Andi Scharfstein,
Seminar on Functional Programming 2006

Why are we here?

[Live Demo of the „Orbitz Bug“:

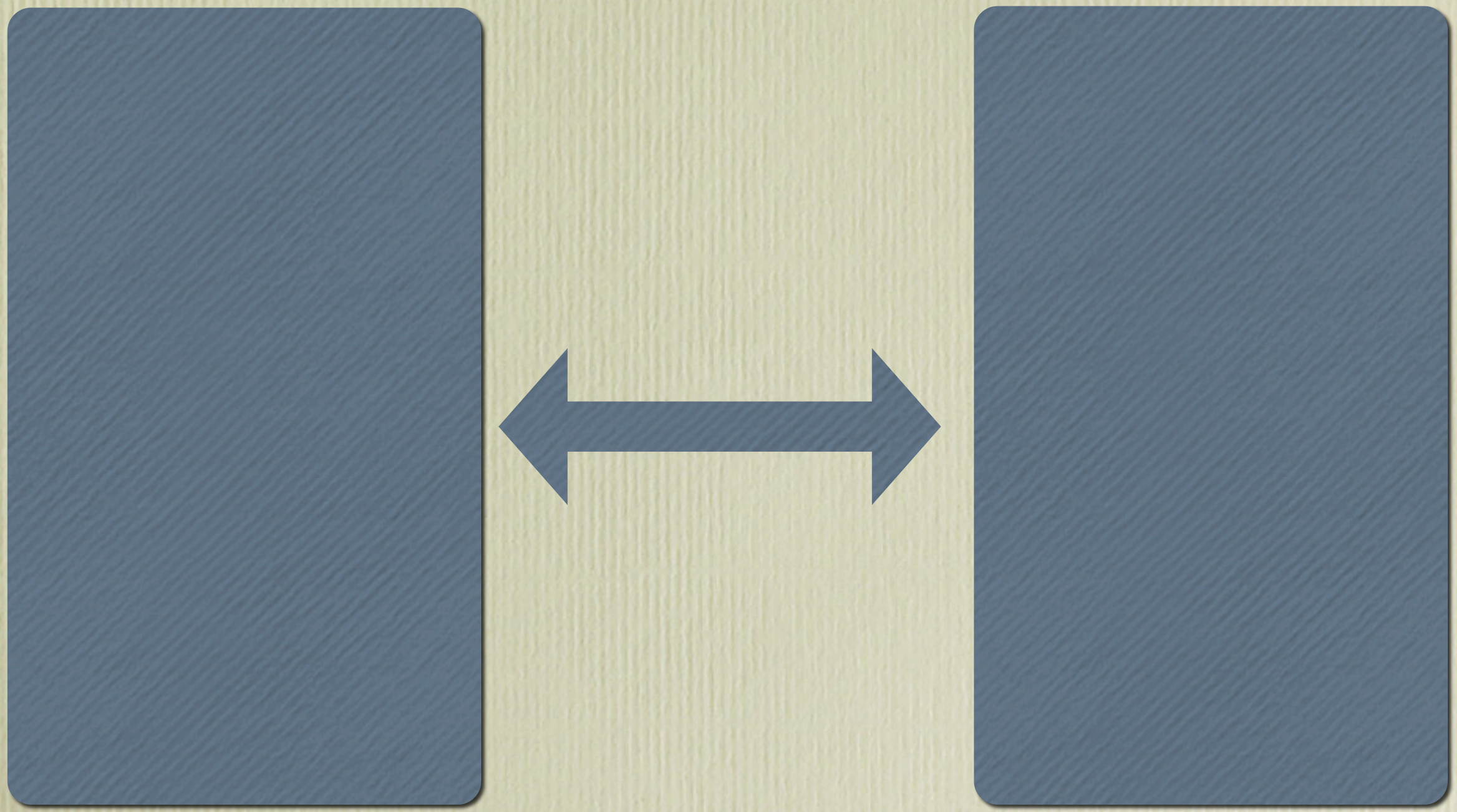
1. Visit orbitz.com in a web browser

2. Open multiple flights in multiple windows]

3. Try to book a flight. Regardless of which flight was selected in your window, the flight that will be booked will always be the flight from the most recently opened window (even if it was closed in the meantime)]

We want to fix such bugs!

Constructing the Model

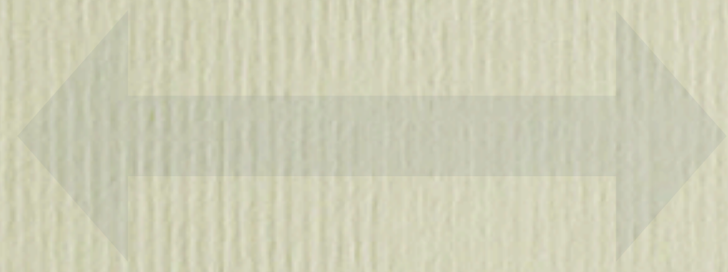


Web Server

Client

The Web Server

Web Server



Client

The Web Server

- Internal storage
- Consists of key/value pairs
- Represented by σ: Key → Value
- Describes server state

Web Server

Client

The Web Server

- Internal storage
- Consists of key/value pairs
- Represented by σ: Key → Value
- Describes server state

- Scripts (dynamic pages, forms)

Client

The Web Server

flight	815
dest.	SF
…	…

Scripts

Web Server

- Internal storage
- Consists of key/value pairs
- Represented by σ: Key → Value
- Describes server state

display-flights.htm	<?php for i in flights display(i) end>
…	…

Client

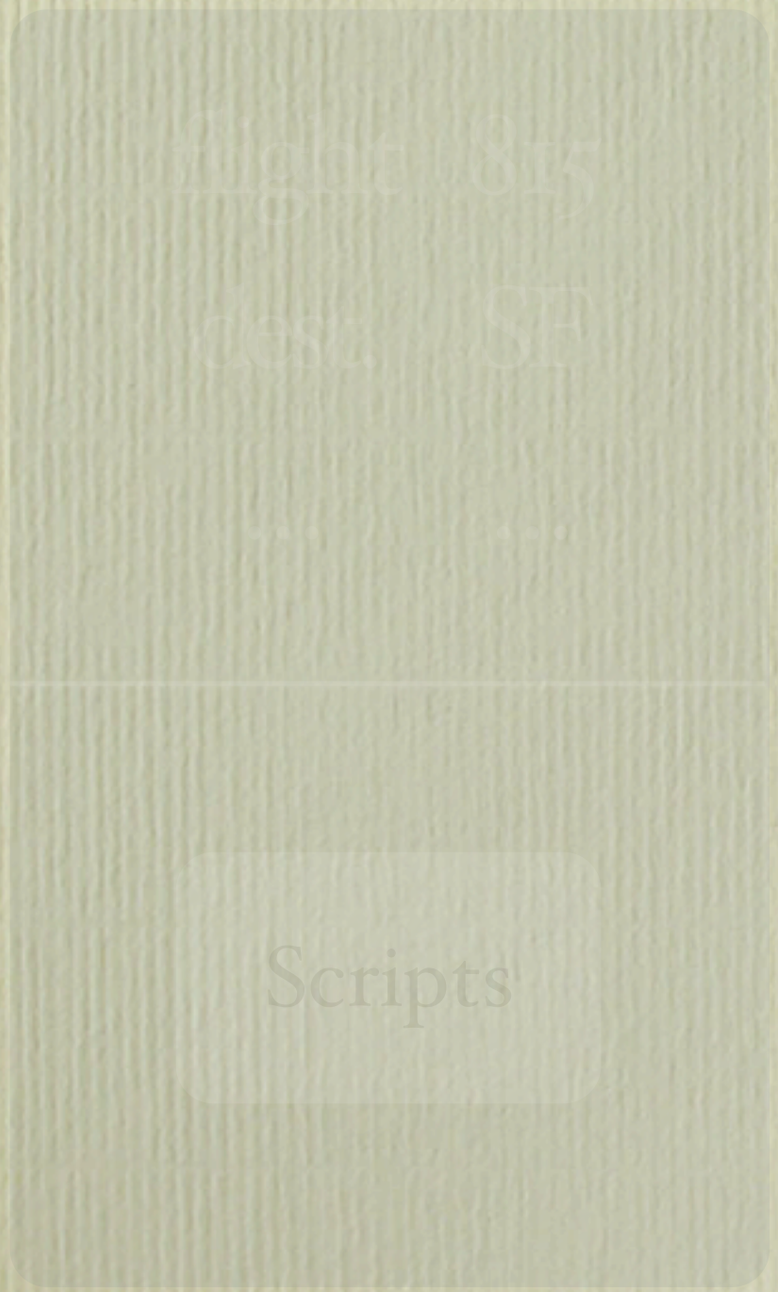
The Web Server

- Internal storage
- Consists of key/value pairs
- Represented by σ: Key → Value
- Describes server state

- Scripts (dynamic pages, forms)
- Lookup function P: URL → Form

Client

The Client



Client

The Client

- Active page

Current form

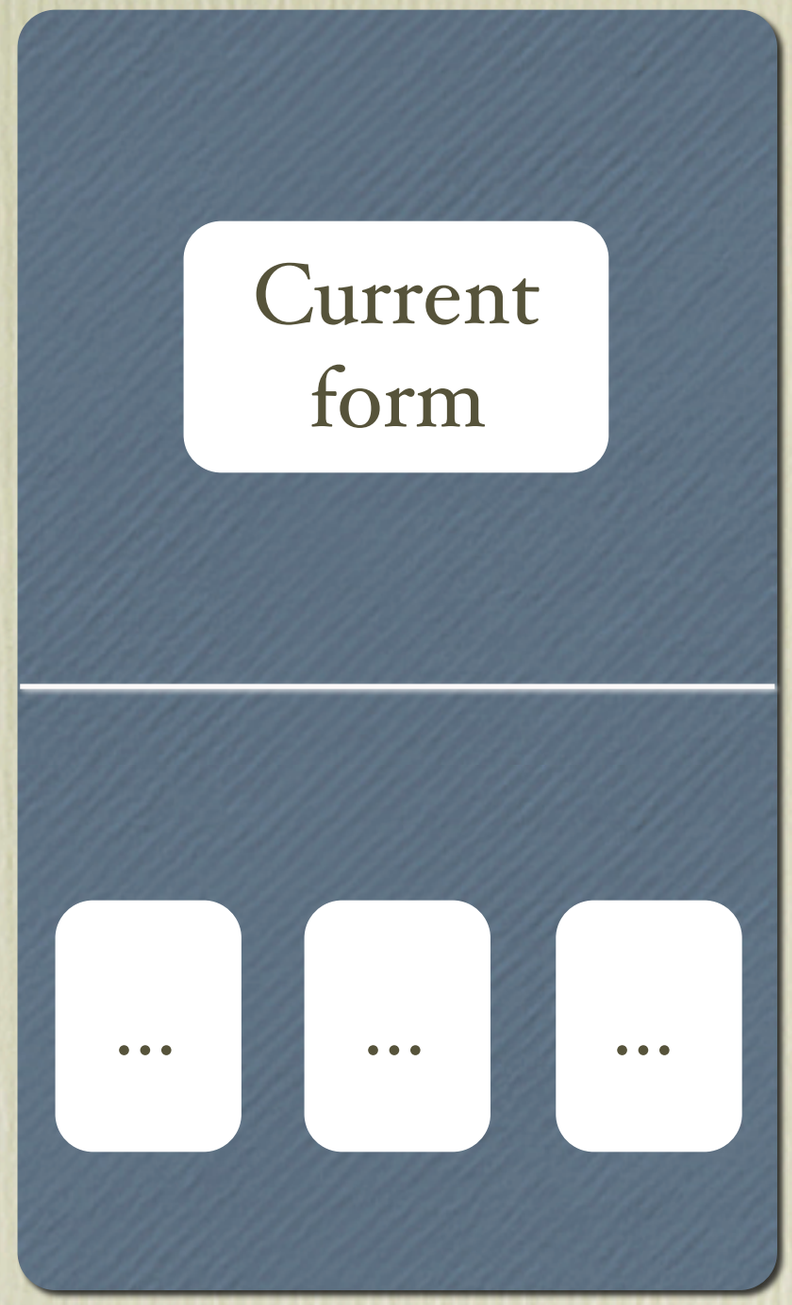
Client

Web Server

Scripts

The Client

- Active page
- Browser cache
- All previously seen forms



Scripts

Web Server

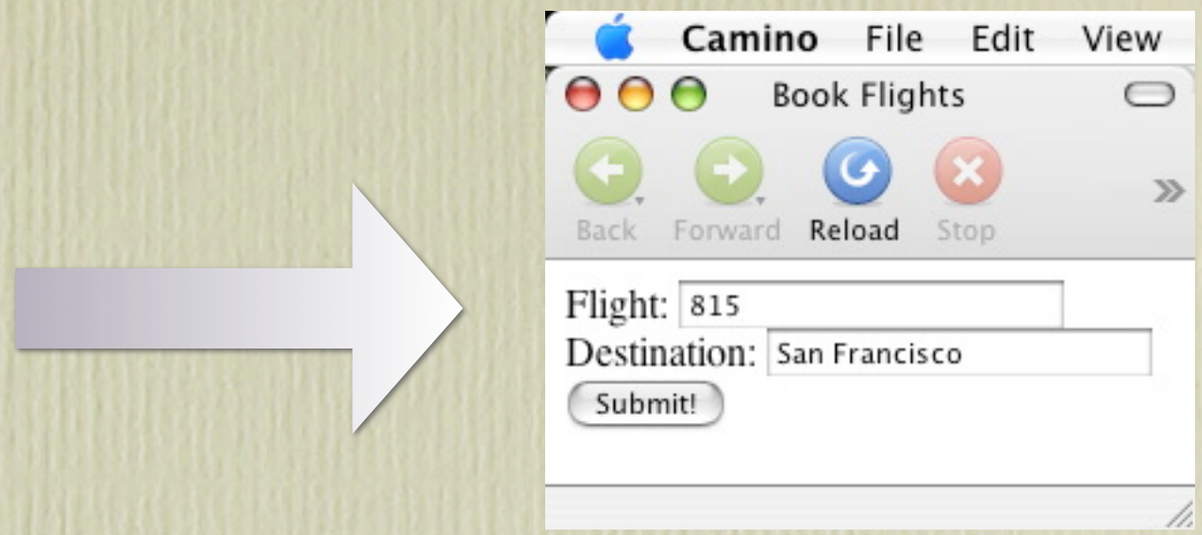
Forms

```
<html>
<form action=‘book-flight.html‘>

   <input name=‘flight‘
          type=‘text‘
          value=‘815‘>

   <input name=‘destination‘
          type=‘text‘
          value=‘San Francisco‘>

   <input name=‘submit‘
          type=‘submit‘
          value=‘Submit!‘>
</form>
</html>
```



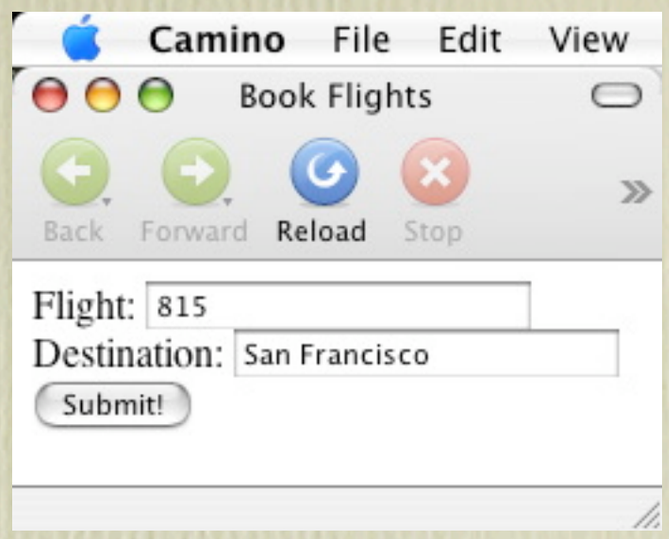
Forms

```
<html>
<form action='book-flight.html'>

   <input name='flight'
          type='text'
          value='815'>

   <input name='destination'
          type='text'
          value='San Francisco'>

   <input name='submit'
          type='submit'
          value='Submit!'>
</form>
</html>
```



Representation:

URL: book-flight.html

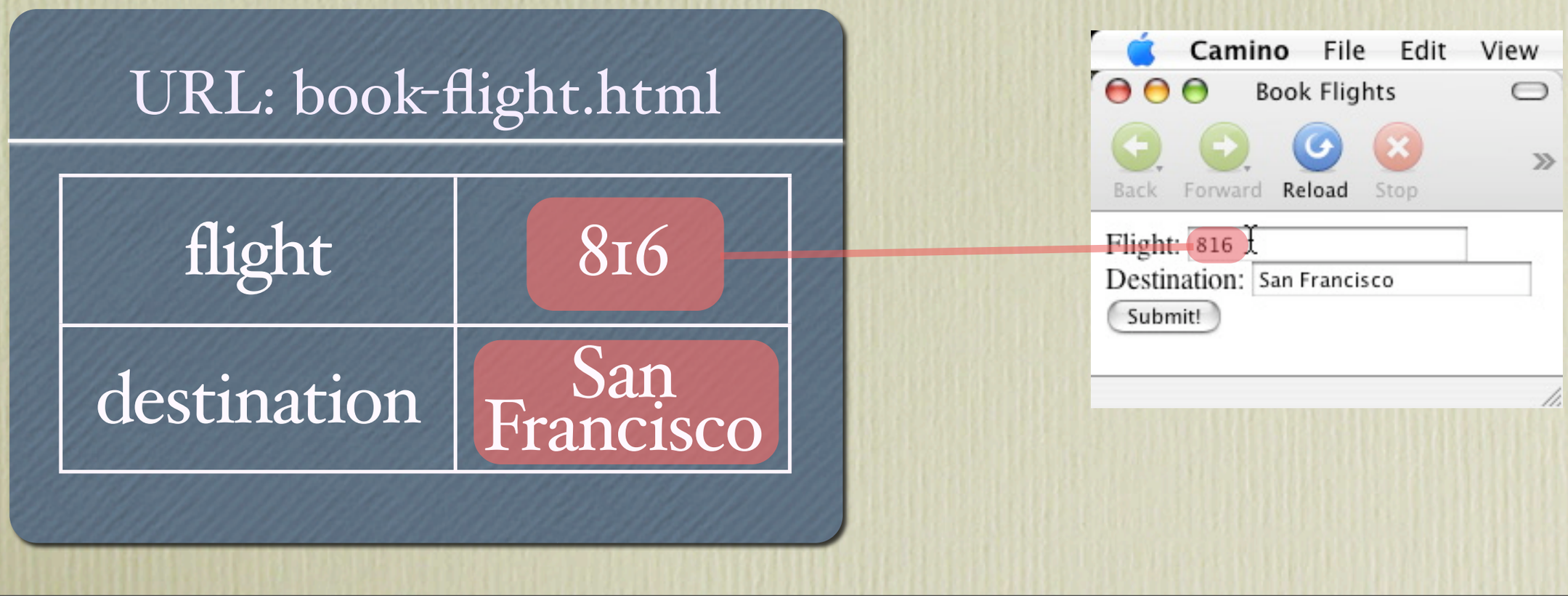
flight	815
destination	San Francisco

Supported Actions

- Users may do any of the following at any time:
  - Enter data into the current form
  - Switch to a cached page
    (e.g., click on the back button)
  - Submit a form

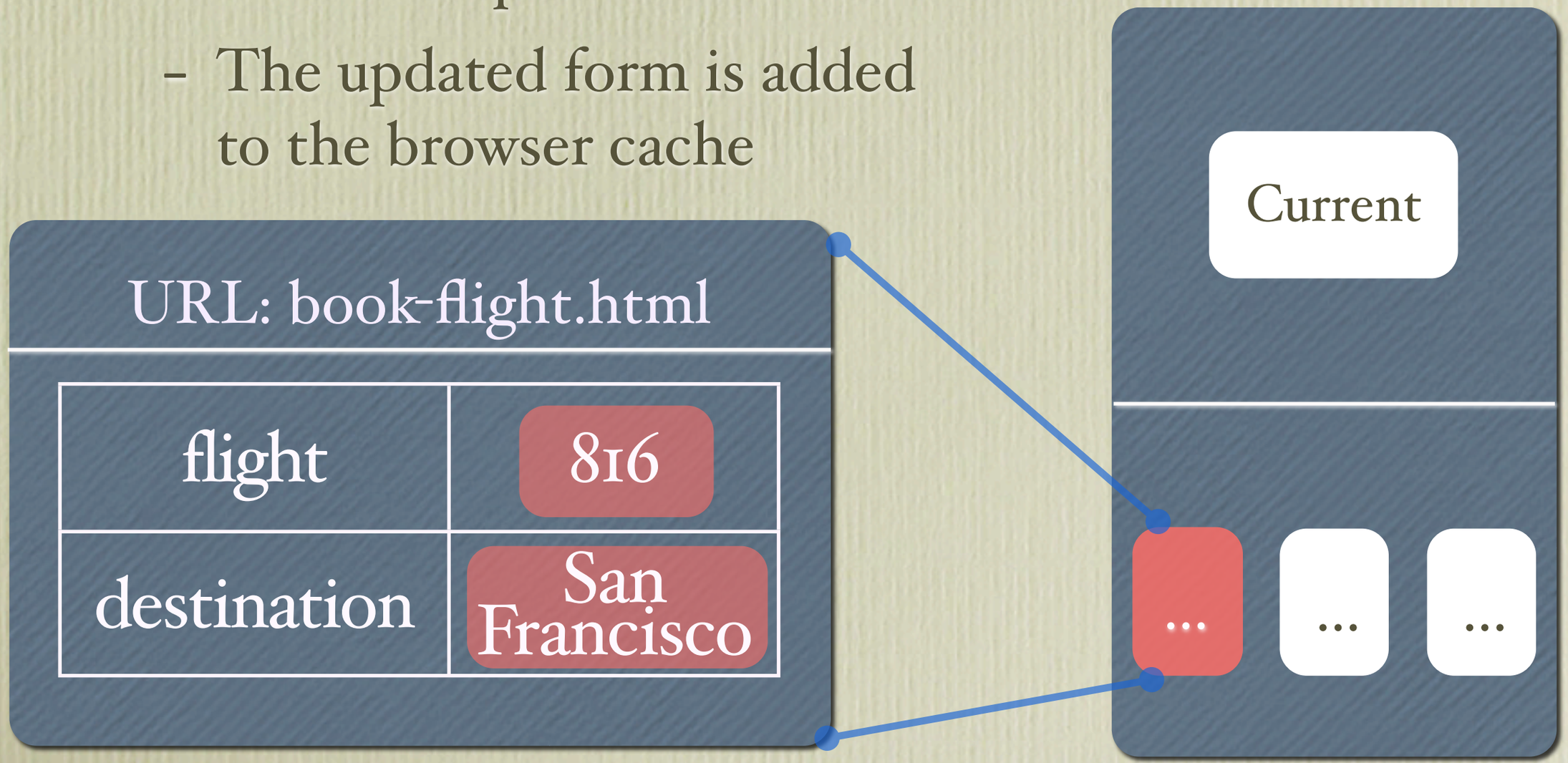
What happens, when...

- Users enter form data:
  - The key/value vector of the form is modified to store the updated value



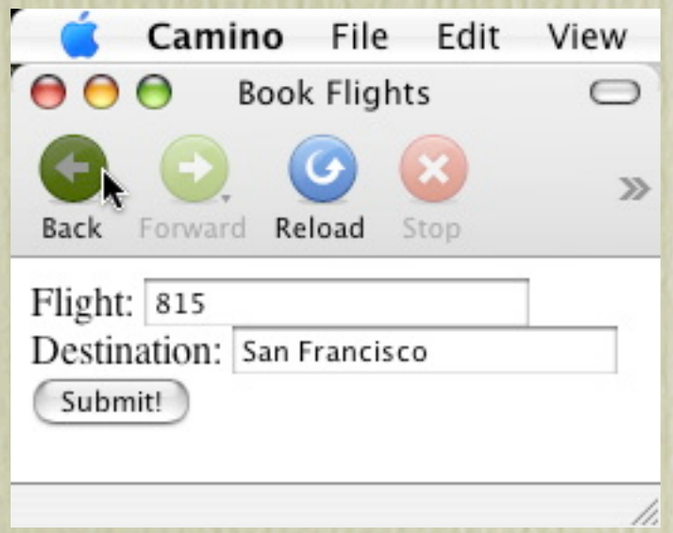
What happens, when...

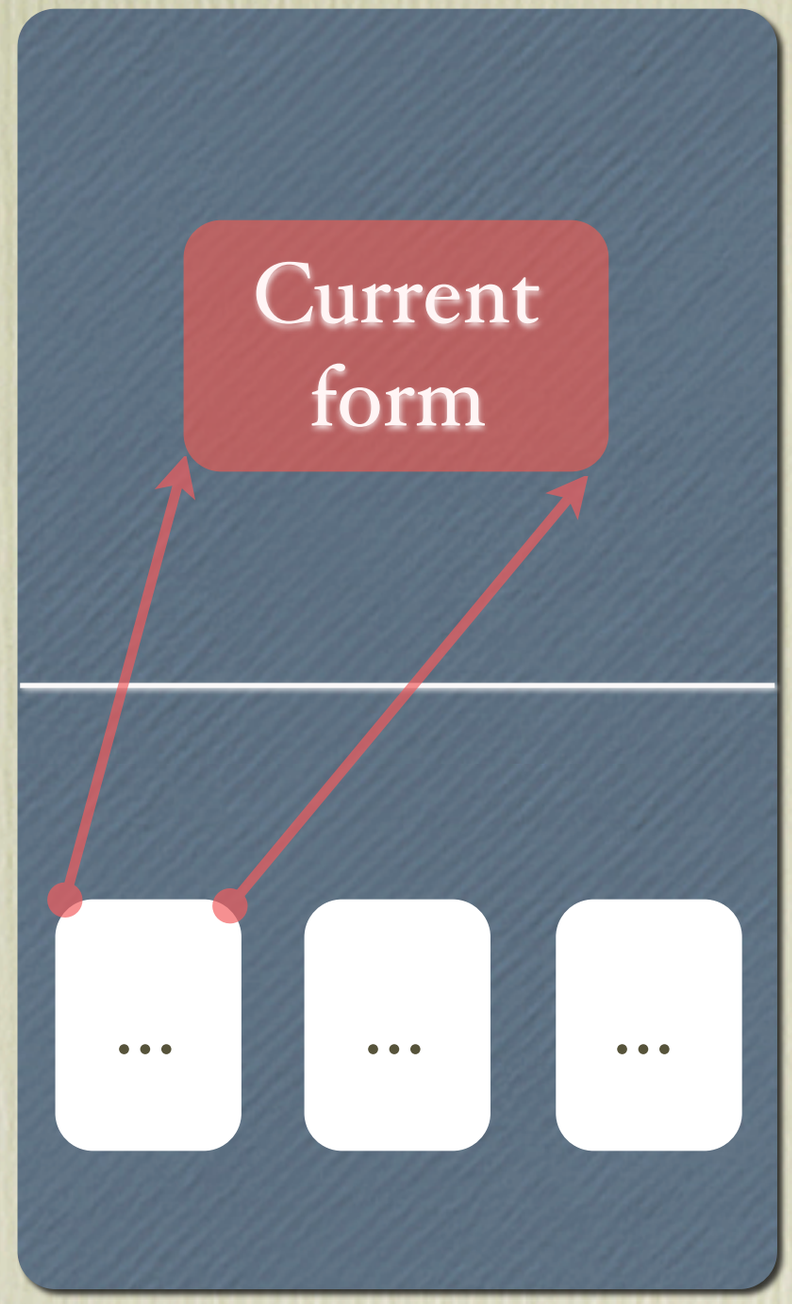
- Users enter form data:
  - The key/value vector of the form is modified to store the updated value
  - The updated form is added to the browser cache



What happens, when...

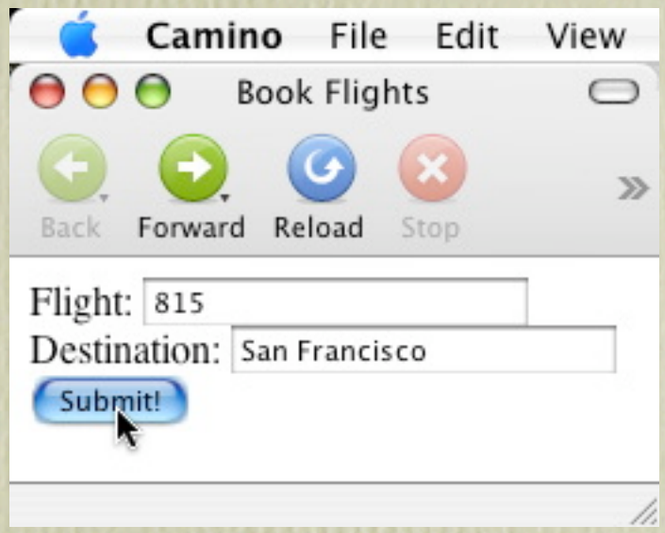
- Users switch to some form:
  - The new form is set as the client‘s „current page“ (but only if it‘s found in the cache)





What happens, when...

- Users submit a form?



Form Submissions

flight	815
dest.	SF
…	…

Scripts

Web Server

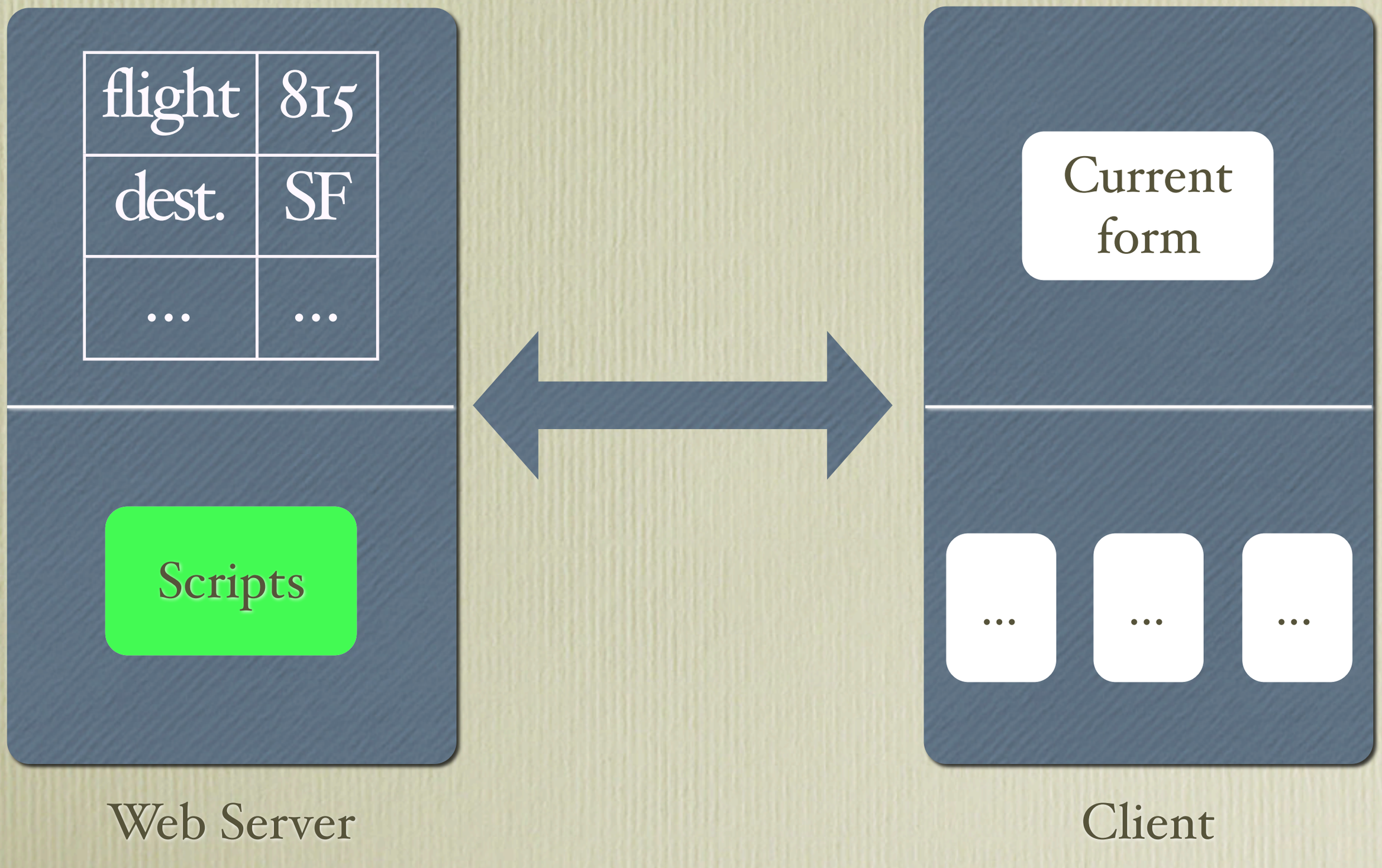
Current form

… … …

Client

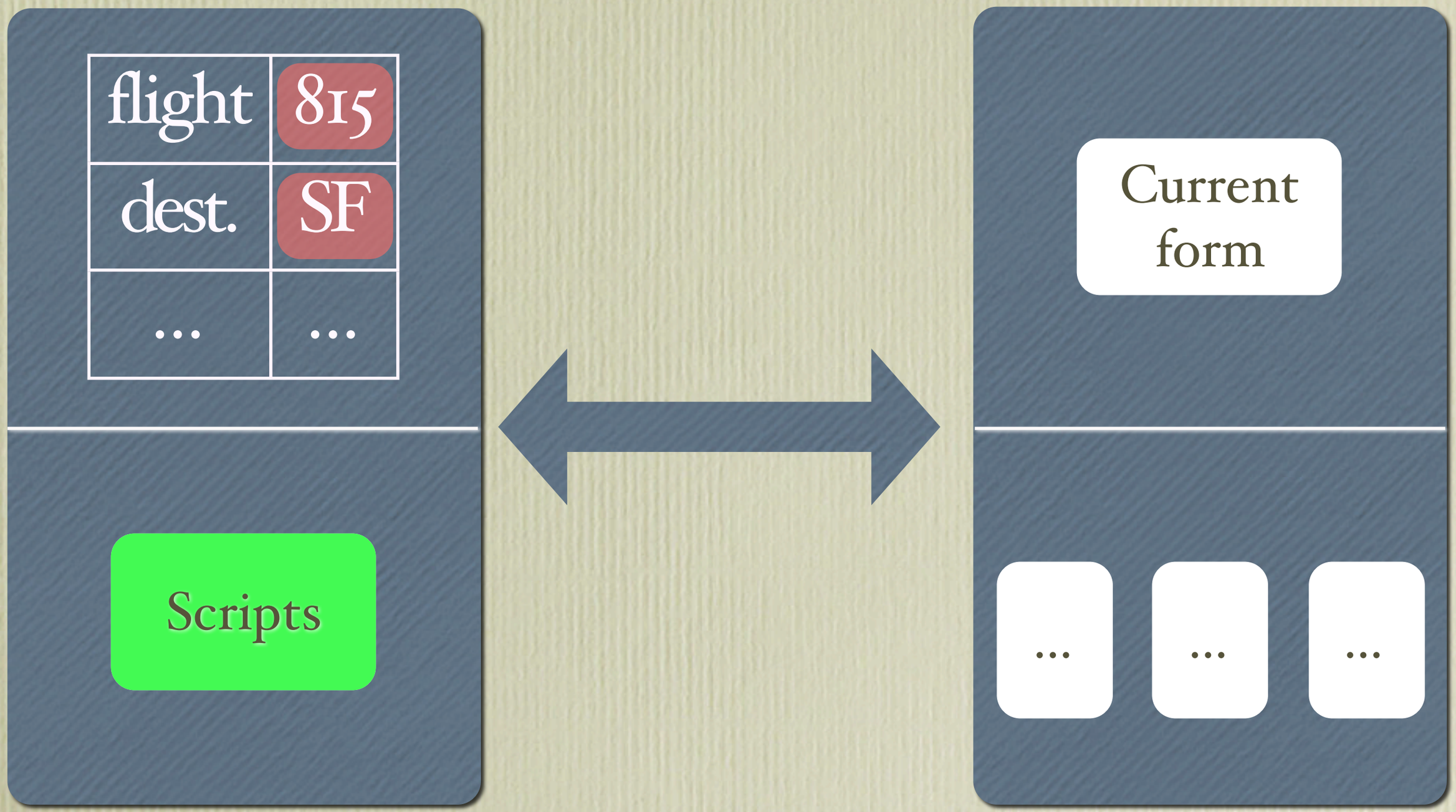
Form Submissions

- Server computes the new form



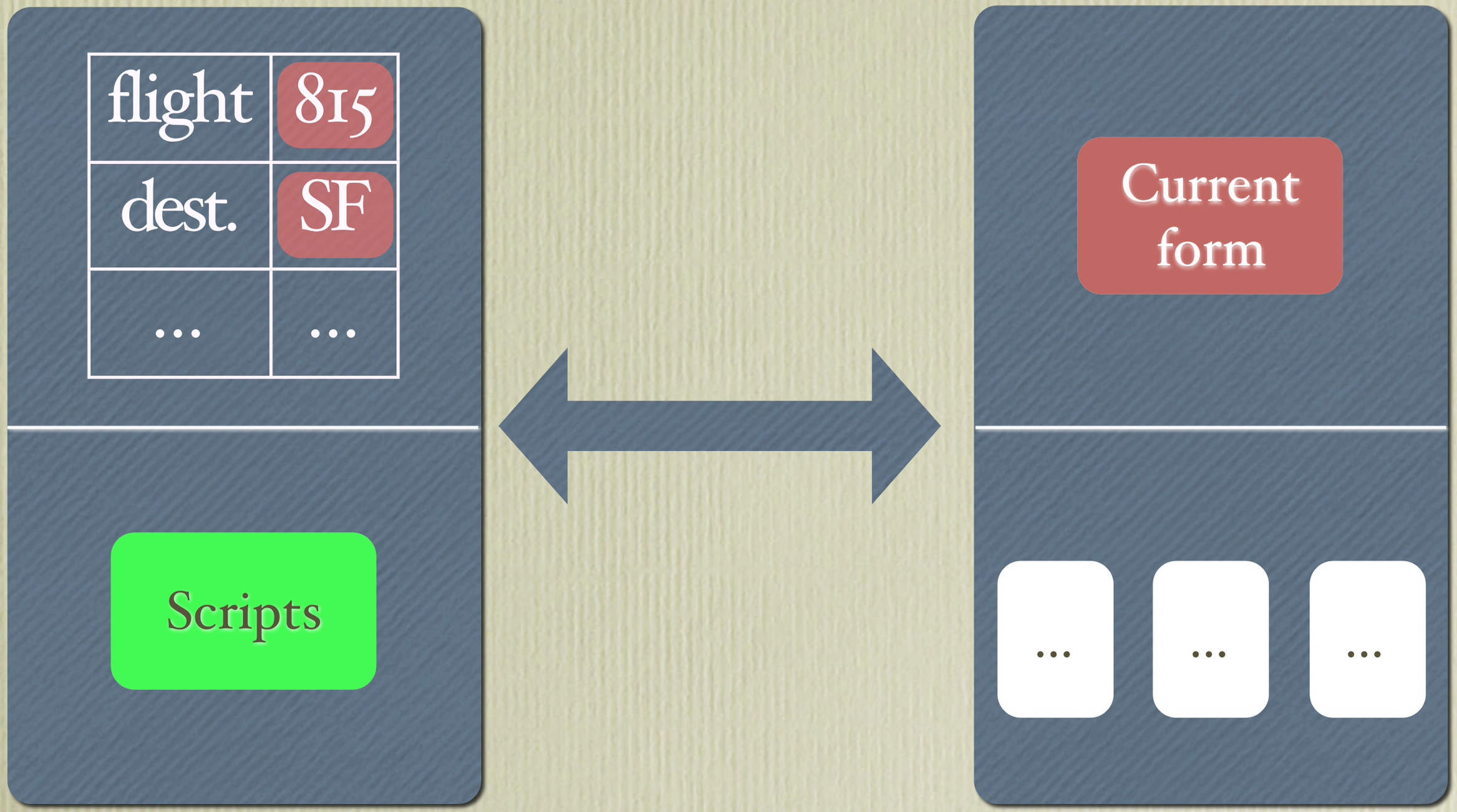
Form Submissions

- Server state (storage) is updated



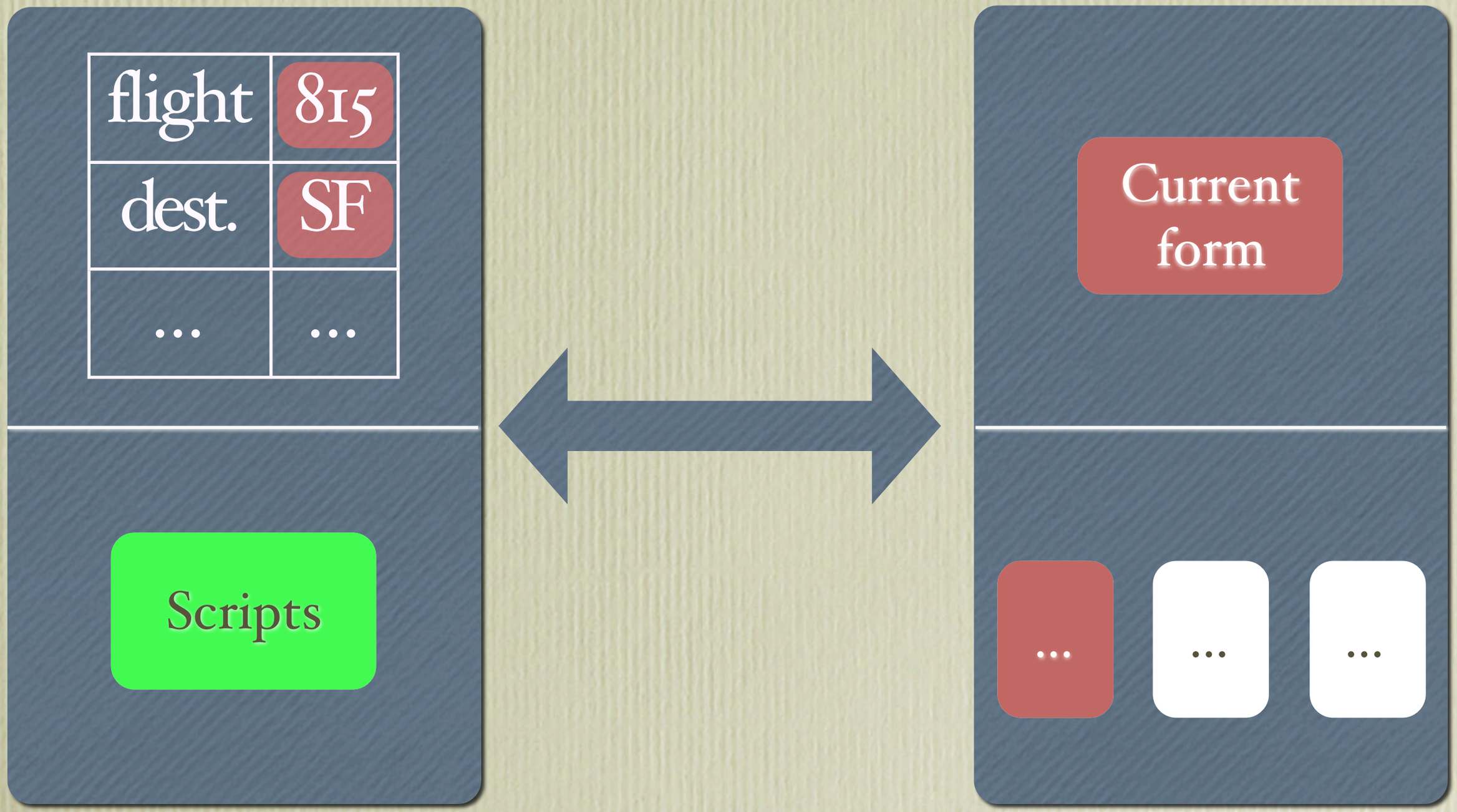
Form Submissions

- Client‘s „current page“ is set to the new form



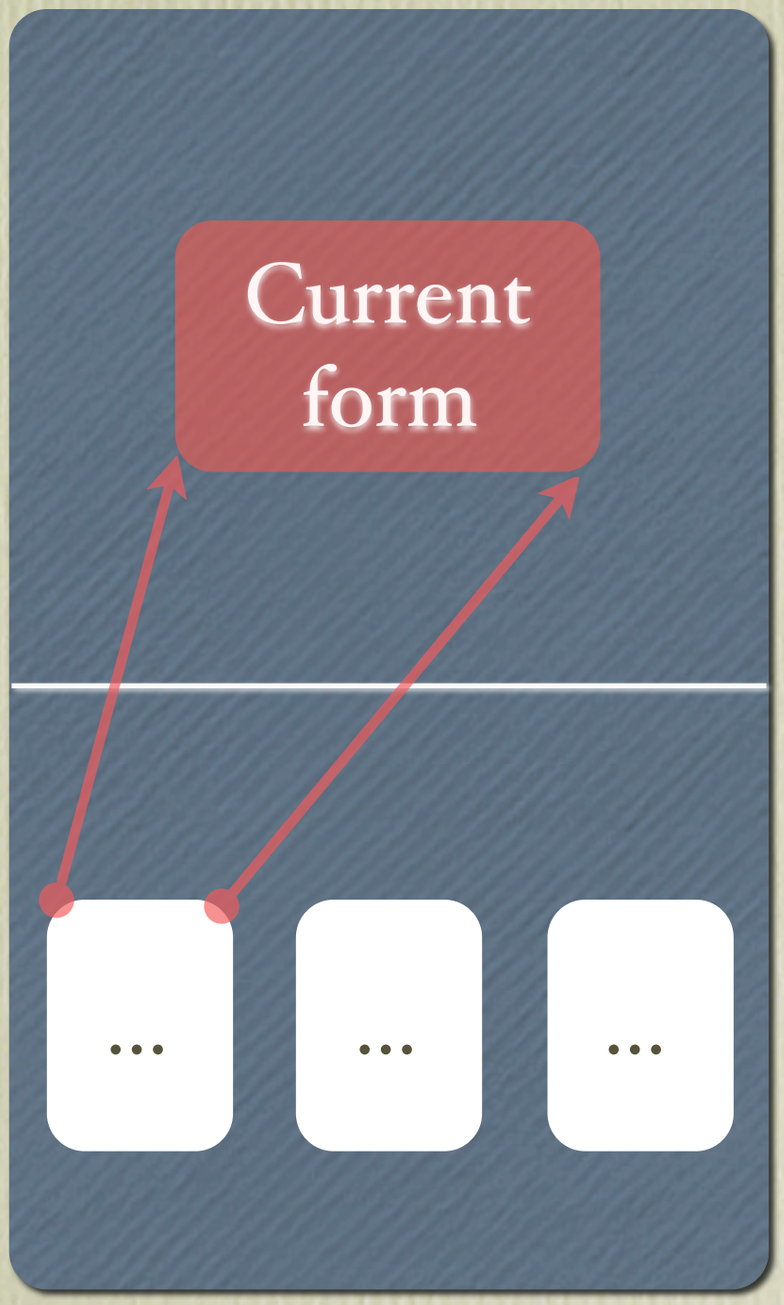
Form Submissions

- New form is added to client‘s browser cache



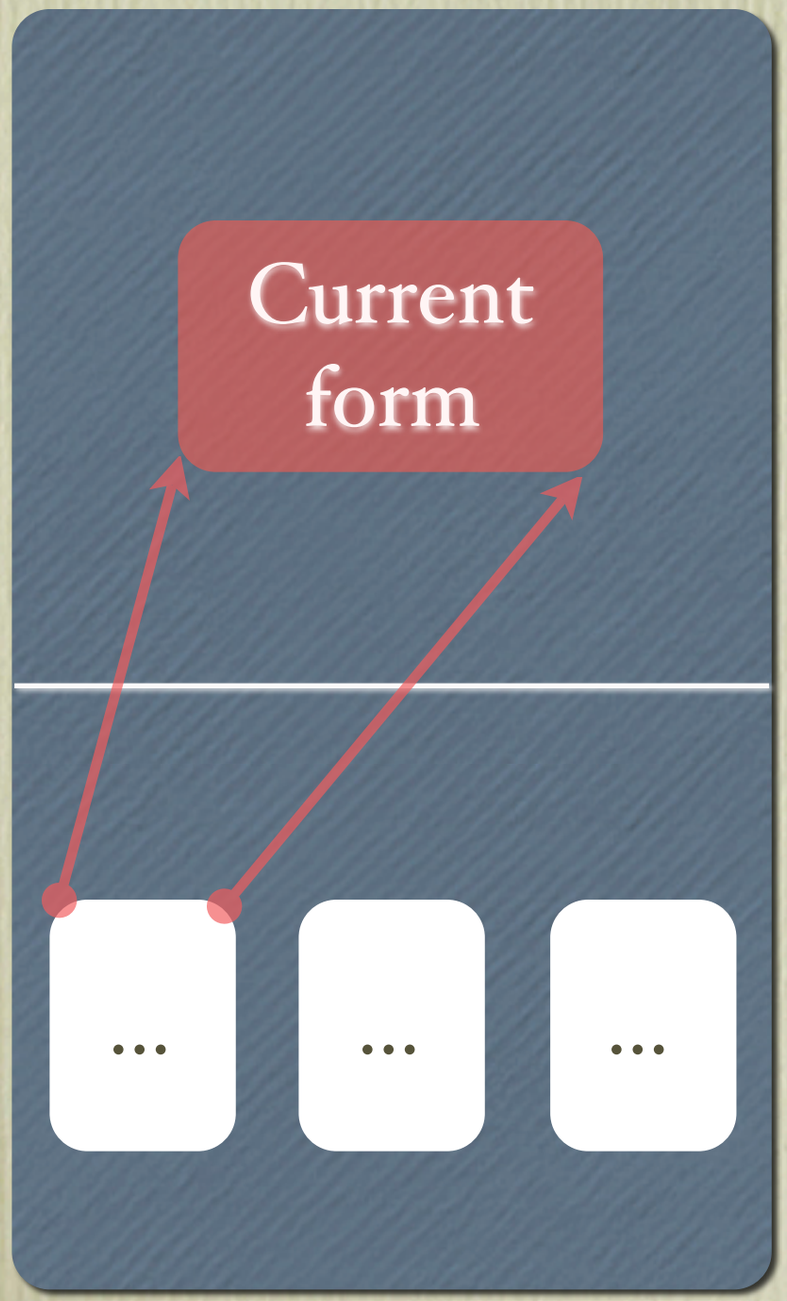
Attention, Mini-Test!

How does switching work again? Explain.



Attention, Mini-Test!

How does switching work again? Explain.



„Rewriting“ describes the transition directly and precisely:

$\langle s, \langle f_0, \vec{f} \rangle \rangle$

$\rightarrow \langle s, \langle f_1, \vec{f} \rangle \rangle$

where $f_1 \in \vec{f}$

Scripting Language

- Use identifiers, variables
- Create functions
- Apply functions
- Create new forms
- Extract values from forms (via keys)
- Basic I/O (Server storage read/write)

Scripting Language

- Use identifiers, variables
- Create functions
- Apply functions
- Create new forms
- Extract values from forms (via keys)
- Basic I/O (Server storage read/write)

Modelling the Bug

Show Flights

flight	-
dest.	-
…	…

Scripts

Web Pages

Web Server

Modelling the Bug

Show Flights

Flight 1 Details

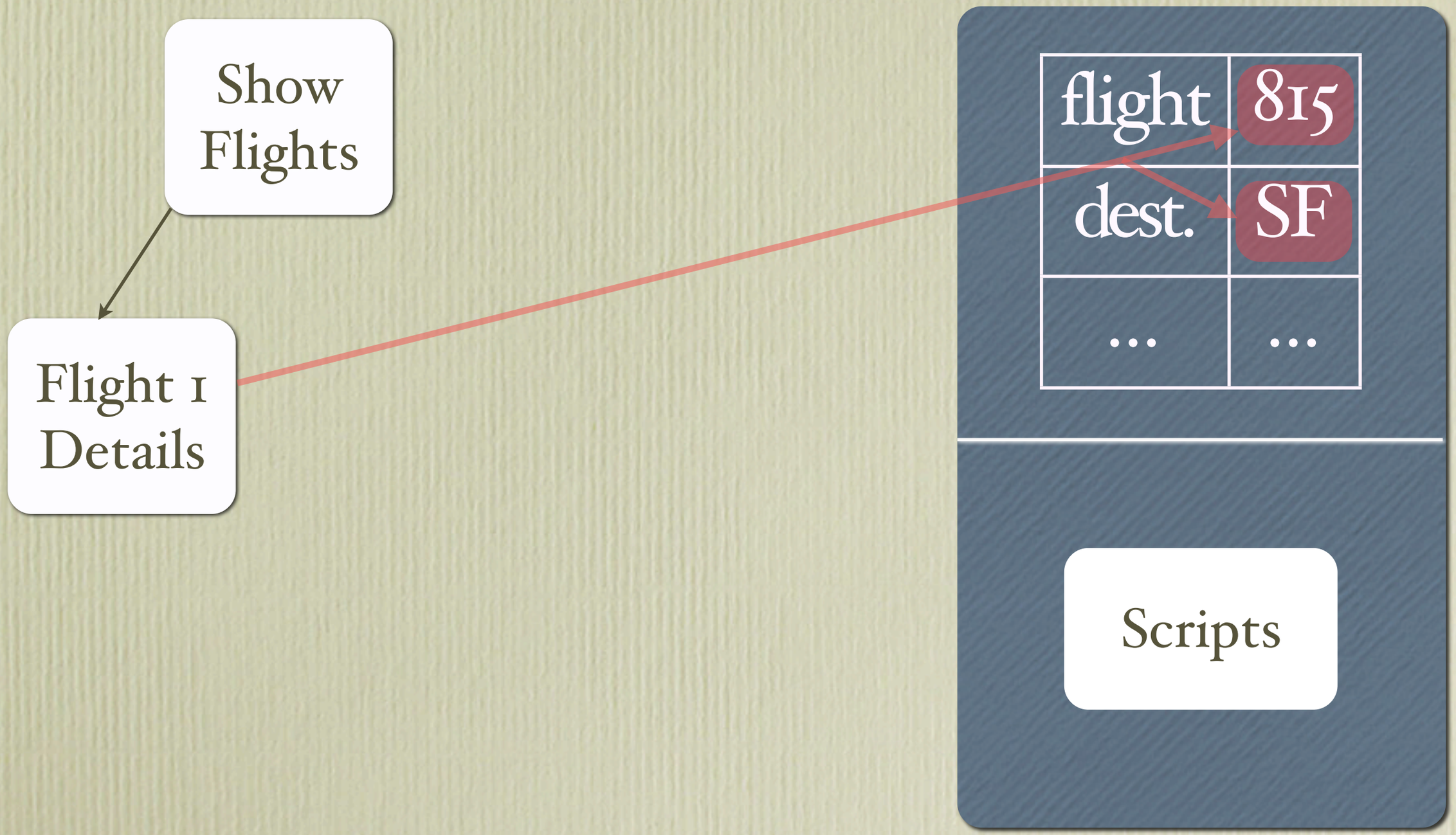
flight	-
dest.	-
...	...

Scripts

Web Pages

Web Server

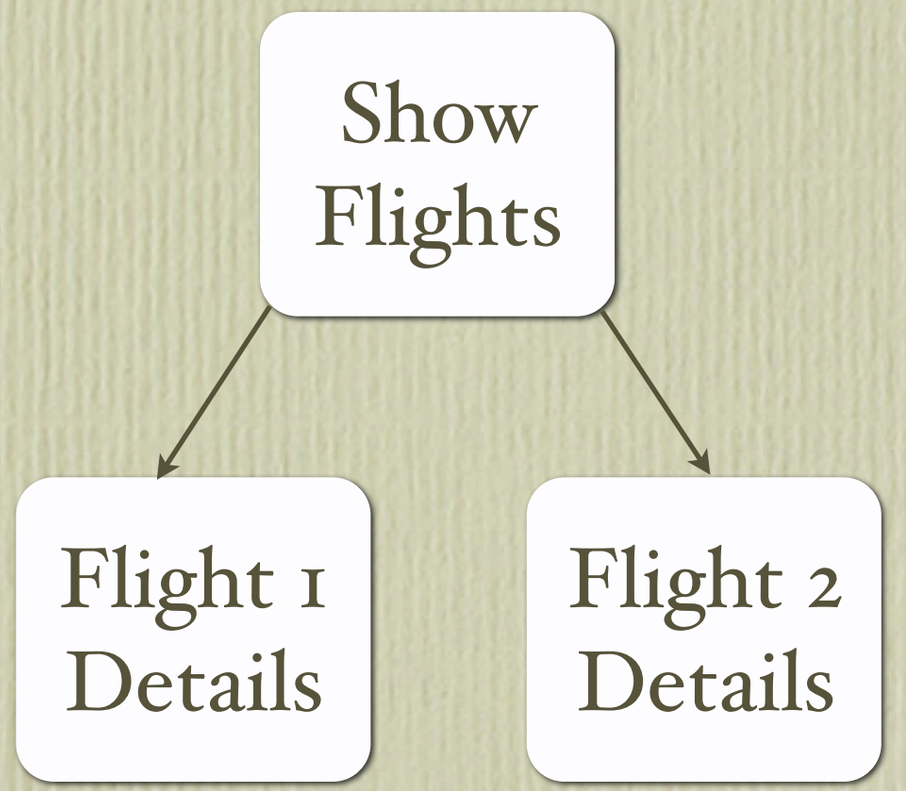
Modelling the Bug



Web Pages

Web Server

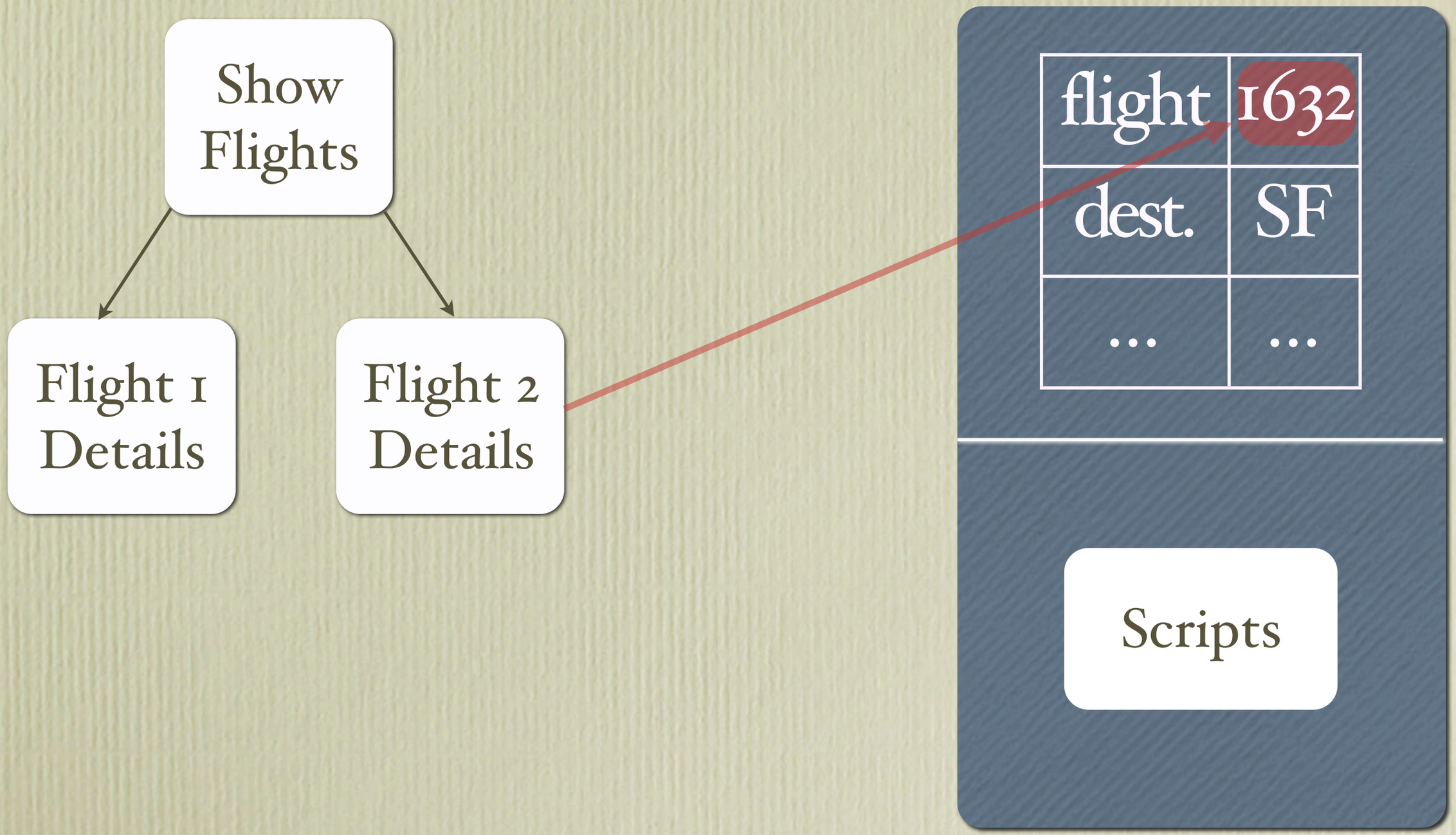
Modelling the Bug



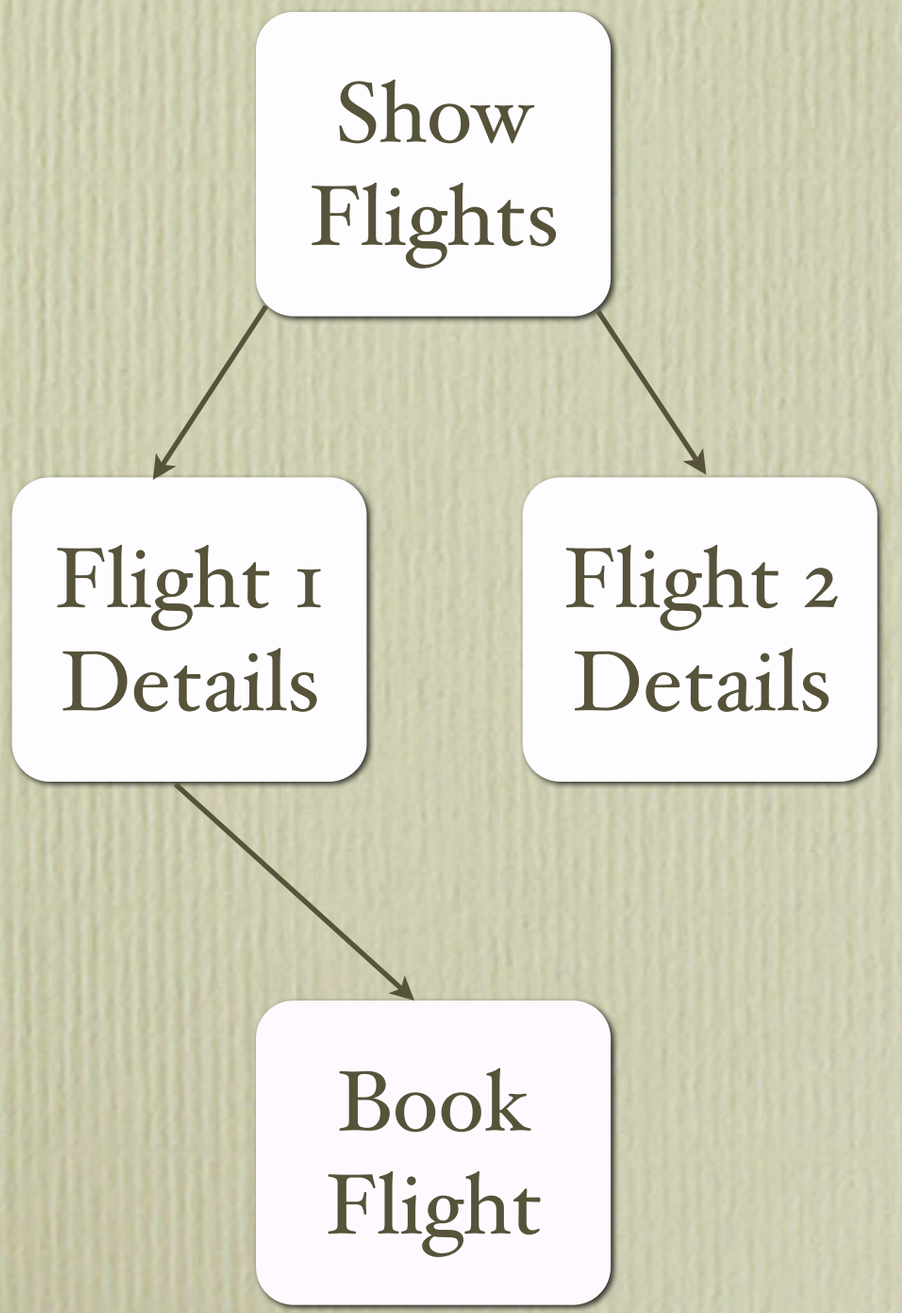
Web Pages

Web Server

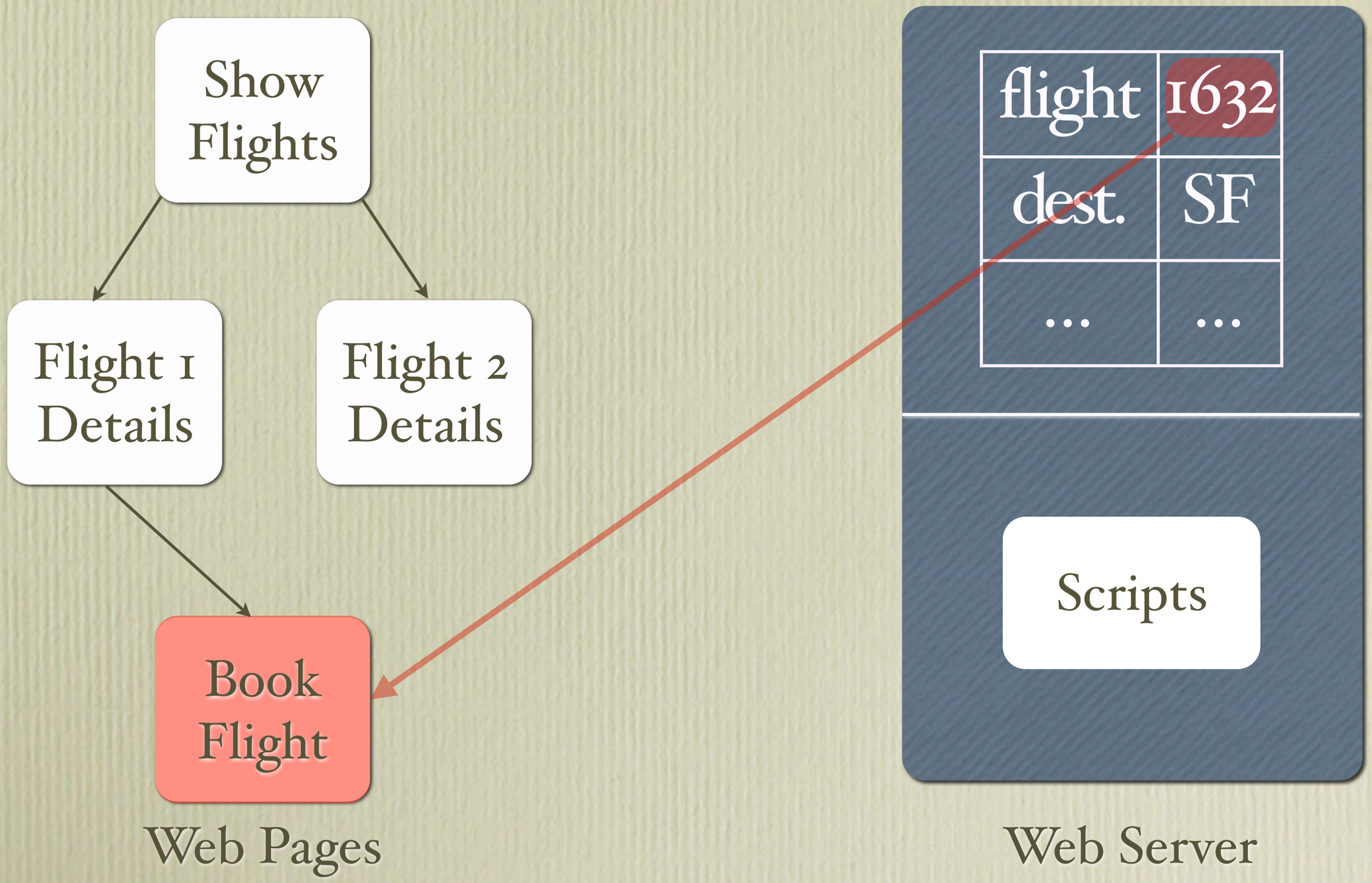
Modelling the Bug



Modelling the Bug



Modelling the Bug



Explaining the Bug

- Obviously, submitting „outdated“ forms causes undesired behaviour
- The HTTP Observer Problem: Server cannot „push“ updates to the client (as in MVC)

➡ At least produce warnings when detecting outdated requests

Detecting outdated requests

- Server needs a notion of time:

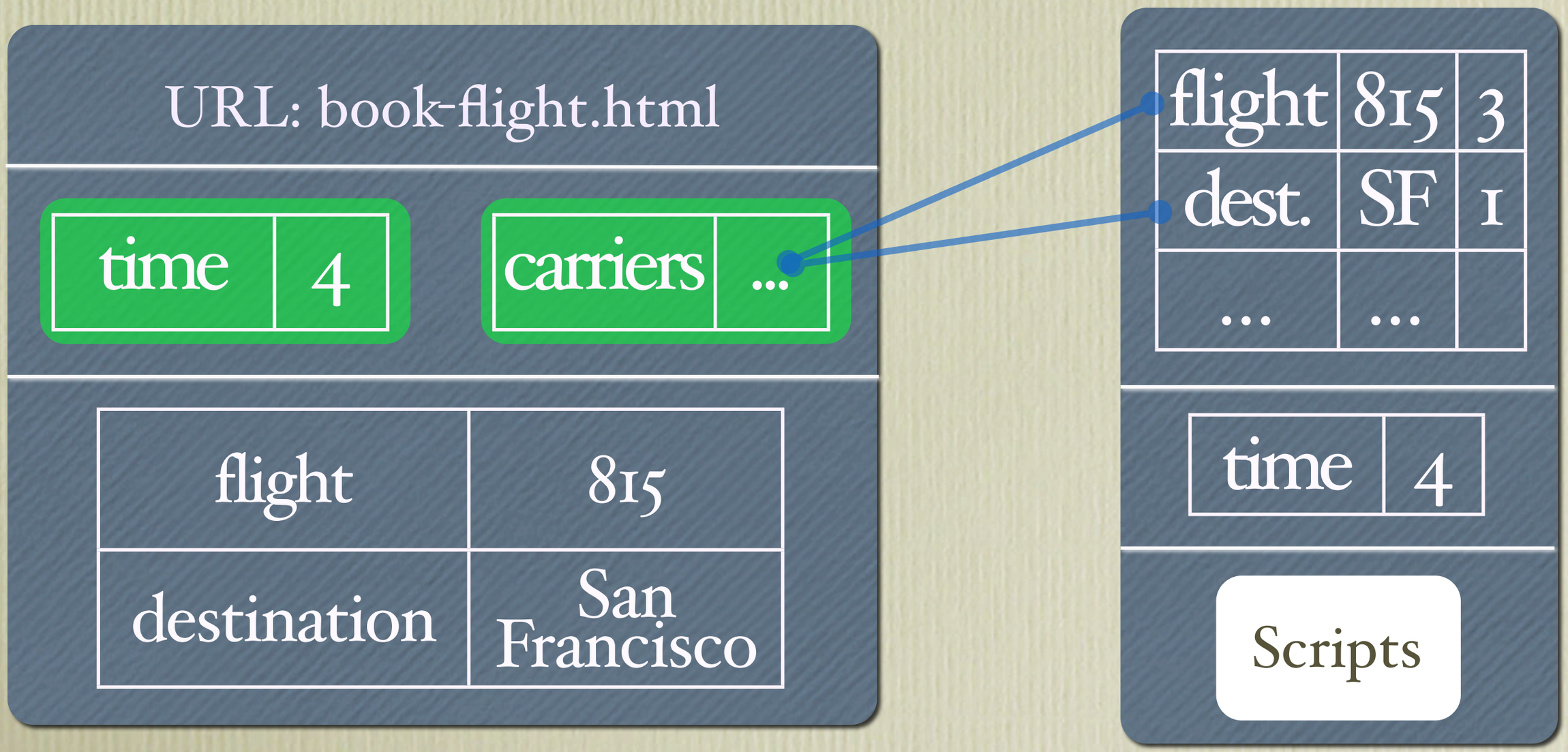
  ➡ Model as number of submits

- Storage records time of last write for each field

Detecting outdated requests

- Introduce „carrier sets“ into forms:
  All locations accessed by this script

- Each form stores its creation time



Detecting outdated requests

- Whenever a form is submitted, check its carrier set against current storage state and compare time stamps
- The carrier set represents the assumptions the script made while working
- If any location from this set was overwritten, script assumptions may have been violated

Thank you!

- We have built a comprehensive, yet simple model of web interactions
- Three basic semantic rules suffice to describe all possible user actions:
  - „switch“
  - „fill-out“
  - „submit“
- Any questions?

References

- Shriram Krishnamurti, Robert Bruce Findler, Paul Graunke, Matthias Felleisen:
„Modeling Web Interactions and Errors“ (2004)

- Daniel R. Licata, Shriram Krishnamurthi:
„Verifying Interactive Web Programs“ (2005)

Addendum: Fun with Types

- Make forms typed!
- Enables static checks for common bugs, like trying to access form data that never got submitted
- Also enables us to give some other safety guarantees
- But: How to keep track of types in a dynamic setting?

Incremental Type Checking

- Uses constraints along with regular type judgements
- Constraints are introduced by creating forms:
  - The successor url of any form must contain a program that takes as input exactly the data (type) of that form
- Otherwise, forms behave essentially like records

Consistency

- Consistency is achieved by checking that all types registered for some form at a particular URL are equivalent
- Since type constraints can be introduced by the regular type system as well as by additional constraints, this is not always the case
- If the types are not consistent, refuse to execute the script