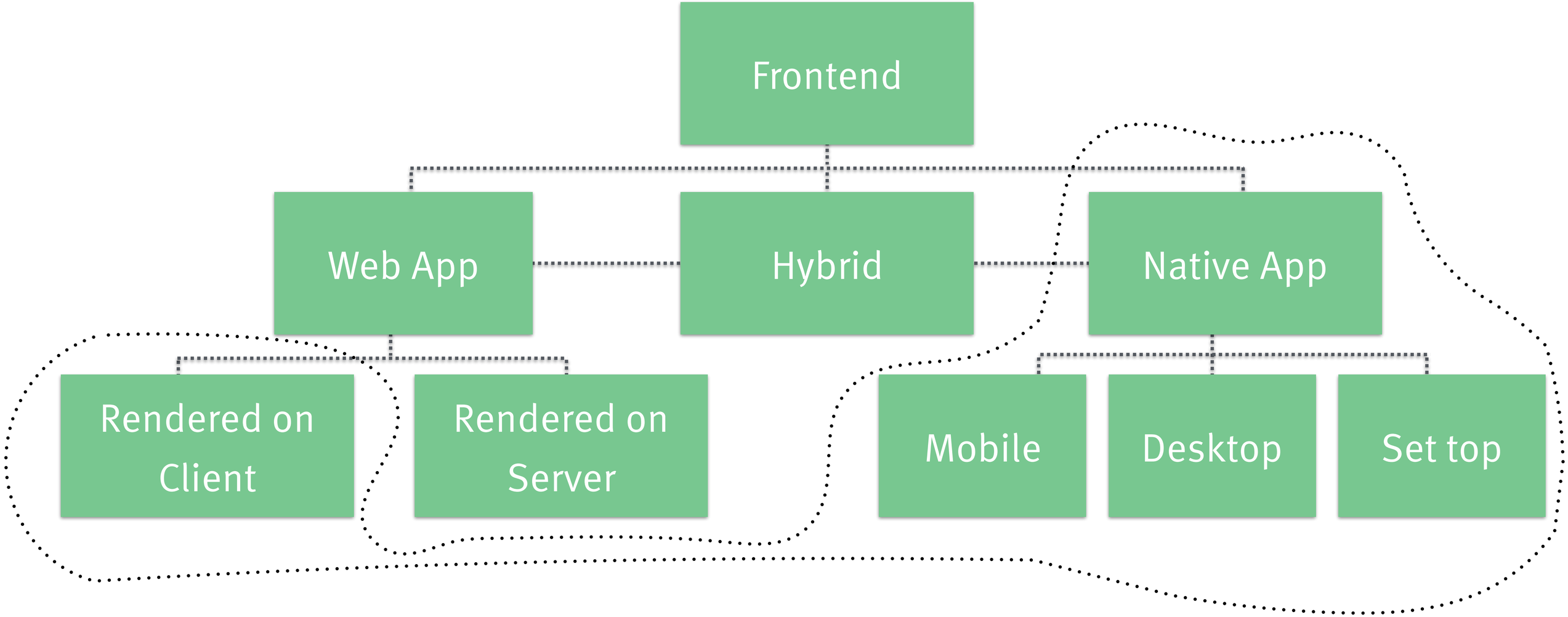
- Embrace, don’t oppose, the Web’s architecture
- Exchange local optimum for benefits of generic approach
- Simpler
- More efficient
- More interoperable

What’s the client side analogy?

Frontend, we've got frontends



Frontend, we've got frontends



Assumption:

JS-centric web apps can be *as good as* native apps

They shouldn't be as bad!

“Web service”[1)]

- Use HTTP as transport
- Ignore verbs
- Ignores URIs
- Expose single “endpoint”
- Fails to embrace the Web

“Web app”[2)]

- Uses browser as runtime
- Ignores forward, back, refresh
- Does not support linking
- Exposes monolithic “app”
- Fails to embrace the browser

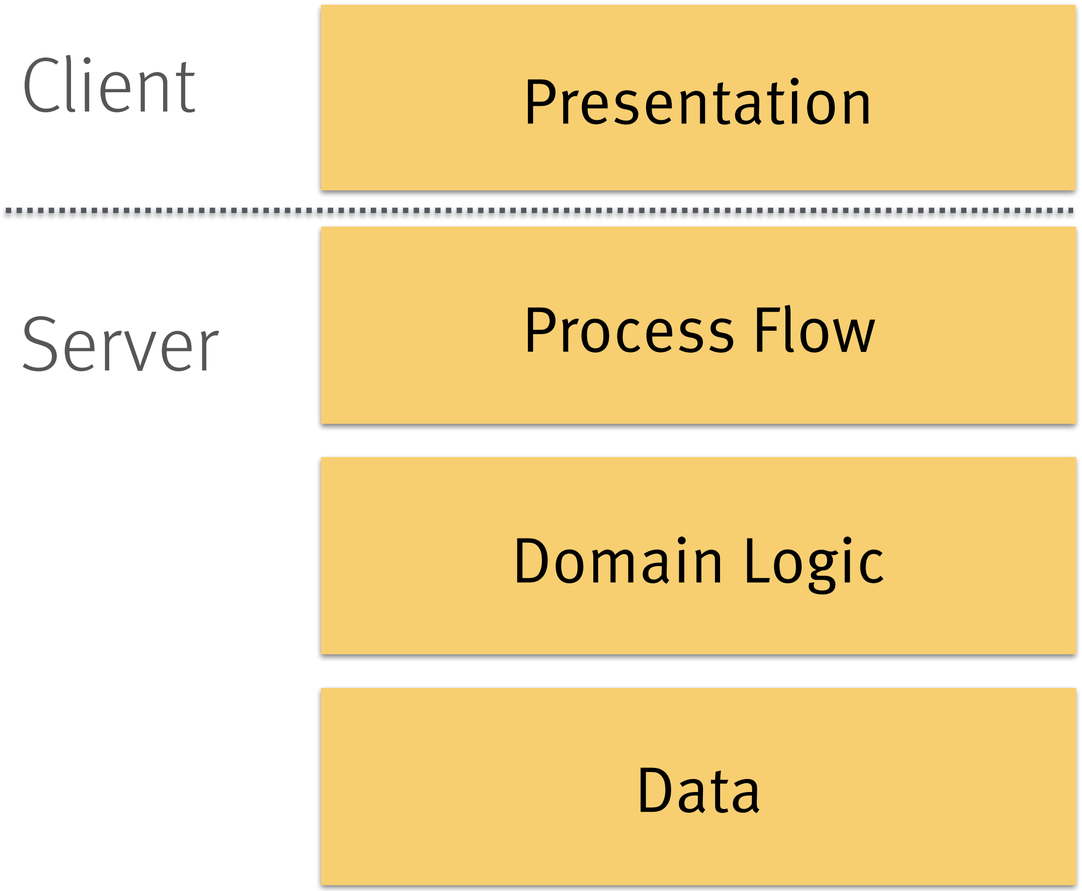
[1)] in the SOAP/WSDL sense

[2)] built as a careless SPA

ROCA: Resource-oriented Client Architecture

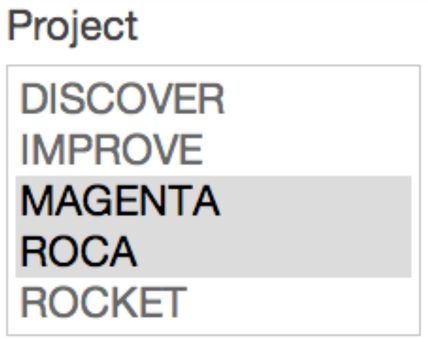
http://roca-style.org

The web-native way of distributing logic

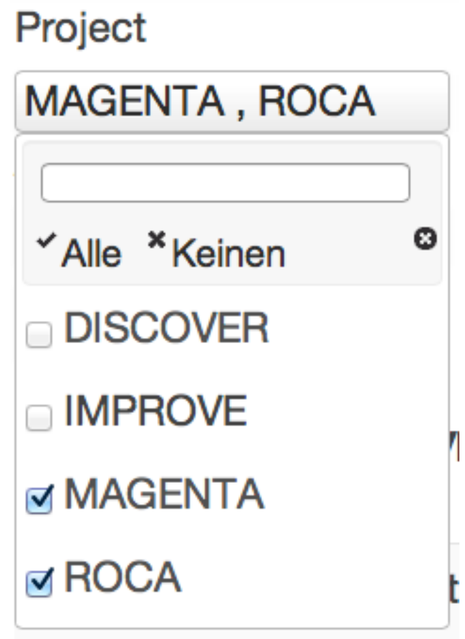


- Rendering, layout, styling on an *unknown* client
- Logic & state machine on server
- Client user-agent extensible via code on demand

```
<div class="filter-column">
    <label for="project">Project</label>
    <select class="multiselect" id="project"
            name="project" size="5" multiple>
      <option>DISCOVER</option>
      <option>IMPROVE</option>
      <option >MAGENTA</option>
      <option>ROCA</option>
      <option>ROCKET</option>
    </select>
</div>
```




```
$('.multiselect', context).each(function() {
        $(this).multiselect({
                selectedList: 2,
                checkAllText: "Alle",
                uncheckAllText: "Keinen"
        }).multiselectfilter({label:"",
                              width:"200px"});
});
```

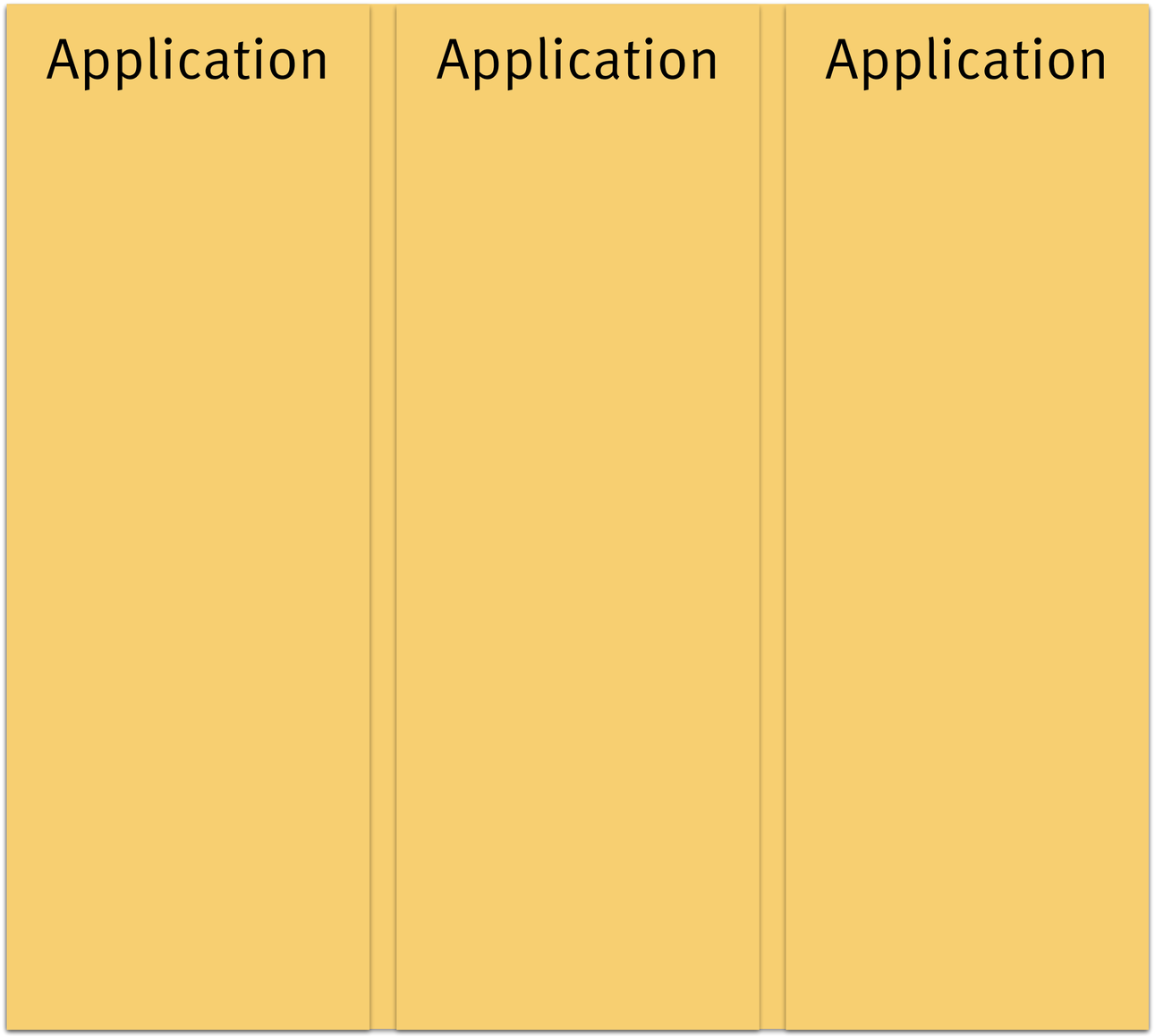


HTML & Hypermedia

- In REST, servers expose a hypermedia format
  - Option 1: Just use HTML
  - Option 2: Just invent your own JSON-based, incomplete clone
- Clients need to be RESTful, too
  - Option 1: Use the browser
  - Option 2: Invent your own, JS-based, buggy, incomplete implementation

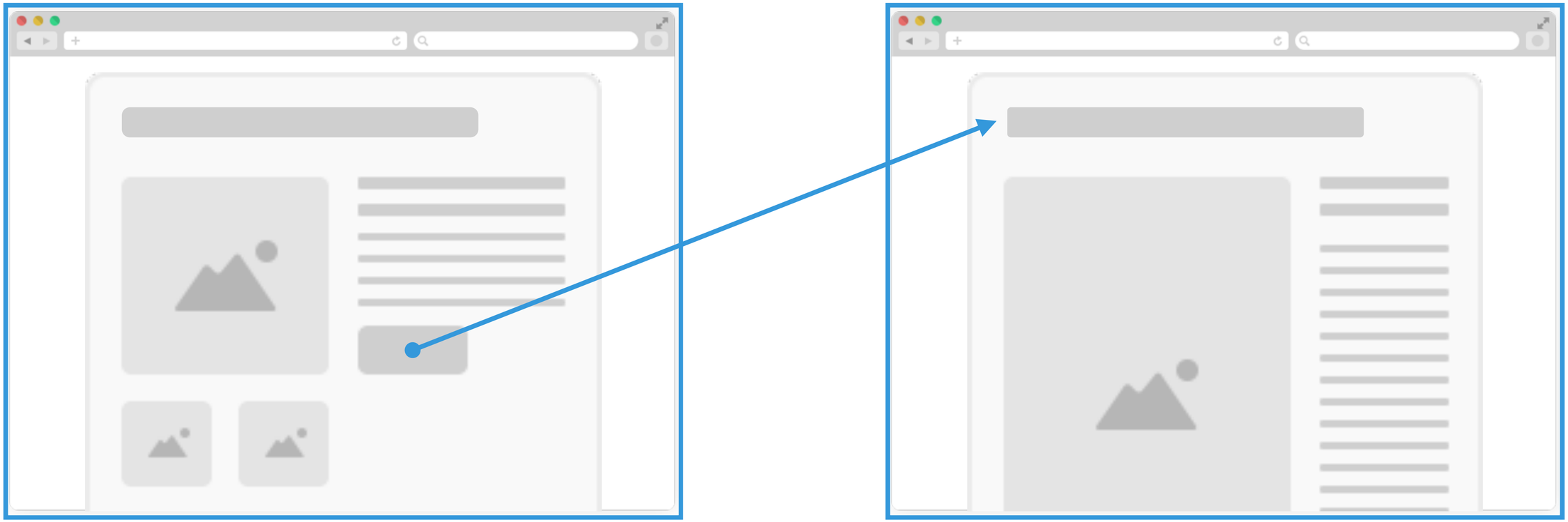
Any sufficiently complicated JavaScript client application contains an ad hoc, informally-specified, bug-ridden, slow implementation of half a browser.

(Me, with apologies to Phillip Greenspun)



- Clients should be modularized as much as servers (cf. µServices)
- Browser as platform
- How to connect separate UIs?

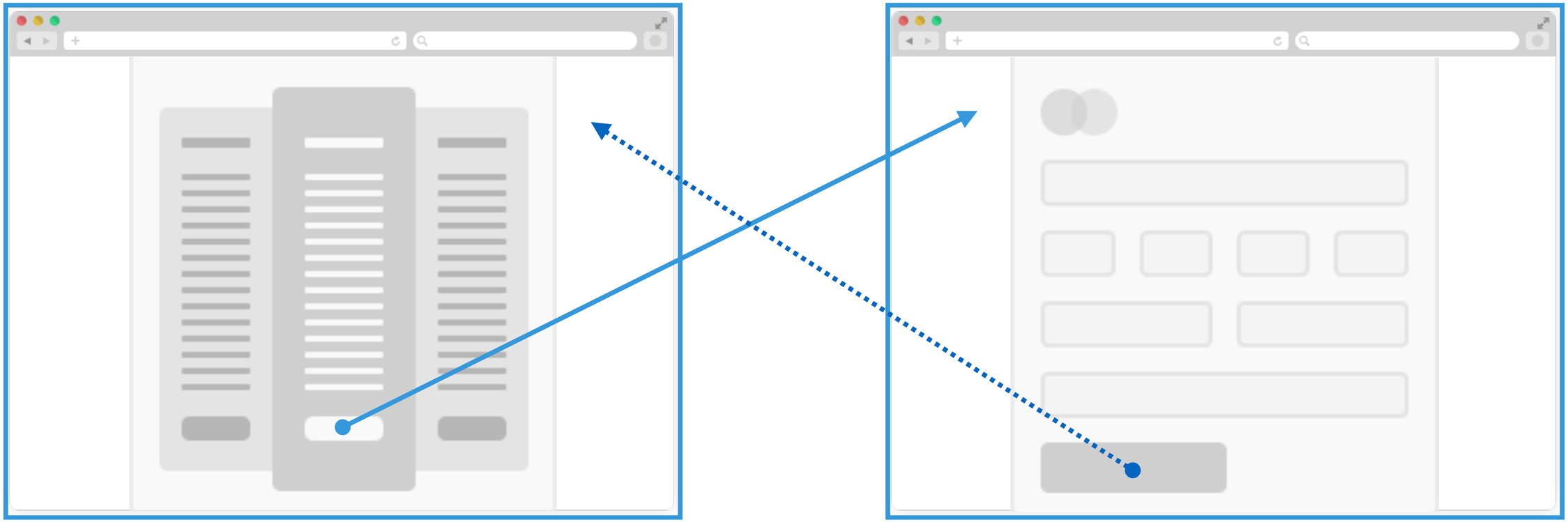
Web UI Integration: Links



System 1

System 2

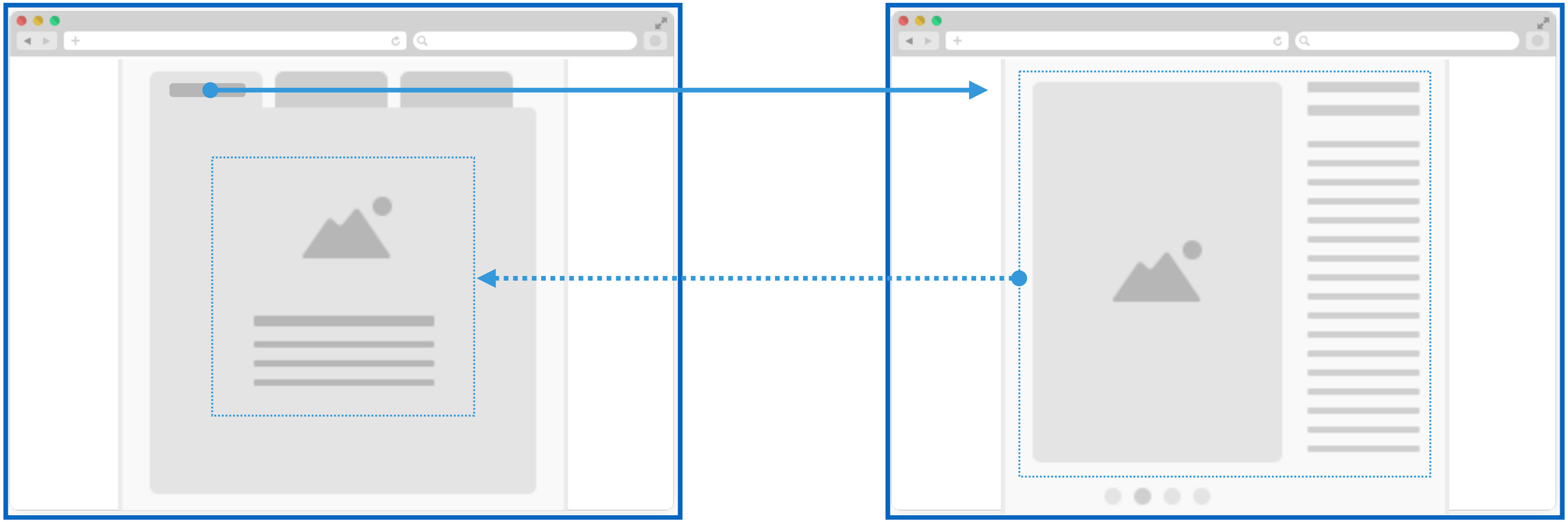
Web UI Integration: Redirection



System 1

System 2

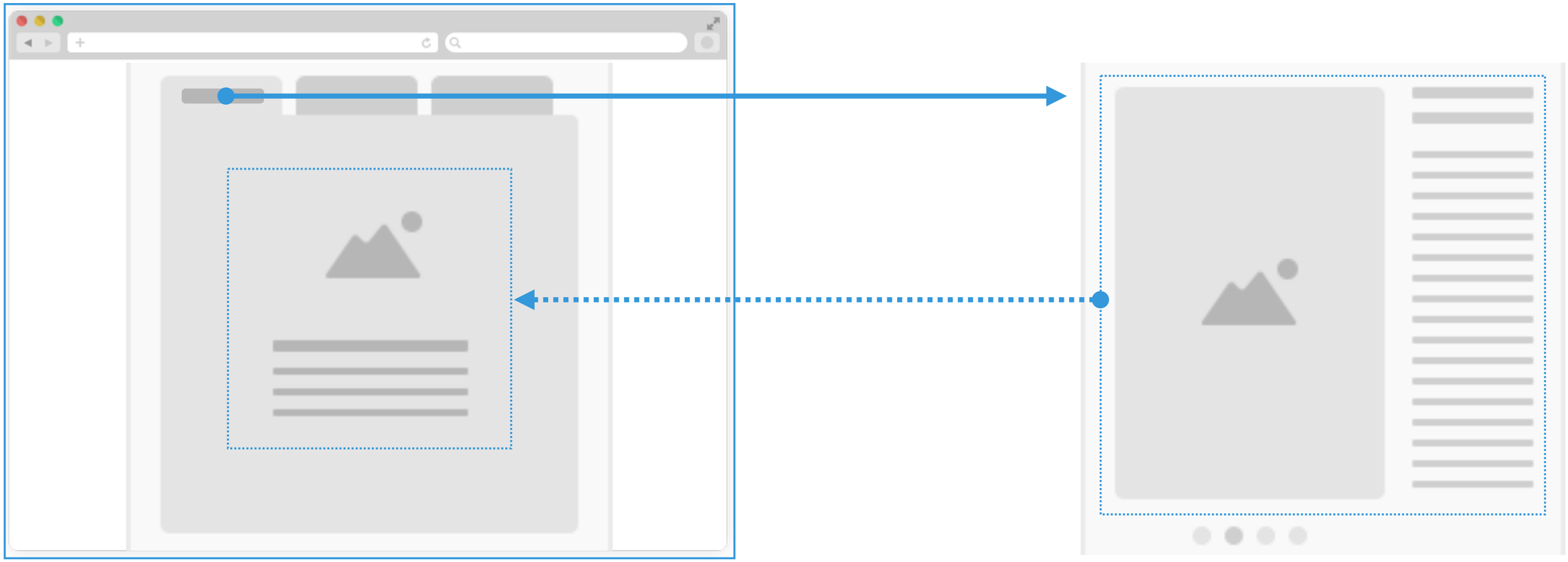
Web UI Integration: Transclusion



System 1

System 2

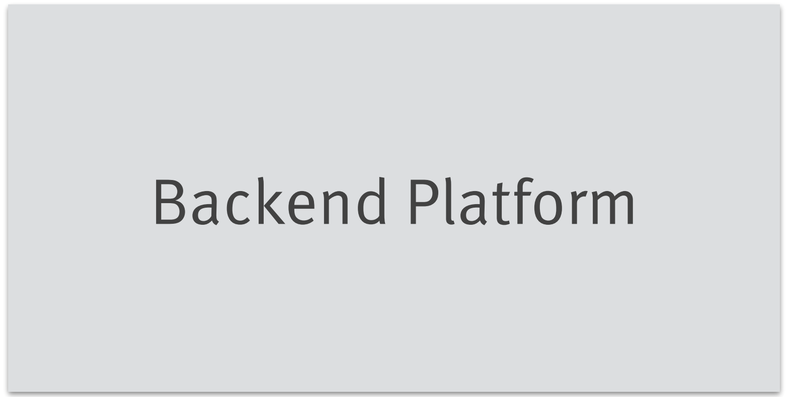
Web UI Integration: Web Components?



System 1

Component

Backend platform goals



- As few assumptions as possible
- No implementation dependencies
- Small interface surface
- Based on standards
- Parallel development
- Independent deployment
- Autonomous operations

What’s the frontend platform analogy?

Frontend Platform

Backend Platform

- As few assumptions as possible
- No implementation dependencies
- Small interface surface
- Based on standards
- Parallel development
- Independent deployment
- Autonomous operations

The browser as a platform

Frontend Platform

Backend Platform

- Independent applications
- Loosely coupled
- Separately deployable
- Based on standard platform
- Updated on the fly
- Any device

How to get away with “just” the Web

- Mobile first
- Responsive design
- Progressive enhancement

- Shared assets
- Pull vs. push
- Sacrifice (some) efficiency

Small frontends, loosely coupled

Summary

The web is more than the sum of its protocols

Constraints are *good*

(when architecture is concerned)

Embrace the web’s constraints – don’t fight them

Thank you –
that’s all I have.

@stilkov

Stefan Tilkov
stefan.tilkov@innoq.com
Phone: +49 170 471 2625

innoQ
www.innoq.com

innoQ Deutschland GmbH

Krischerstr. 100
40789 Monheim am Rhein
Germany
Phone: +49 2173 3366-0

Ohlauer Straße 43
10999 Berlin
Germany
Phone: +49 2173 3366-0

Ludwigstr. 180E
63067 Offenbach
Germany
Phone: +49 2173 3366-0

Kreuzstraße 16
80331 München
Germany
Phone: +49 2173 3366-0

innoQ Schweiz GmbH

Gewerbestr. 11
CH-6330 Cham
Switzerland
Phone: +41 41 743 0116