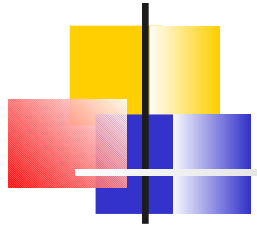


Various use of continuations in Kahua

Applications in practical web
programming experience

Katsutohi Itoh

Kahua Project



Higher-Level APIs for web-applications



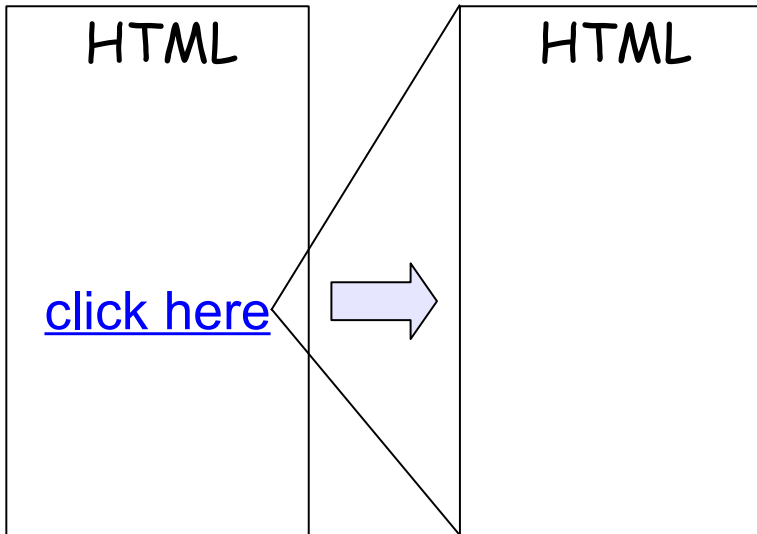
Continuation for web-application

- Continuation passing style is easy to write control flow on web programming.
- But to write practical web applications, we want to deal with:
 - Individual components (like widgets)

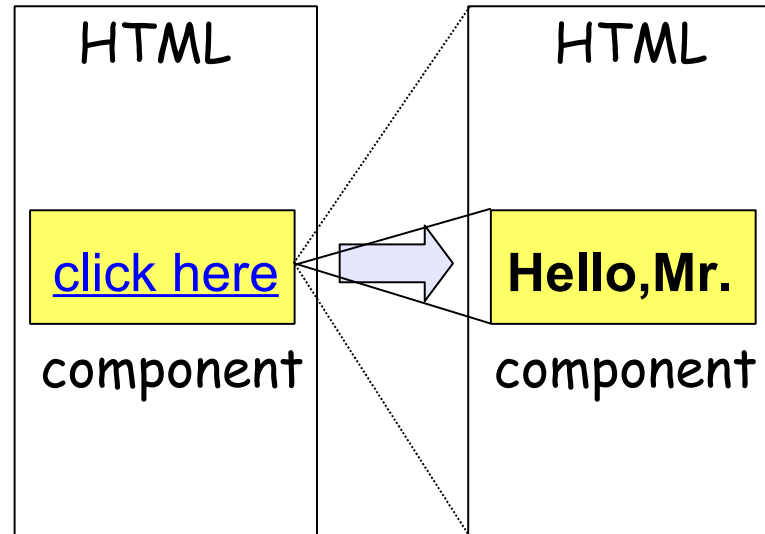


Continuation of components

page -> page



component -> component?
component -> page?



How continuations of components work?

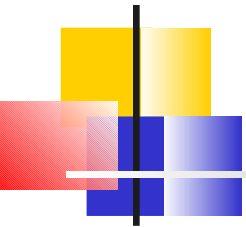


Start from this issue

Write a program:

- Produce a page with an input field & a submit button.
- When input text is submitted, display an anchor link saying "click here".
- When the link is clicked, display the text posted to the first page.

by "Take THE Arc Challenge"
<http://www.paulgraham.com/arcchallenge.html>



Easy to write control flow

```
::: said
(define-entry (said)
  (page
    (form/cont/
      (@@/ (cont (lambda ()
        (let1 say (kahua-context-ref "say")
          (page
            (a/cont/ (@@/ (cont (lambda ()
              (page
                (p/ "you said: " say))))))
              "click here"))))))))
        (readln/ "say")
        (submit/))))))
```

→ Call continuation procedure
→ Generate page



Individual component

In practical web applications

- **Many components co-exist** in most web pages
- Want some kind of components to work **individually**
 - ex. Something to redraw a part of page - login box embedded in page like as reddit.com, calendar as date selector at any blog site ...
- Not want a component's continuation to have the whole next page



Motivation

We want to write like this:

```
:: using individual "said"  
(define-entry (said5)  
  (page  
    (map/ said ('("Alf" "Willie" "Kate" "Lynn" "Brian")))))
```

The "said5" has 5 individual "said" components.

How about this?

:: Does this "said" works individually?

```
(define (said id)
  (form/cont/
    (@@/ (cont (lambda ()
                (let1 say (kahua-context-ref id)
                  (a/cont/ (@@/ (cont (lambda ()
                                      (p/ id " said: " say))))
                    "click here"))))))
    (readln/ id)
    (submit/)))
```

→ Call continuation procedure
→ Generate page

:: http://localhost/app/said5

```
(define-entry (said5)
  (page (map/ said ('("Alf" "Willie" "Kate" "Lynn" "Brian")))))
```





Diff : Said application

```
(define-entry (said)
  (page
    (form/cont/
      (@@/ (cont (lambda ()
                  (let1 say (kahua-context-ref "say")
                    (page
                      (a/cont/ (@@/ (cont (lambda ()
                                          (page
                                            (p/ "you said: " say))))))
                                          "click here"))))))))
      (readln/ "say")
      (submit/))))))
```



Diff : Said component(?)

```
(define-entry (said id)

  (form/cont/
    (@@/ (cont (lambda ()
                (let1 say (kahua-context-ref "say")

                  (a/cont/ (@@/ (cont (lambda ()

                                (p/ id " said: " say))))
                                "click here")))))

    (readln/ "say")
    (submit/)))
```



The problem

Expected

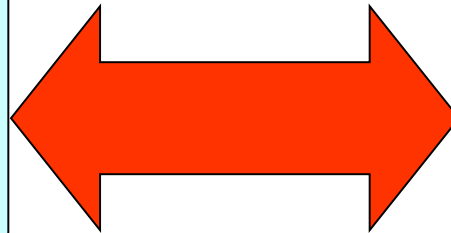
Each component
to be
independent
from the others

Continuation
does not have to
know the others

Happened

Continuation
generates **the**
whole page

Continuation
must know the
others





Solution : parts-cont

:: this "said" works as we expected

```
(define (said person)
```

```
  (form/cont/ (@/ (id person)) :: 3.form has the target id
```

```
    (@@/ (target person)      :: 2.update a target-id's node
```

```
      (parts-cont           :: 1.generate new node
```

```
        (lambda ()
```

```
          (let1 say (kahua-context-ref person)
```

```
            (div/ (@/ (id person))
```

```
              (a/cont/ (@@/ (target person)
```

```
                (parts-cont
```

```
                  (lambda ()
```

```
                    (p/ person " said: " say))))
```

```
                  "click here"))))))))
```

```
(readln/ person)
```

```
(submit/)))
```

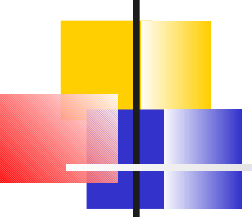


Diff : buggy said component

```
(define (said person)
  (form/cont/
    (@@/
      (cont
        (lambda ()
          (let1 say (kahua-context-ref person)

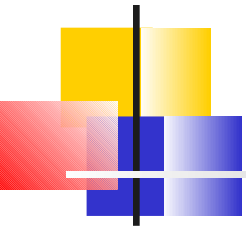
            (a/cont/ (@@/
              (cont
                (lambda ()
                  (p/ person " said: " say))))
              "click here")))))

  (readln/ person)
  (submit/)))
```



Diff : parts-cont version

```
(define (said person)
  (form/cont/ (@/ (id person))
    (@@/ (target person)
      (parts-cont
        (lambda ()
          (let1 say (kahua-context-ref person)
            (div/ (@/ (id person))
              (a/cont/ (@@/ (target person)
                (parts-cont
                  (lambda ()
                    (p/ person " said: " say))))
                "click here"))))))
    (readln/ person)
    (submit/)))
```



What "parts-cont" does

Make each "said" to work individually

Alf

Input form *submit!* → Anchor link *click!* → Show text

Willie

form link Text

Brian

form link Text

Lynn

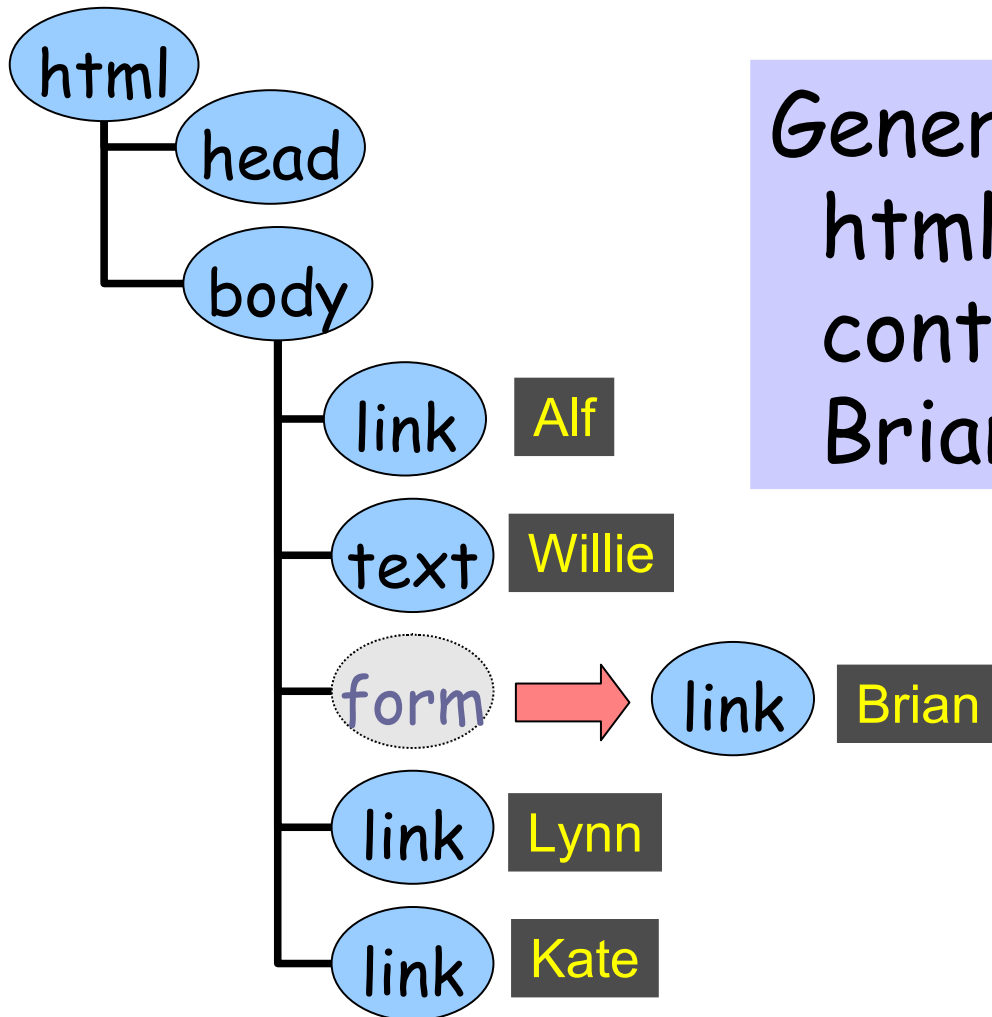
form link Text

Kate

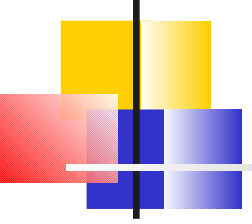
form link Text



What "parts-cont" does



Generate the whole html tree by the continuation of Brian's "said"



Mechanism : the key idea

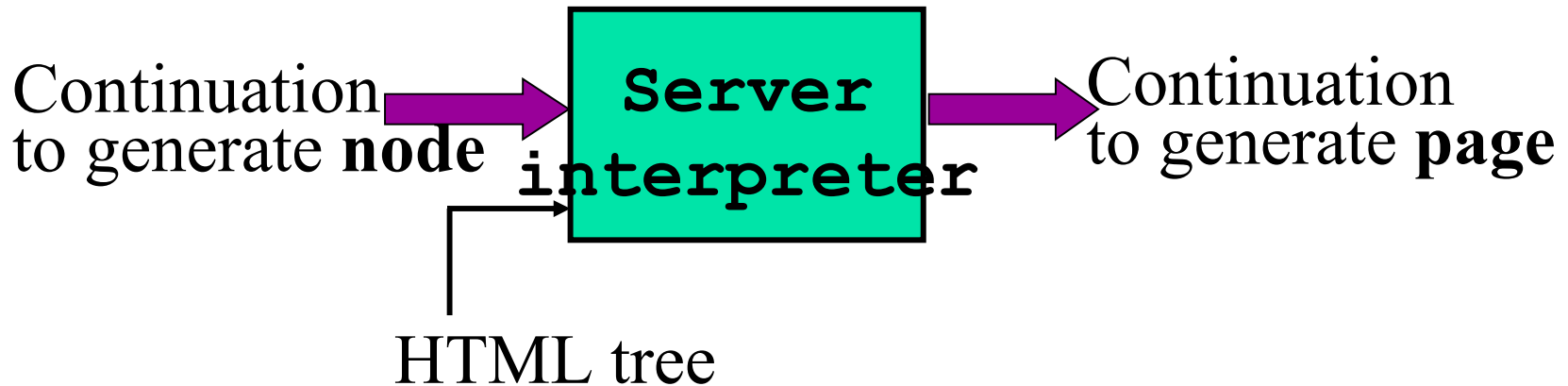


Create continuation that generate next page by replace target node with a new node which return from “parts-cont” clause



Design of the mechanism

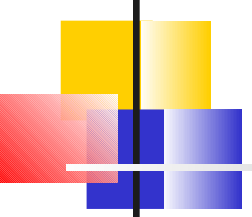
create continuation that generate next page by
replace target node with a new node which return
from “parts-cont” clause





More ...

The "parts-cont" mechanism
highlights a new need **to**
keep client-side context



Keep client-side context

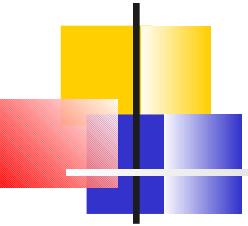
```
:: "said" with "keep"
(define (said id)
  (form/cont/ (@/ (id id))
    (@@/ (target id) (keep #t) ;; only add keep clause
      (parts-cont
        (lambda ()
          (let1 say (kahua-context-ref id)
            (div/ (@/ (id id))
              (a/cont/ (@@/ (target id) (keep #t)
                (parts-cont
                  (lambda ()
                    (p/ id " said: " say))))
                "click here"))))))
    (readln/ id)
    (submit/)))
```



A more interesting sample

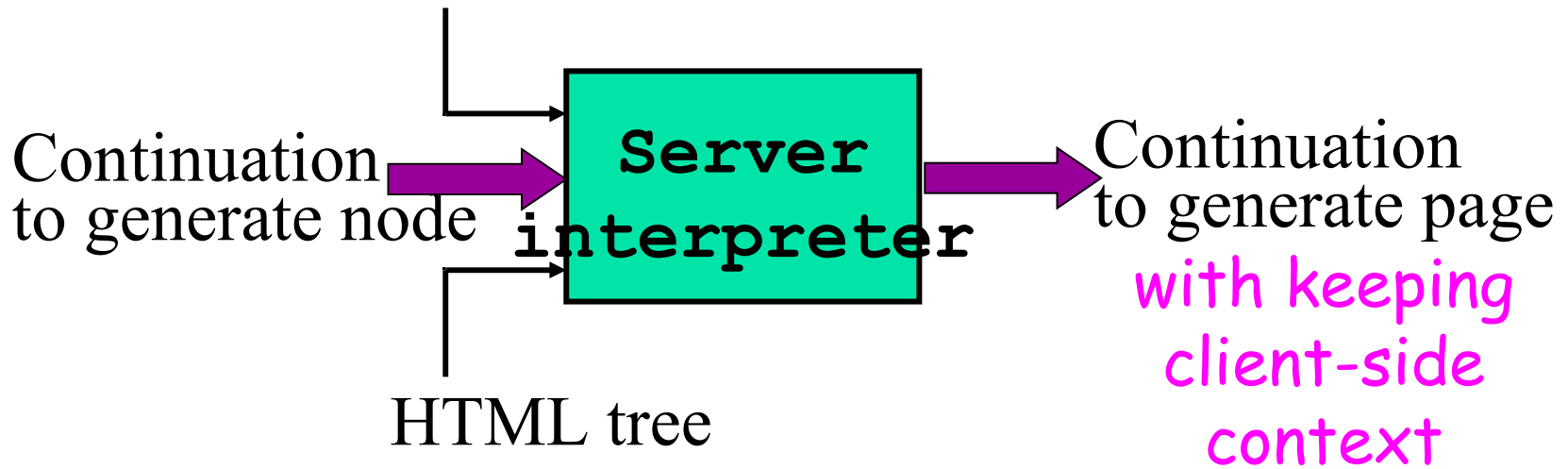
:: this "calendar/" function works as a widget of date selector.

```
(define-entry (plan)
  (page
    (form/cont/
      (@@/ (cont (entry-lambda (:keyword from to memo)
                  (make <plan>
                        :from from :to to :memo memo)
                        (plan))))
          (calendar/ "from" "Start" (current-date))
          (calendar/ "to" "End" (current-date))
          (readtext/ "memo")
          (submit/))
      (map/ display/
        (sort (coerce-to <list>
                  (make-kahua-collection <plan>)) plan>=?))))
```



Design of the mechanism

client-side context





Now, we have ...

The "parts-cont" mechanism,
which supports individual
components.

We can write web application
in smart way.



What next?

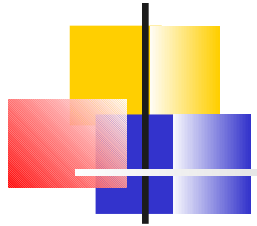
Refine the design of “parts-cont” mechanism.

Challenge this by using partial continuation technique.



What next?

Of course,
Fix **some known bugs** of parts-cont...



Thank you