Winning the Application Server Arms Race

Using Smalltalk to Redefine Web Development

Avi Bryant

Web apps: why bother?

"I'm more of the mind that HTML based apps suck."

— James Robertson

Web apps: why bother?

"One of the reasons to use Lisp in writing Web-based applications is that you *can* use Lisp. When you're writing software that is only going to run on your own servers, you can use whatever language you want."

— Paul Graham

What is a web app?

"A collection of functions that take HTTP requests as input and produce HTTP responses as output."

What is a web app?

/foo

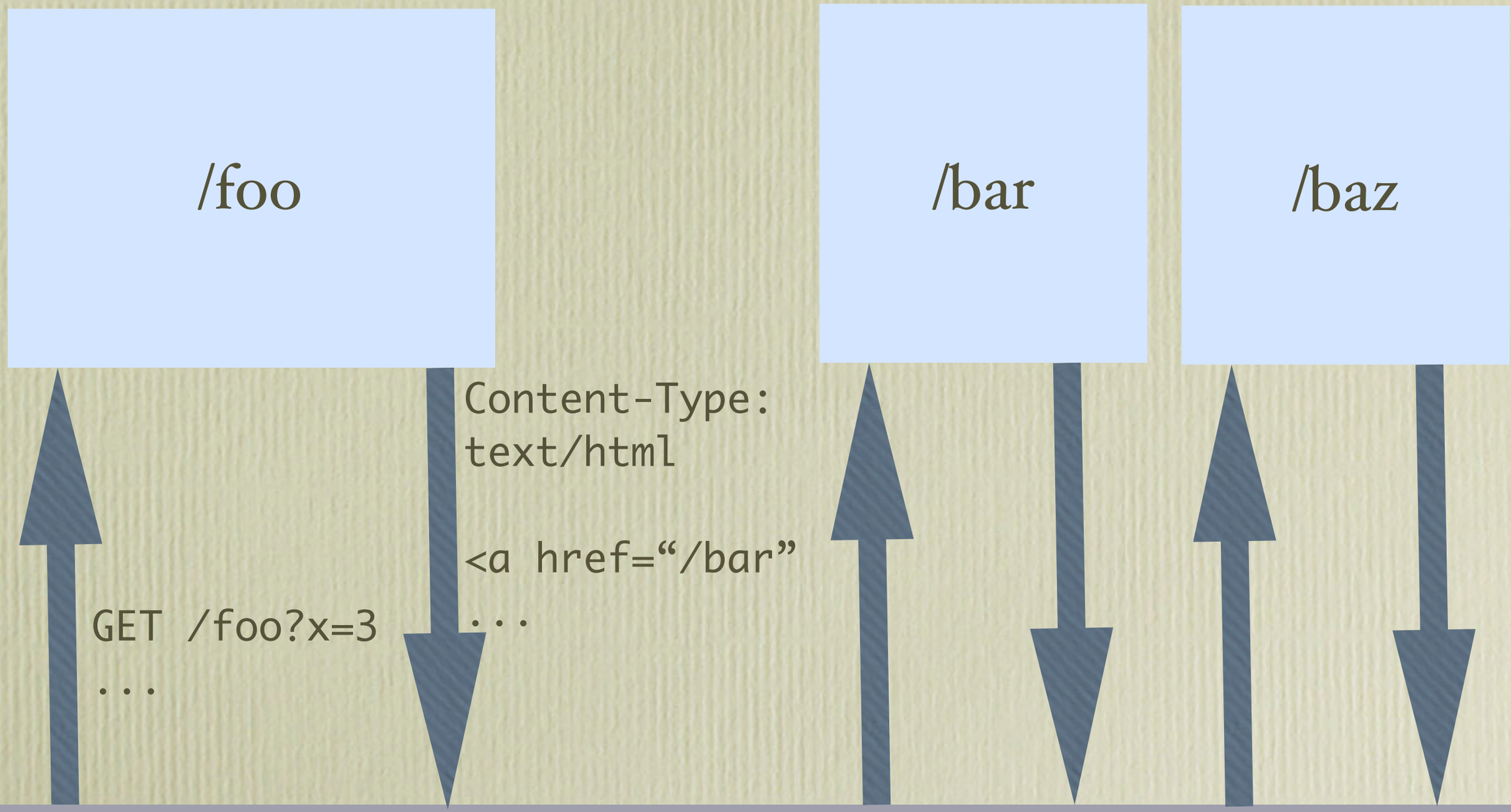
Content-Type:
text/html

...

GET /foo?x=3
...

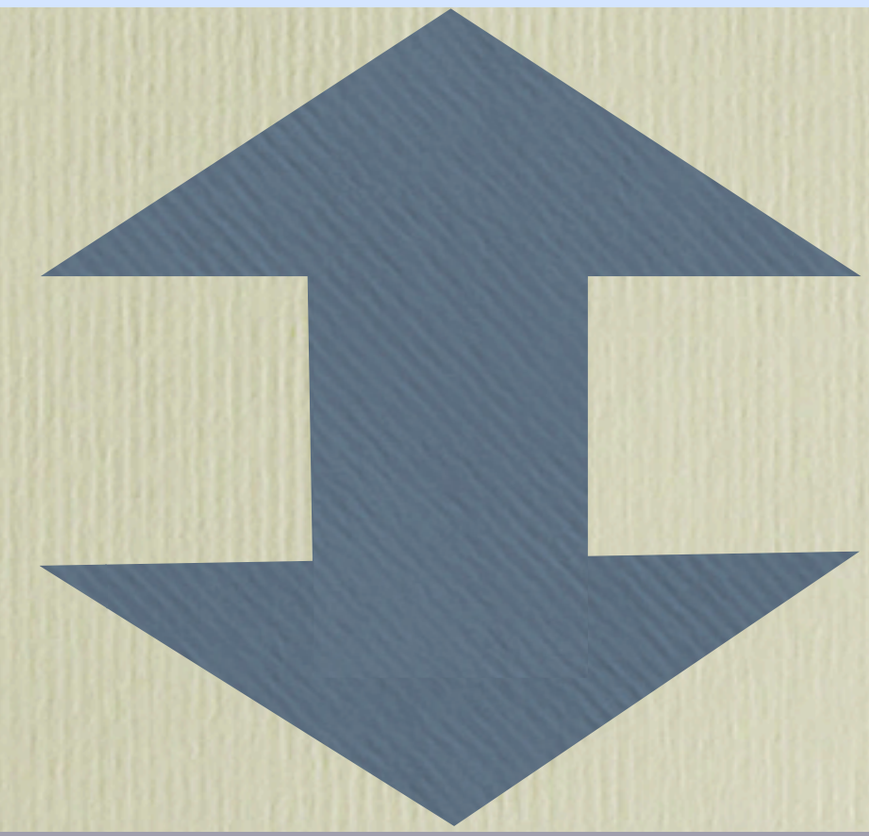
User with Web Browser

What is a web app?



Client/server app

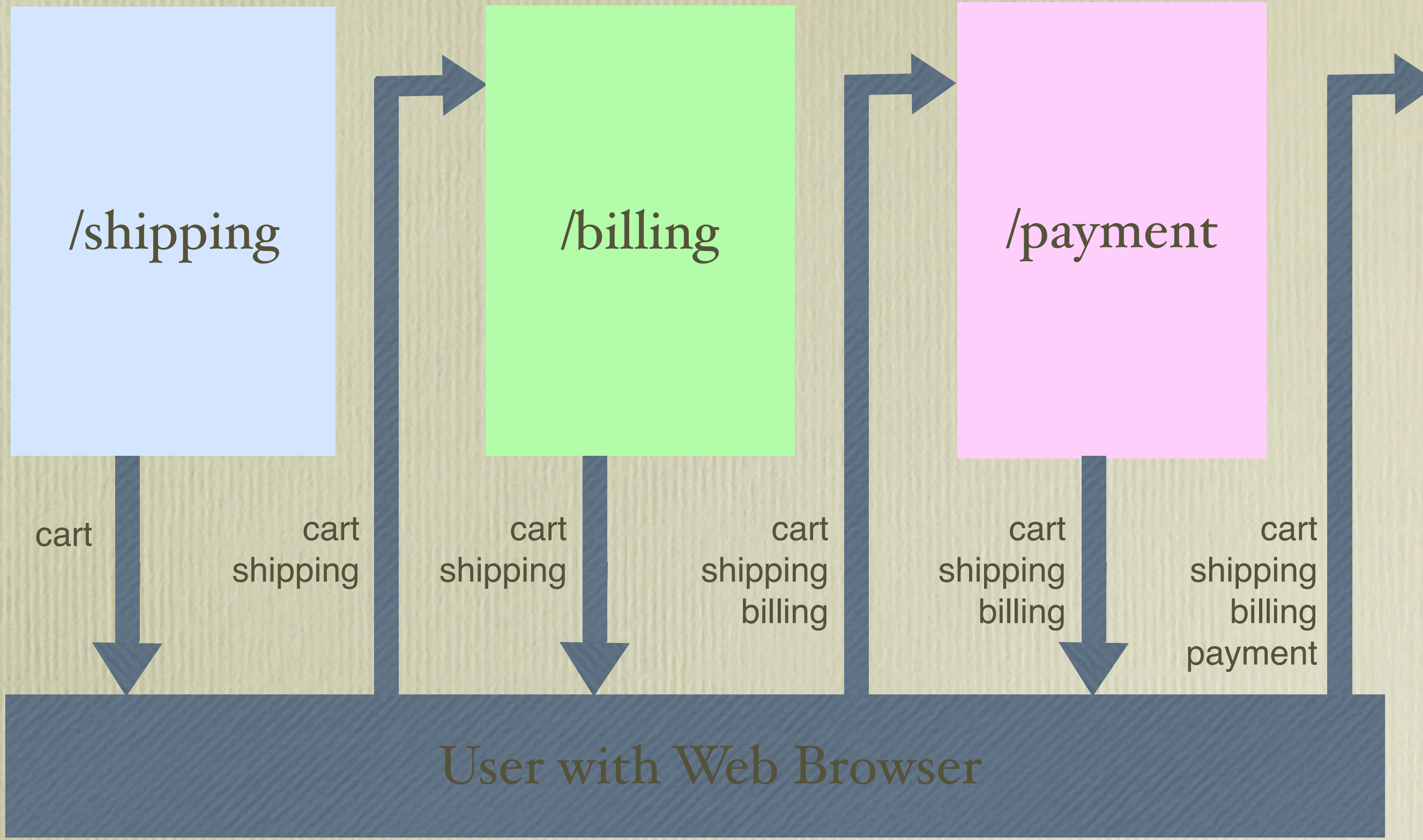
GemStone Server

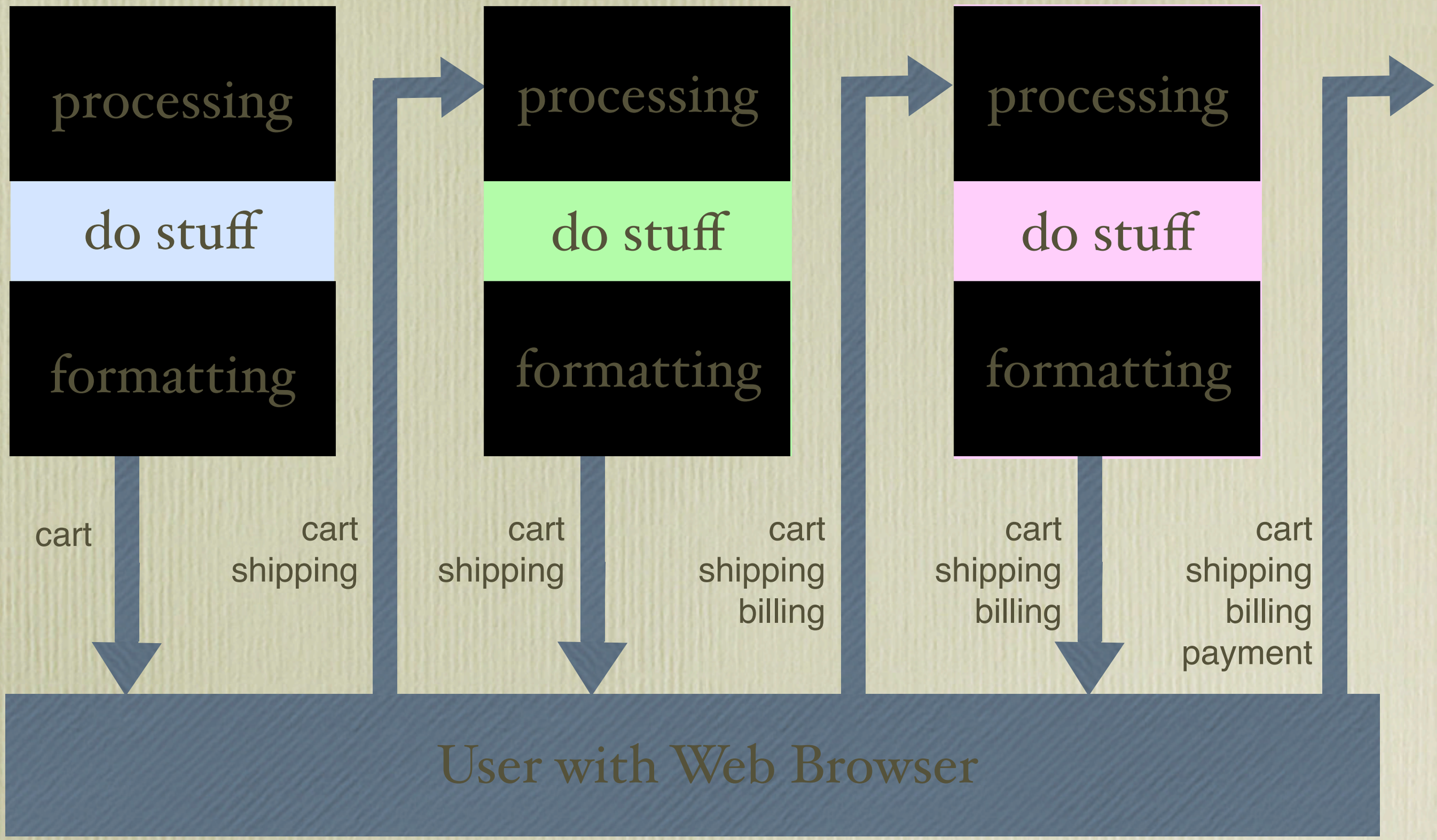


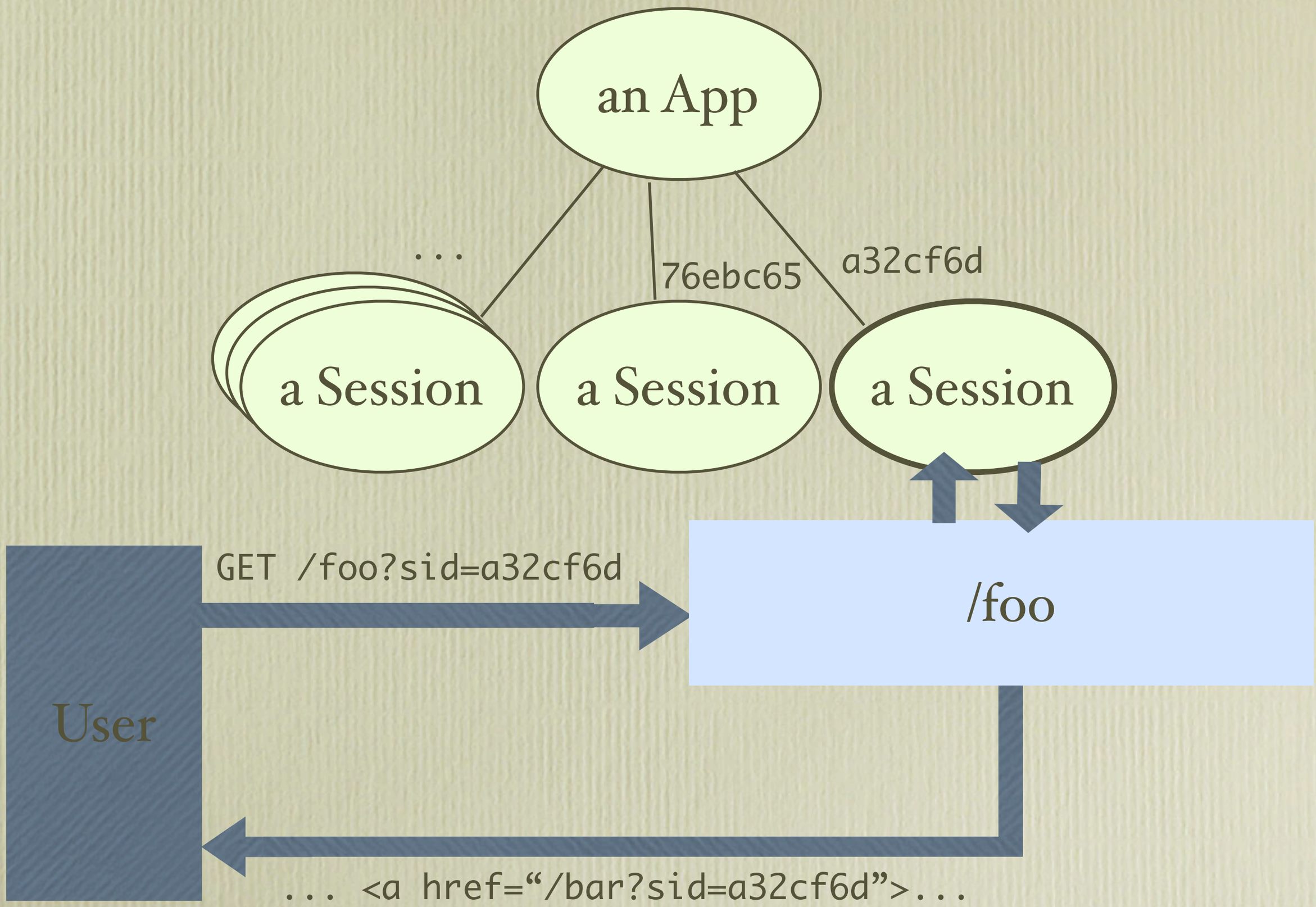
VisualWorks Client

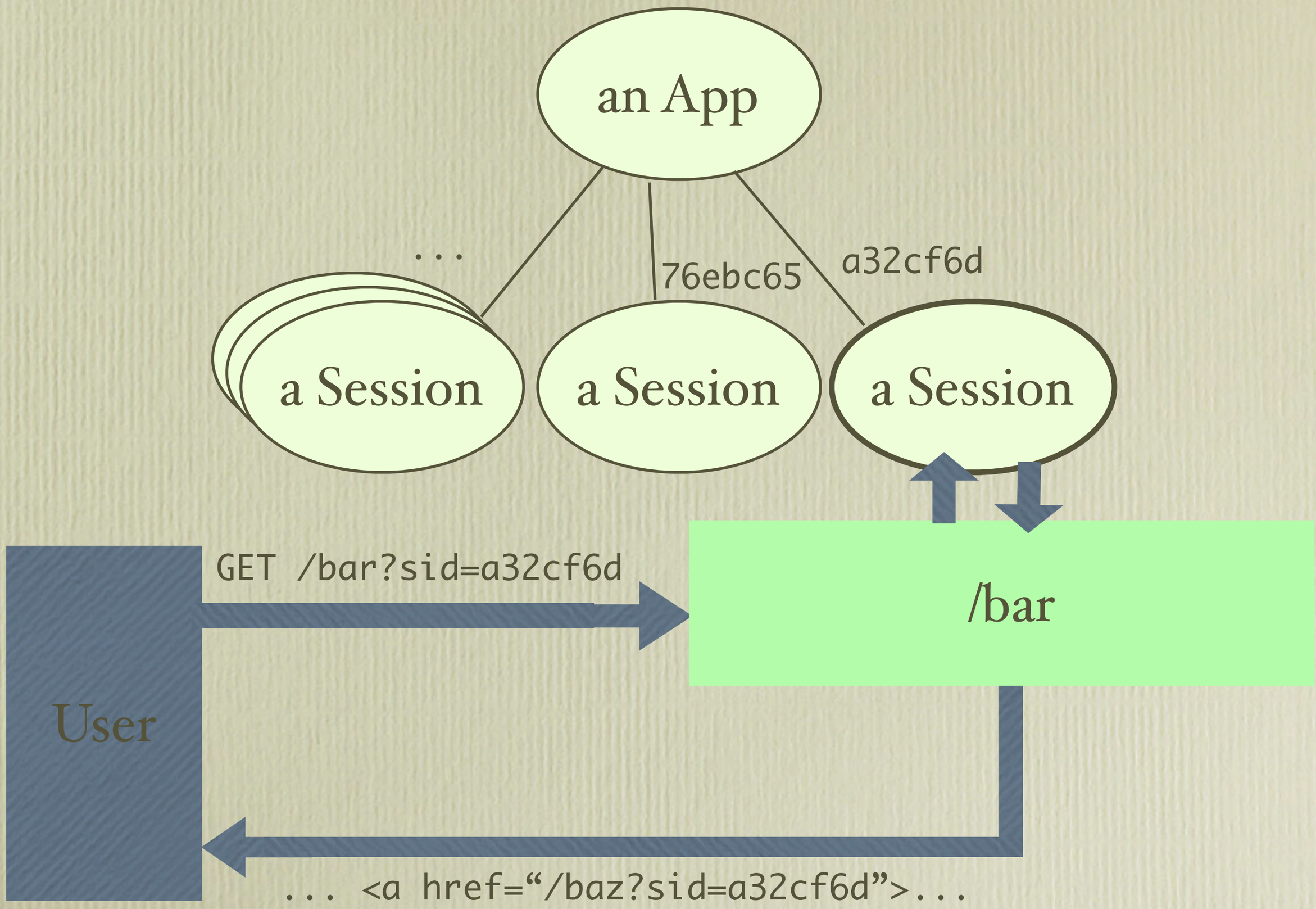

Get Shipping Address
Get Billing Address
Get Payment Info
Show Confirmation

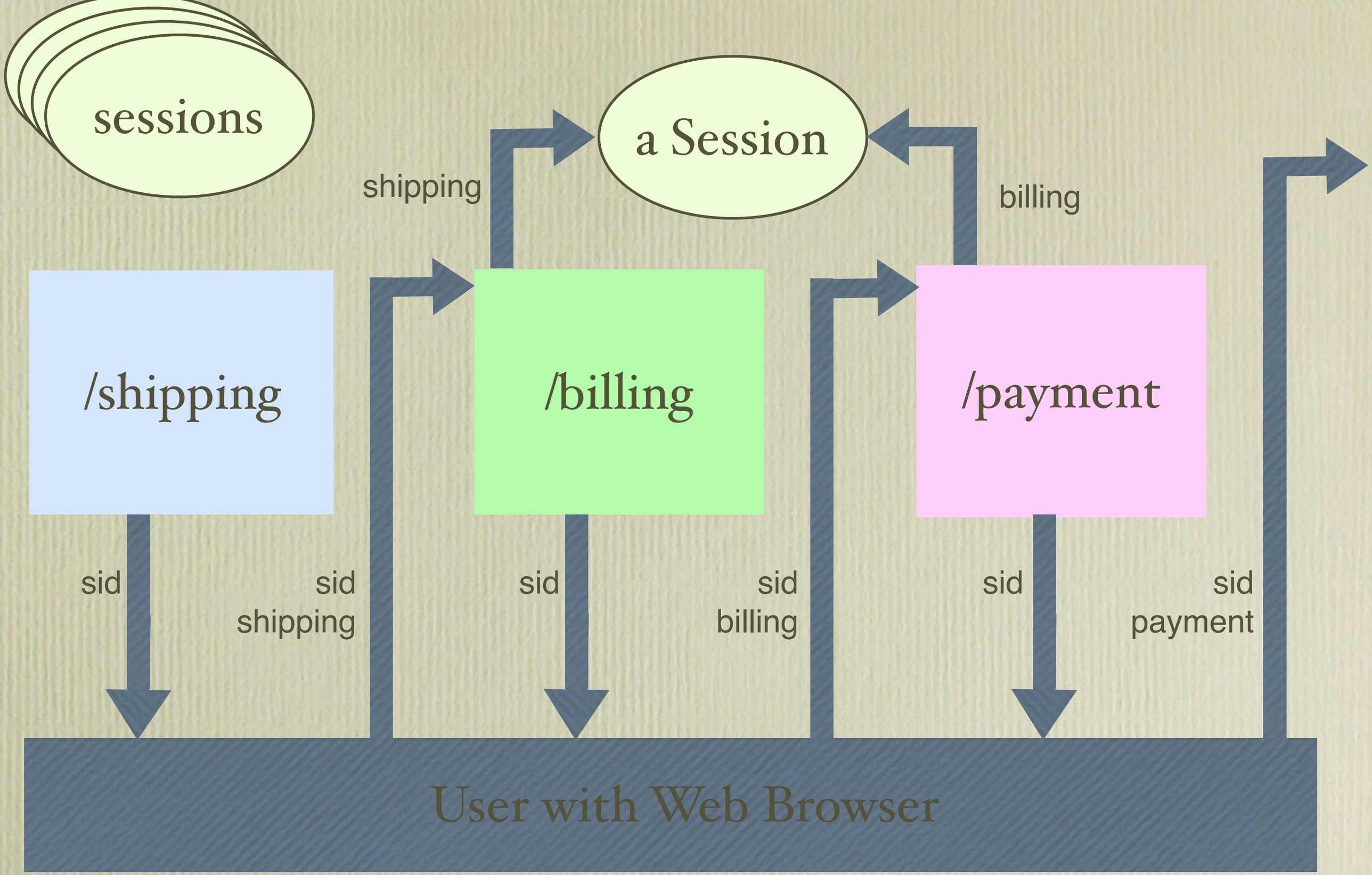

Get Shipping Address
Get Billing Address
Get Payment Info
Show Confirmation
Cart info
Cart info
Shipping info
Cart info
Shipping info
Billing info
Cart info
Shipping info
Billing info
Payment info


/shipping
/billing
/payment
cart
cart
shipping
cart
shipping
cart
shipping
billing
cart
shipping
billing
cart
shipping
billing
payment
User with Web Browser

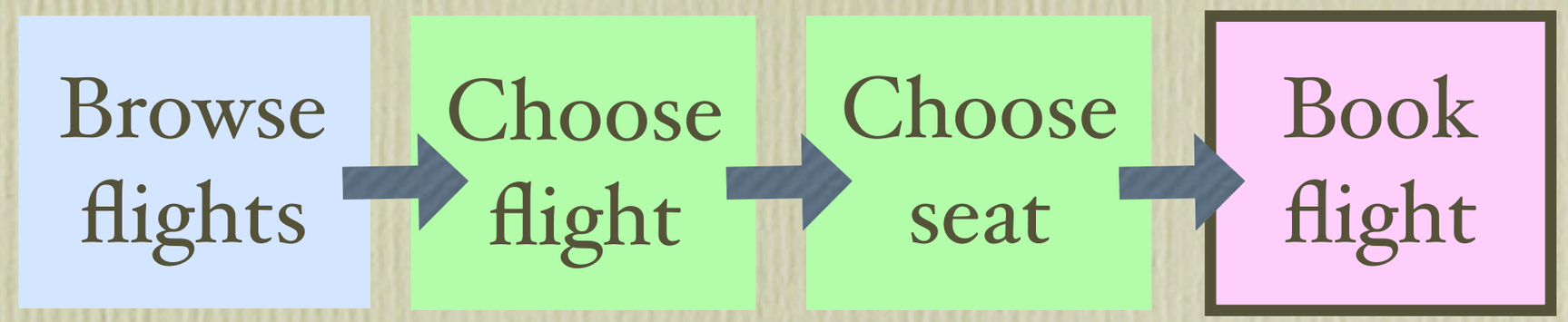

processing
do stuff
formatting
cart
cart
shipping
processing
do stuff
formatting
cart
shipping
cart
shipping
billing
processing
do stuff
formatting
cart
shipping
billing
cart
shipping
billing
payment
User with Web Browser


an App
. . .
76ebc65
a32cf6d
a Session
a Session
a Session
/foo
User
GET /foo?sid=a32cf6d
... <a href="/bar?sid=a32cf6d">...

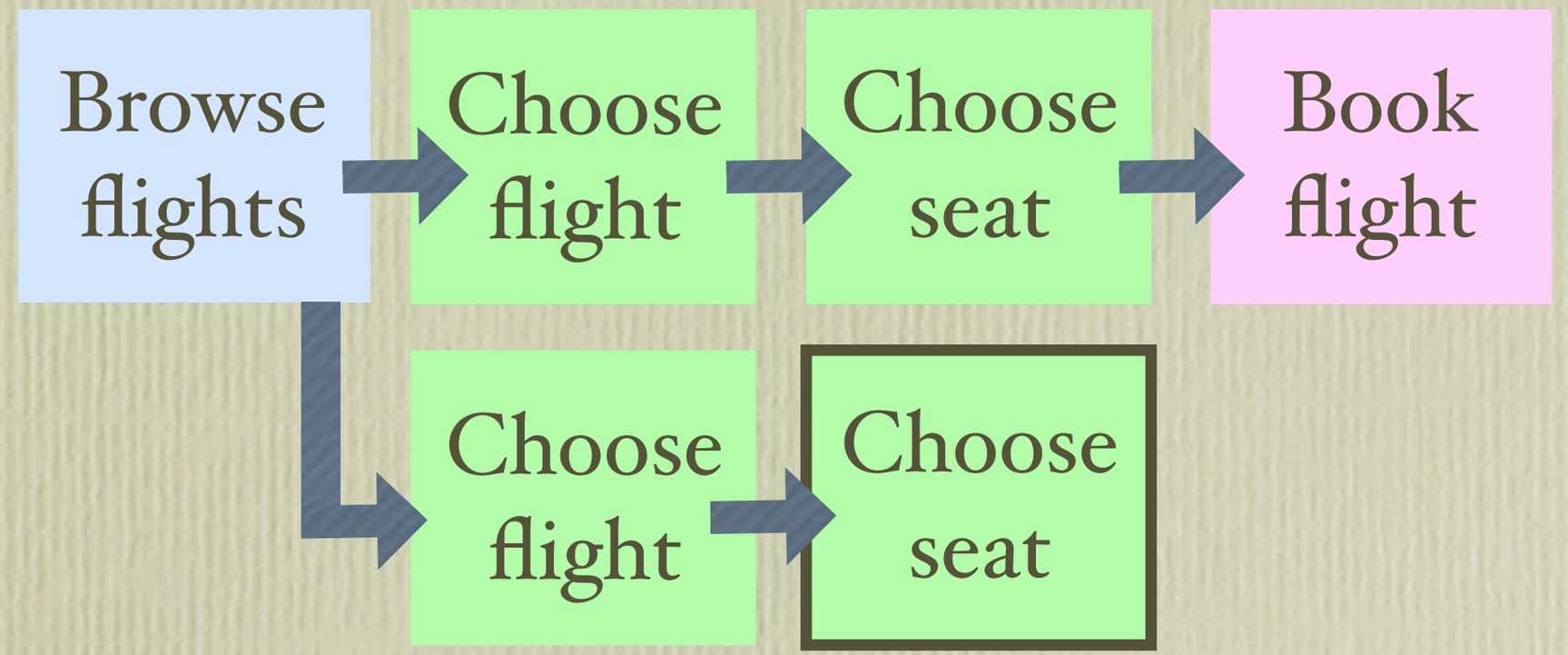

an App
...
76ebc65
a32cf6d
a Session
a Session
a Session
/bar
User
GET /bar?sid=a32cf6d
... <a href="/baz?sid=a32cf6d">...


sessions
a Session
shipping
billing
/shipping
/billing
/payment
sid
sid
shipping
sid
sid
billing
sid
sid
payment
User with Web Browser

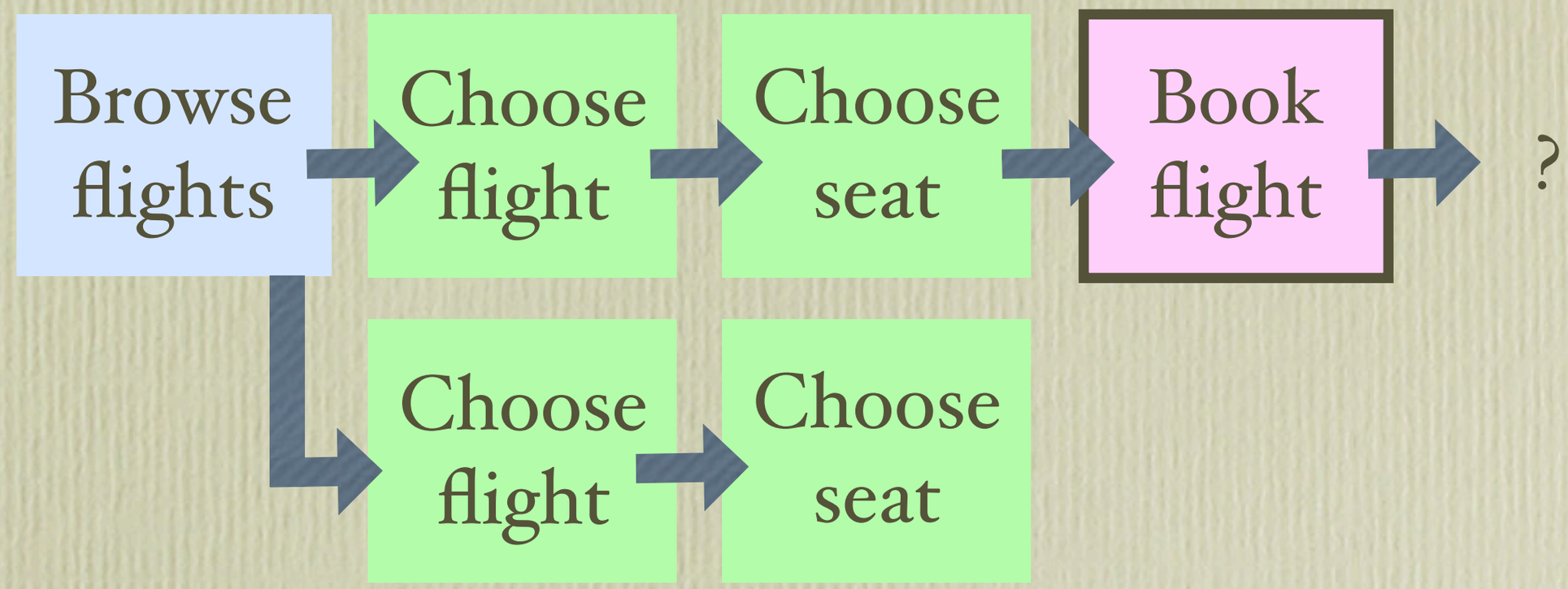
Why global session state is evil

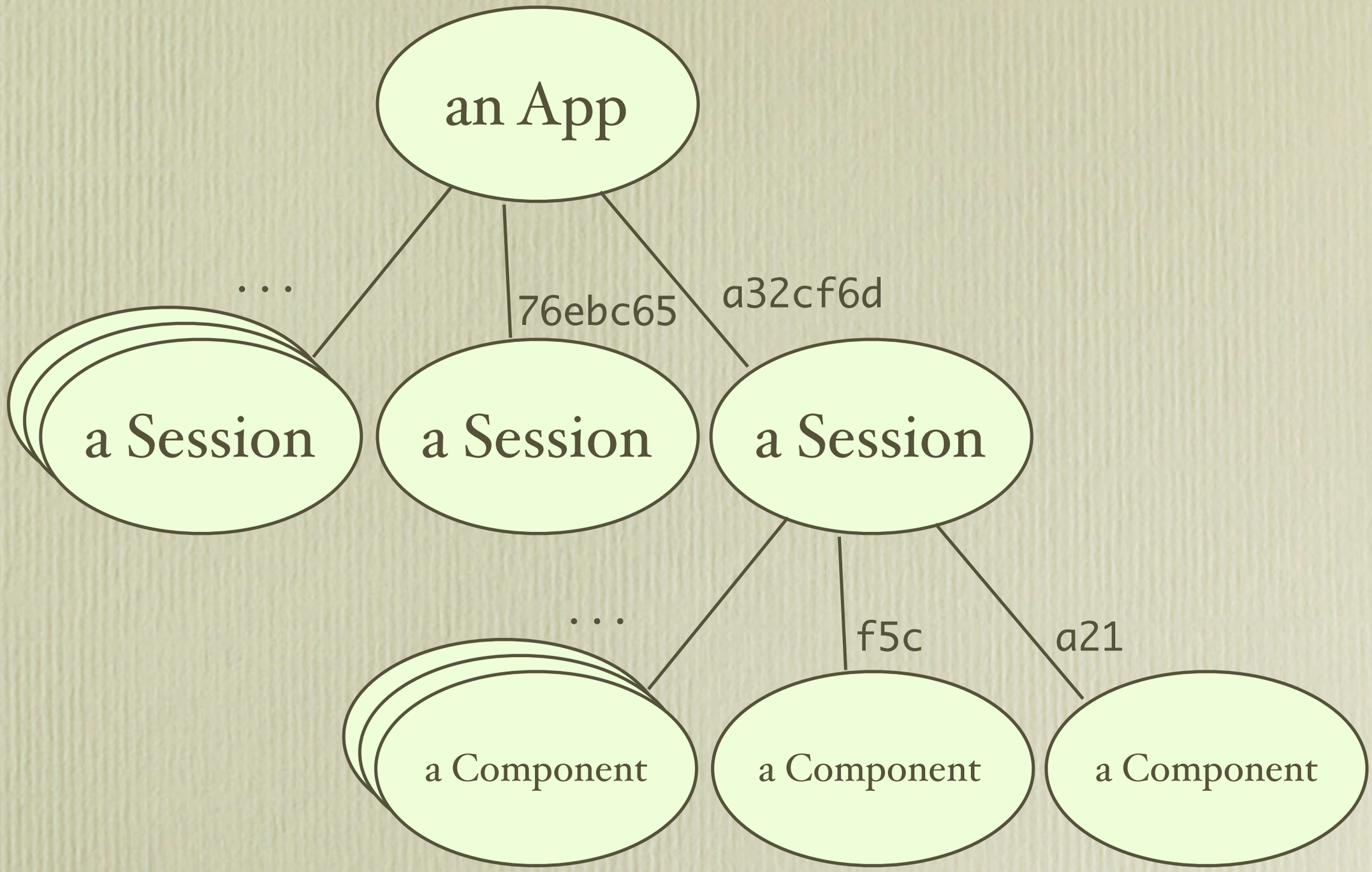


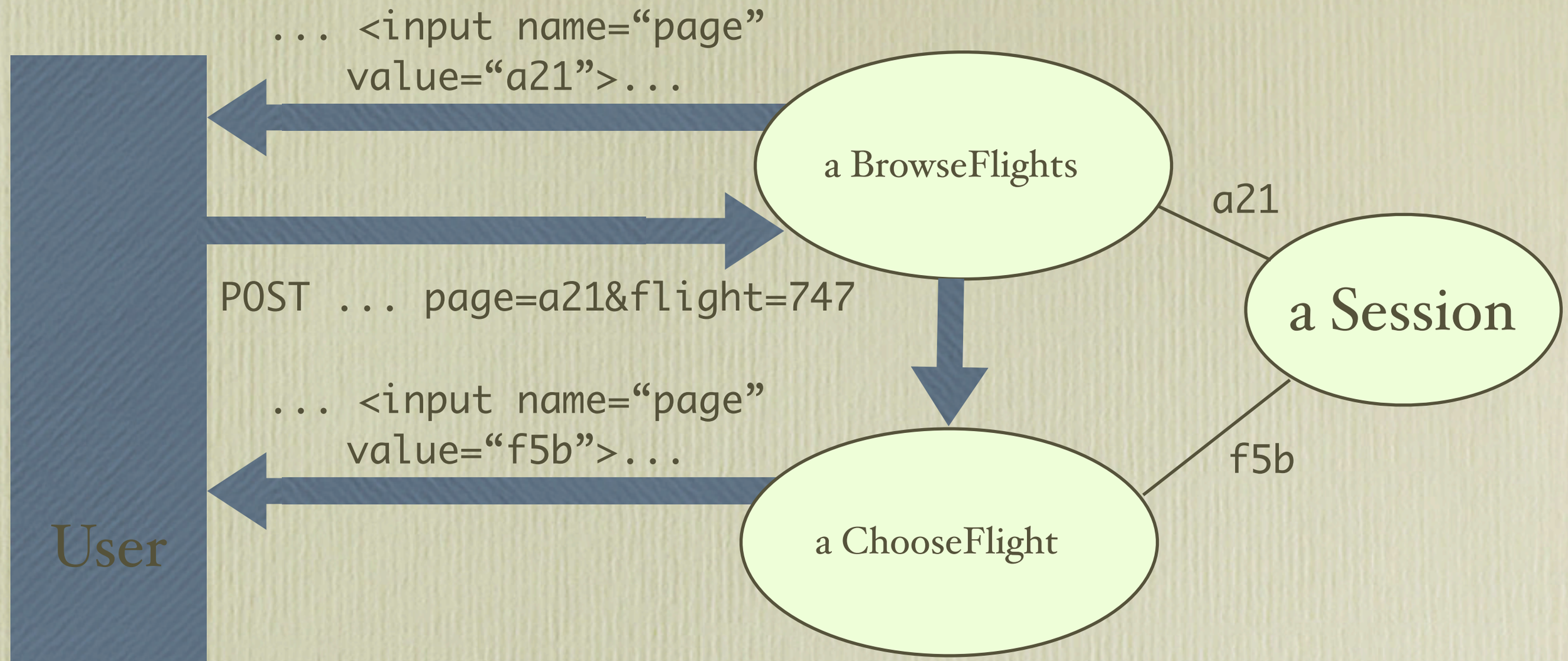
Why global session state is evil

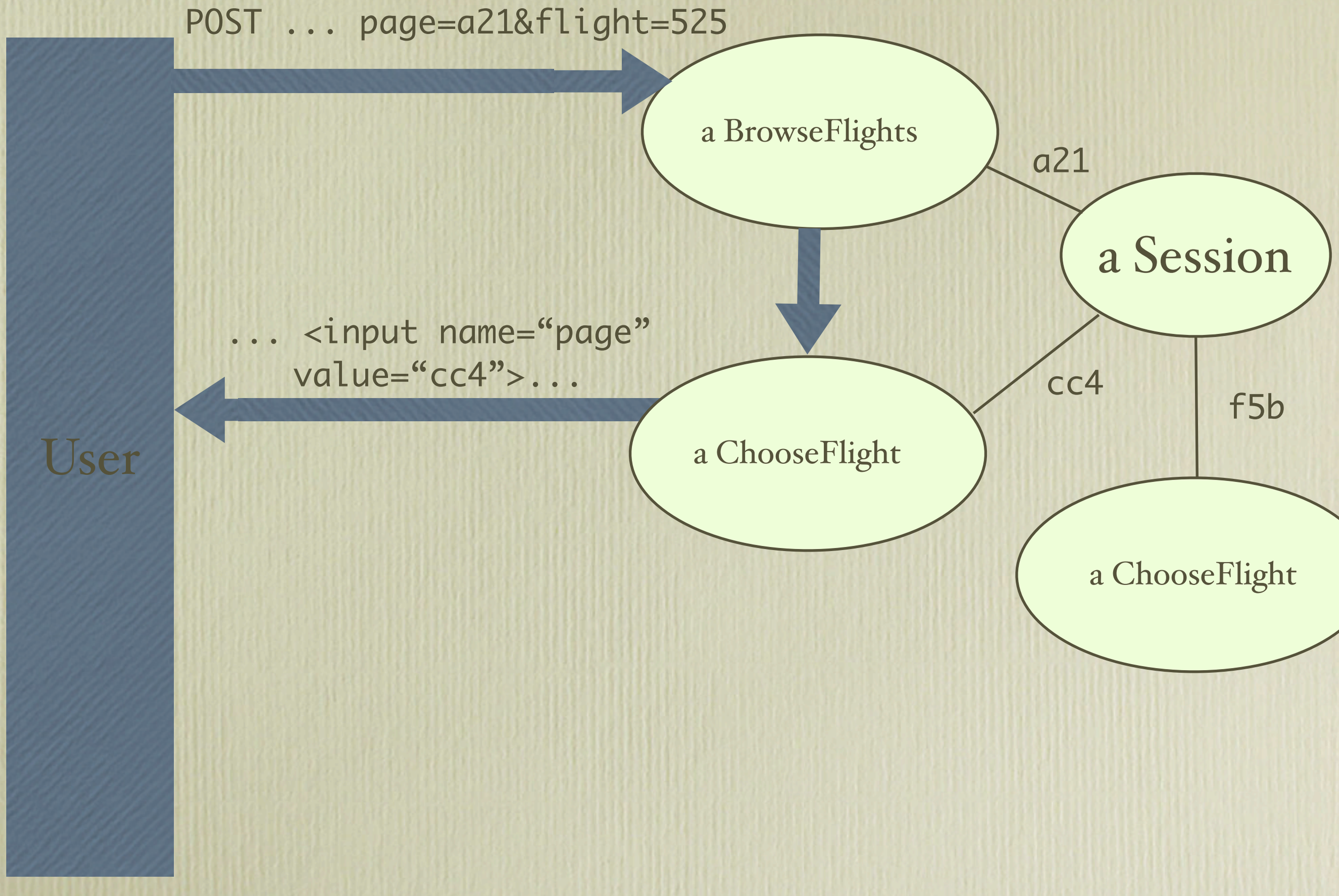


Why global session state is evil




an App
. . .
76ebc65
a32cf6d
a Session
a Session
a Session
. . .
f5c
a21
a Component
a Component
a Component


... <input name="page"
value="a21">...
a BrowseFlights
a21
a Session
POST ... page=a21&flight=747
... <input name="page"
value="f5b">...
f5b
a ChooseFlight
User


POST ... page=a21&flight=525
a BrowseFlights
a21
a Session
... <input name="page"
value="cc4">...
cc4
f5b
a ChooseFlight
User
a ChooseFlight

```
WOComponent
  subclass: #BillingAddress
  instanceVariableNames: 'ship bill'

 response
  ^ '<form ...'

processRequest: aRequest
  bill := self addressFrom: aRequest.

  ^ PaymentInfo new
     shippingAddress: ship;
     billingAddress: bill;
     yourself
```

“Seaside is to most other Smalltalk web toolkits as Smalltalk is to most other OO languages, it’s as simple as that.”

— Cees de Groot

WebObjects

- http://www.apple.com/webobjects/
- http://jakarta.apache.org/tapestry/

```
WOComponent
  subclass: #BillingAddress
  instanceVariableNames: ‘ship bill’


processRequest: aRequest
  bill := self addressFrom: aRequest.
```

```
^ PaymentInfo new
    shippingAddress: ship;
    billingAddress: bill;
    yourself
```

```
WOComponent
  subclass: #BillingAddress
  instanceVariableNames: ‘ship bill’


processRequest: aRequest
  bill := self addressFrom: aRequest.

^ PaymentInfo new
    shippingAddress: ship;
    billingAddress: bill;
    next: (ConfirmationPage new)
    yourself
```

```
WOComponent
  subclass: #BillingAddress
  instanceVariableNames: ‘ship bill’


processRequest: aRequest
  bill := self addressFrom: aRequest.
  ^ next
     shippingAddress: ship;
     billingAddress: bill;
     yourself
```

```
checkoutProcess
 ^ (ShippingAddress new next:
    (BillingAddress new next:
       (PaymentInfo new next:
          (ConfirmationPage new))))
```

```
WOComponent
  subclass: #BillingAddress
  instanceVariableNames: ‘ship bill’

processRequest: aRequest
  bill := self addressFrom: aRequest.

  ^ next value: bill

“BillingAddress new next:
  [:bill |
  PaymentInfo new
    billingAddress: bill;
    ....]”
```

```
checkoutProcess
 ^ ShippingAddress new next:
     [:ship |
     BillingAddress new next:
      [:bill |
      PaymentInfo new next:
       [:pay |
  ConfirmationPage new
   shippingAddress: ship;
   billingAddress: bill;
   paymentInfo: pay;
   yourself]
```

```
checkoutProcess
 ^ PaymentInfo new next:
     [:pay |
     ShippingAdress new next:
      [:ship |
      BillingAddress new next:
       [:bill |
      ConfirmationPage new
       shippingAddress: ship;
       billingAddress: bill;
       paymentInfo: pay;
       yourself]
```

"...programming language features do well (all other things being equal) when they eliminate either distant or dynamic state and replace it with either close or lexical state. The underlying point being that we may favour language features that facilitate copying and modifying small bits of code -- fragments which work in their new context -- as a fundamental programming activity."

— Graydon Hoare

```
checkoutProcess
 |pay ship bill|
 pay := self call: PaymentInfo new.
 ship := self call: ShippingAddress new.
 bill := self call: BillingAddress new.
 self call:
  (ConfirmationPage new
shippingAddress: ship;
billingAddress: bill;
paymentInfo: pay)
```

"We could write the code to say, if the user clicks on this link, go to the color selection page, and then come back here.... It made our software visibly more sophisticated than that of our competitors."

— Paul Graham

```
<form>
Name: <input name="name"><br>
Street: <input name="street"><br>
City: <input name="city"><br>
Country:
  <select name="country">
    <option value="CA">Canada</option>
    <option value="US">US</option>
  </select>
<br>
<input type="submit">
</form>
```

aRequest(‘name’->‘Avi’,
‘street’-> ‘123 W. 15th’
‘city’->‘Vancouver’,
‘country’-> ‘CA’)

```
renderOn: html

html form: [
  html label: 'Name'.
  html textInputNamed: 'name'; break.
  html label: 'Street'.
  html textInputNamed: 'street'; break.
  html label: 'City'.
  html textInputNamed: 'city'; break.
  html label: 'Country'.
  html selectNamed: 'country' do: [
    html optionNamed: 'CA' label: 'Canada'.
    html optionNamed: 'US' label: 'US'.
  ]; break.
  html submitButton.
]

processRequest: aRequest
  name := aRequest at: 'name'.
  street := aRequest at: 'street'.
  ...
```

```
renderOn: html

html form: [
  html label: 'Name'.
  html textInputWithCallback: [:v | name := v]; break.
  html label: 'Street'.
  html textInputWithCallback: [:v | street := v]; break.
  html label: 'City'.
  html textInputWithCallback: [:v | city := v]; 'city'; break.
  html label: 'Country'.
  html selectFromList: self countries; break.
  html submitButtonWithAction: [self saveAddress].
]
```

```
<form>
Name: <input name="1"><br>
Street: <input name="2"><br>
City: <input name="3"><br>
Country:
  <select name="4">
    <option value="5">Canada</option>
    <option value="6">US</option>
  </select>
<br>
<input type="submit" name="7" value="Submit">
</form>
```

aCallbackStore(
‘1’->[:v | name := v],
‘2’->...,
‘7’->[self saveAddress])

aRequest(
‘1’->‘Avi’,
‘2’->...,
‘7’-> ‘Submit’)