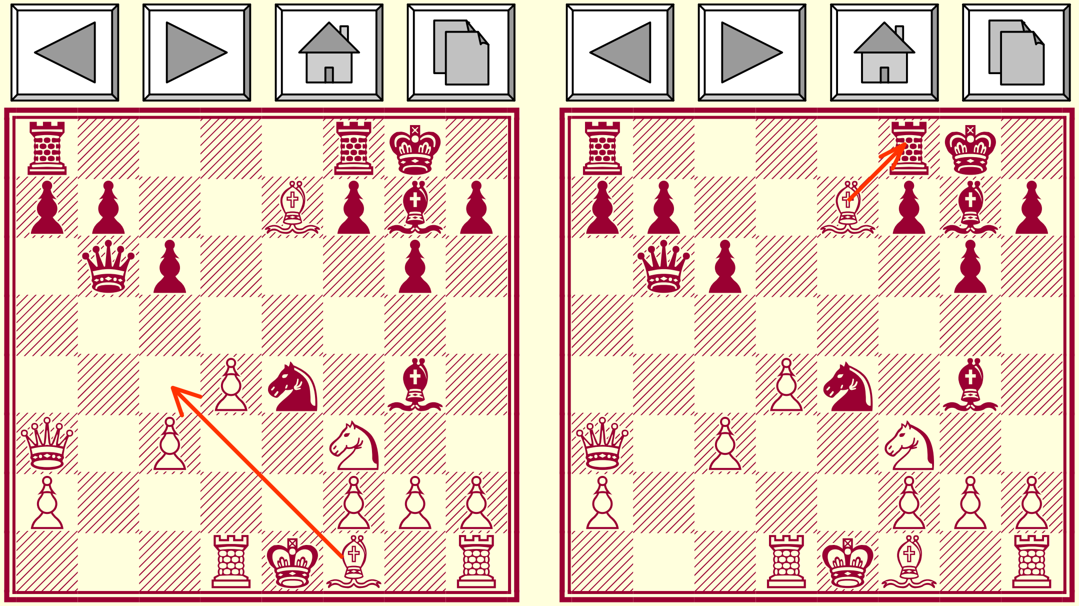
Automatically Restructuring Programs for the Web

Matthews, Graunke, Krishnamurthi, Findler, Felleisen



Web Scripts

≥ 50% of Web pages are generated on demand.

The so-called Web “*scripts*” are nowadays complex, evolving programs.

However, existing technology is inadequate.

Interactive Programming Paradigm

```
fun input msg =
  print msg;
  read

fun adder =
  print
    (input
      “1st number?”)
    +
    (input
      “2nd number?”)
```

Pro: interaction is computation driven;

⇒ natural programming style.

Serious Engineering Problem

Web scripts must terminate after producing one single page (exception: Fast CGI);

> **⇒ control information is erased between user interactions (Fast CGI solves this problem).**

Back button + window cloning;

> **⇒ the client becomes a co-routine with unbounded resumption points (Fast CGI *can't* solve it).**

Current Approaches

Solution: come up with a hack to explicitly store/recover state per hand.

```
fun produce-html msg hidden-mark env = ...

fun adder =
  hidden-mark = extract-hidden-mark
  env         = extract-env
  ans         = extract-answer
  if hidden-mark = undefined then
    produce-html “1st number?” “step 1” []
  else if hidden-mark = “step 1” then
    produce-html “2nd number?” “step 2” [ans]
  else if hidden-mark = “step 2” then
    produce-html ((hd env) + ans) “done” []
```

Current Approaches

Program Inversion: the interaction becomes user driven!

But inversion is:

- **unnatural;**
- **complicated;**
- **error prone, if done per hand;**
- **counter-productive.**

A Better Solution?

Use a PL that can explicitly manipulate continuations to grab and store them on the server [Queinnec 2000].

Problems:

most PLs don't support `call/cc`
⇒ existing infrastructure becomes useless;

distributed garbage collection problem;

timeouts are an imperfect solution.

A Better Solution?

~~Use a PL that has `call/cc`~~.

Write usual interactive programs in your favourite PL and environment;

use existing, well understood FP techniques to *automatically* transform them into programs for the Web.

The Preprocessing Solution

How can we grab, send, and resume continuations in an arbitrary PL? By transforming the program with:

continuation passing style (CPS),

lambda lifting,

defunctionalization.

One last step generates a program for the web.

Continuation Passing Style

```
fun input msg =
  print msg;
  read

fun adder =
  print
    (input
      “1st number?”)
    +
    (input
      “2nd number?”)
```

```
fun input msg f =
  print msg;
  f read

fun adder =
  input
    “1st number?”
    λ n0 =>
      input
        “2nd number?”
        λ n1 =>
          print n0 + n1
```

Lambda lifting

```
fun input msg f =
  print msg;
  f read

fun adder =
  input
    “1st number?”
    λ n_0 =>
      input
        “2nd number?”
        λ n_1 =>
          print n_0 + n_1
```

```
fun input msg f env =
  print msg;
  f env read

fun adder =
  input “1st number?”
        f_0 []

fun f_0 [] n_0 =
  input “2nd number?”
        f_1 [n_0]

fun f_1 [n_0] n_1 =
  print n_0 + n_1
```

Defunctionalization

```
fun input msg f env =
  print msg;
  f env read

fun adder =
  input "1st number?"
        $f_0$ []

fun $f_0$ [] $n_0$ =
  input "2nd number?"
        $f_1$ [$n_0$]

fun $f_0$ [$n_0$] $n_1$ =
  print $n_0$ + $n_1$
```

```
fun input msg idx env =
  print msg;
  apply idx env read

vector funs = {$f_0$, $f_1$}

fun apply idx env =
  funs.idx env

fun adder =
  input "1st number?"
        0 []

fun $f_0$ [] $n_0$ =
  input "2nd number?"
        1 [$n_0$]

fun $f_0$ [$n_0$] $n_1$ =
  print $n_0$ + $n_1$
```

The Preprocessing Solution

What have we done till now?

- **CPS: the only function that interacts with the user (`input`) is passed a continuation.**
  - **λ lifting: the structure of the program is flattened; all functions are named and global.**
    - **Defunctionalization: no function is passed or returned.**
      - **Till now, the function `input` simply executes the continuation passed to it...**

Interactive Program → CGI Program

```
fun input msg idx env =
  print msg;
  apply idx env read

vector funs...
fun apply...

fun adder =







  input "1st number?"
        0 []
fun f0...
fun f1...
```

```
fun input msg idx env =
  produce-html
    msg idx env

vector funs...
fun apply...

fun adder =

  idx = extract-idx;
  env = extract-env;
  ans = extract-answer;
  apply idx env ans;
  handle NoCont =>
    input "1st number?"
          0 []
fun f0...
fun f1...
```

You can do it also in C...

```
#include <stdio.h>

typedef struct {
  int code;
  void *env;
} closure;

typedef void
 (*closuretype)(void*, void*);

void input(char *s, closure *k);

closure *make_closure(
  int code, void *env){
  closure *k = (closure*)
    malloc(sizeof(closure));
  k->code = code, k->env = env;
  return k;
}

void f0(void *env, void *n0) {
  closure *k = make_closure(1, n0);
  input("2nd number?", k);
}

void f1(void *n0, void *n1) {
  printf("%d\n", (int) n0 + (int) n1);
}
```

```
closuretype closures[] = {f0, f1};

void apply(closure *k, void *args){
  int code = k->code;
  void *env = k->env;
  free(k);
  (*(closures[code]))(env, args);
}

void input(char *s, closure *k){
  char buffer[10];
  int i;
  printf("%s", s);
  fgets(buffer,10, stdin);
  i = atoi(buffer);
  apply(k, (void *) i);
}

int main() {
  closure *k =
    make_closure(0, (void *) 0);
  input("1st number?", k);
  return 0;
}
```

...or even in BASIC...

The dispatcher:

```
REM adder
...
     IF idx = 0 THEN GOTO 100
ELSE IF idx = 1 THEN GOTO 200


...


100 REM f0
...
200 REM f1
...
```

Save Store in Cookies

Problem: the store is independent from the continuations.

```
val high_score = ref 0;

...

high_score := !high_score + 1;
```

Solution: save the store in a cookie.

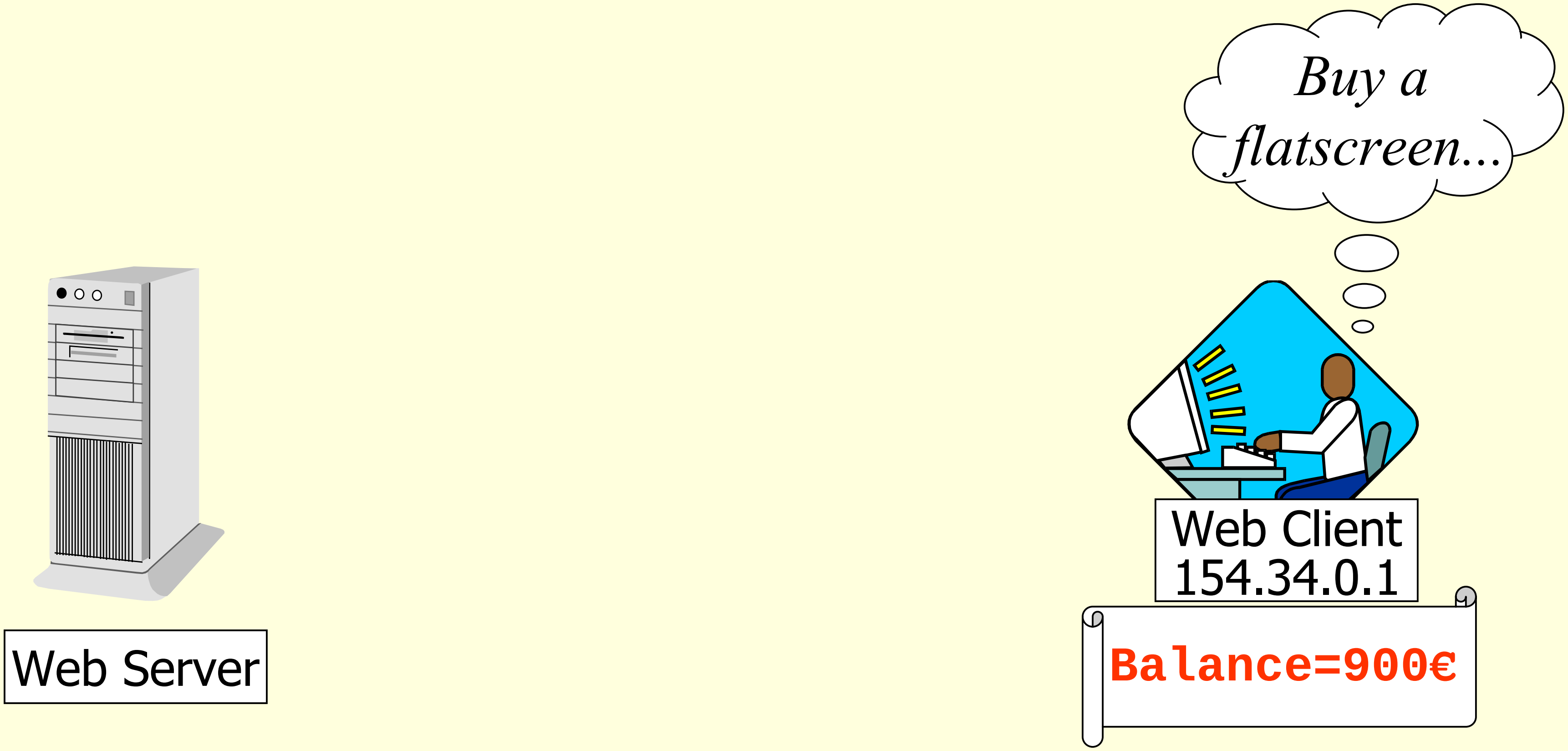
Problem: Race Conditions

Web Server

Web Client
154.34.0.1

Balance=900€

Problem: Race Conditions



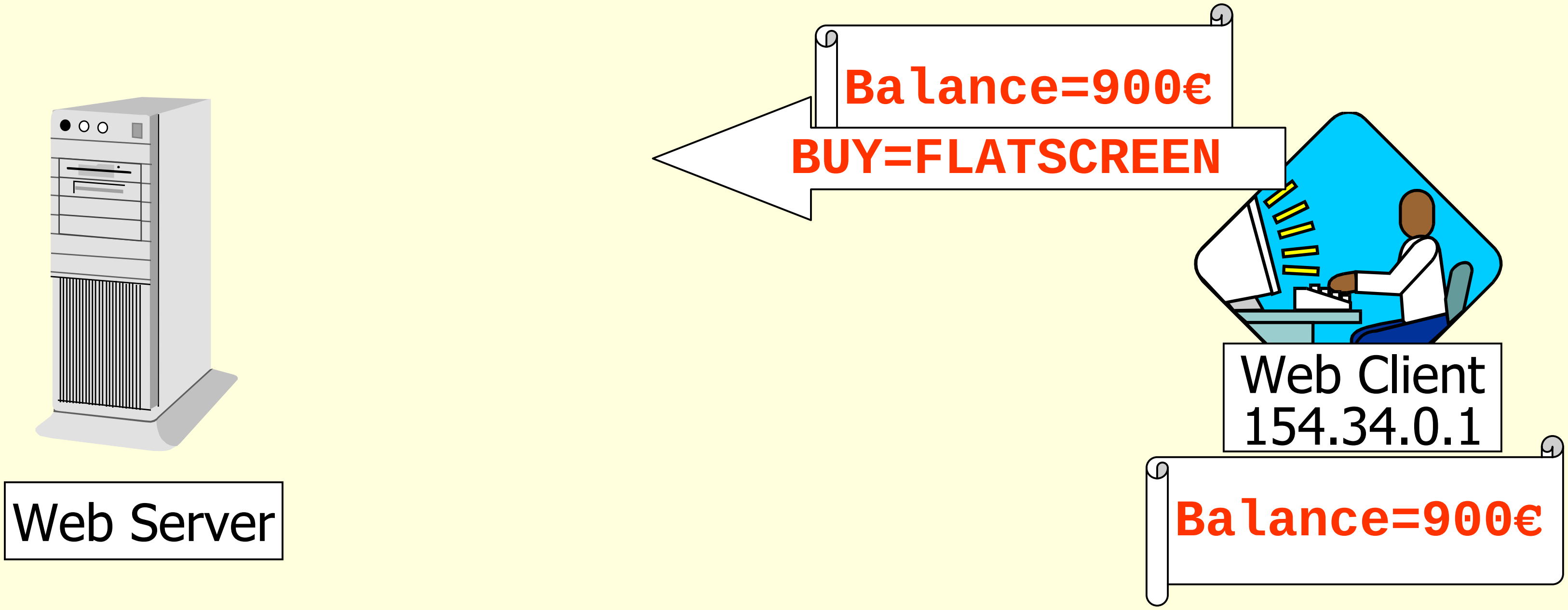
Buy a flatscreen…

Web Server

Web Client
154.34.0.1

Balance=900€

Problem: Race Conditions



Balance=900€

BUY=FLATSCREEN

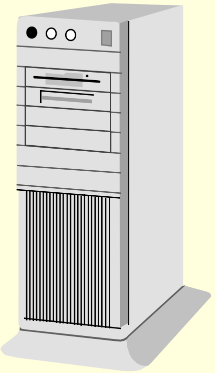
Web Server

Web Client
154.34.0.1

Balance=900€

Problem: Race Conditions

Balance=160€

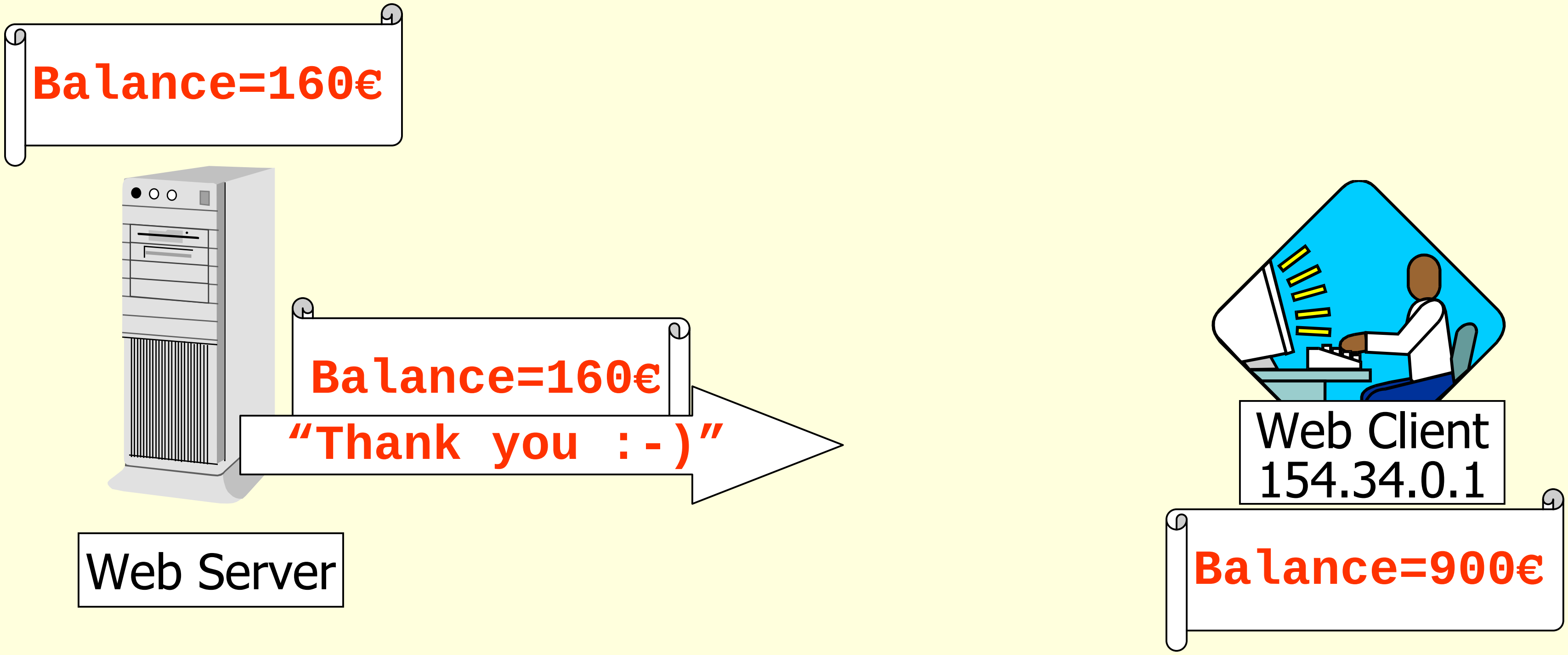


Web Server

Web Client
154.34.0.1

Balance=900€

Problem: Race Conditions



Balance=160€

Balance=160€

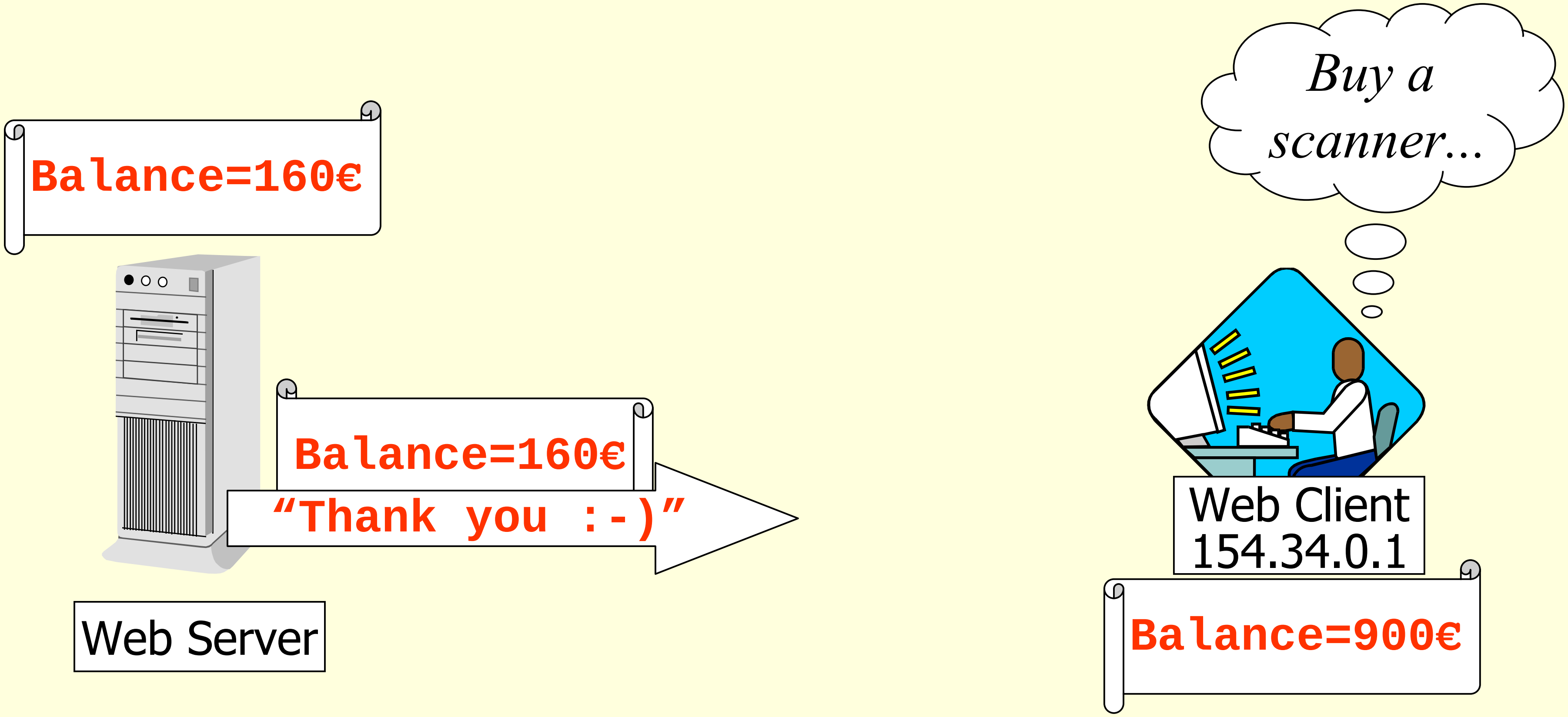
“Thank you :-)”

Web Server

Web Client
154.34.0.1

Balance=900€

Problem: Race Conditions



Balance=160€

Balance=160€

“Thank you :-)”

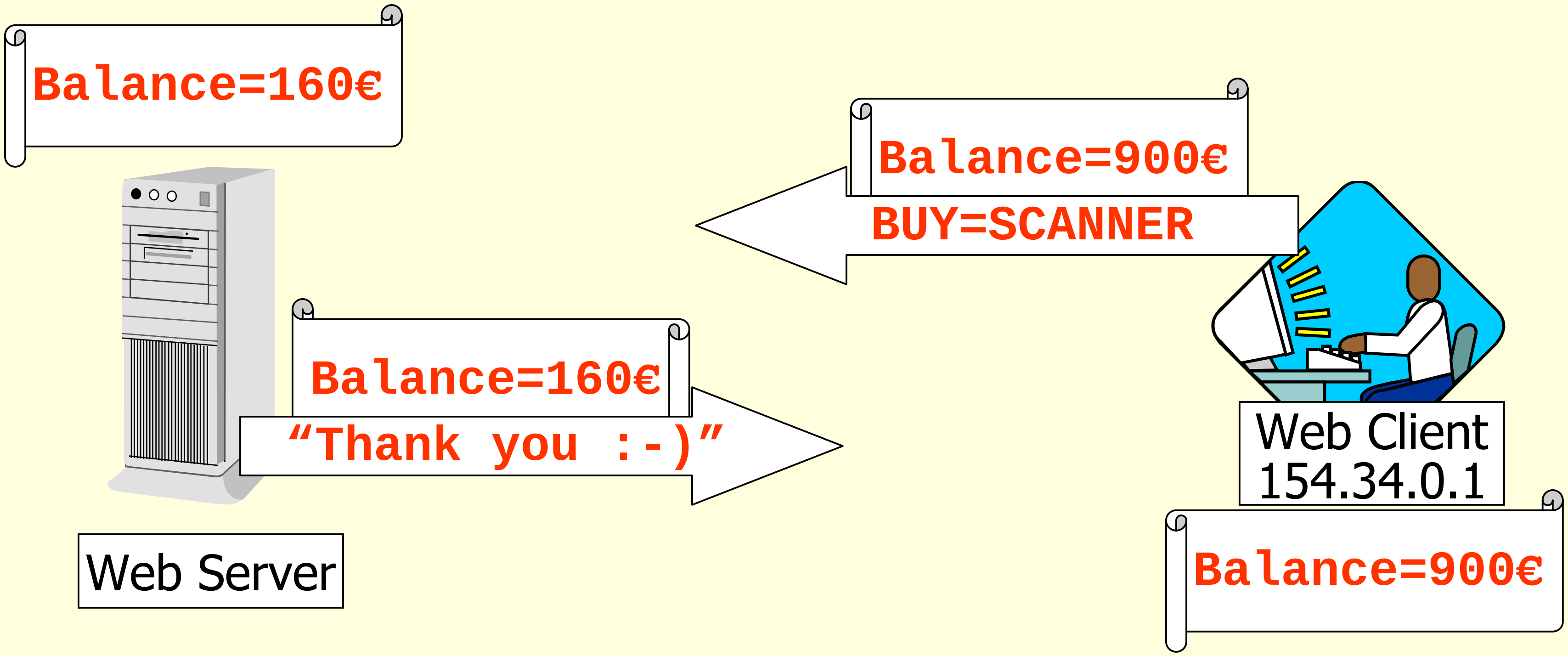
Web Server

Buy a scanner…

Web Client
154.34.0.1

Balance=900€

Problem: Race Conditions



Balance=160€

Balance=900€

BUY=SCANNER

Balance=160€

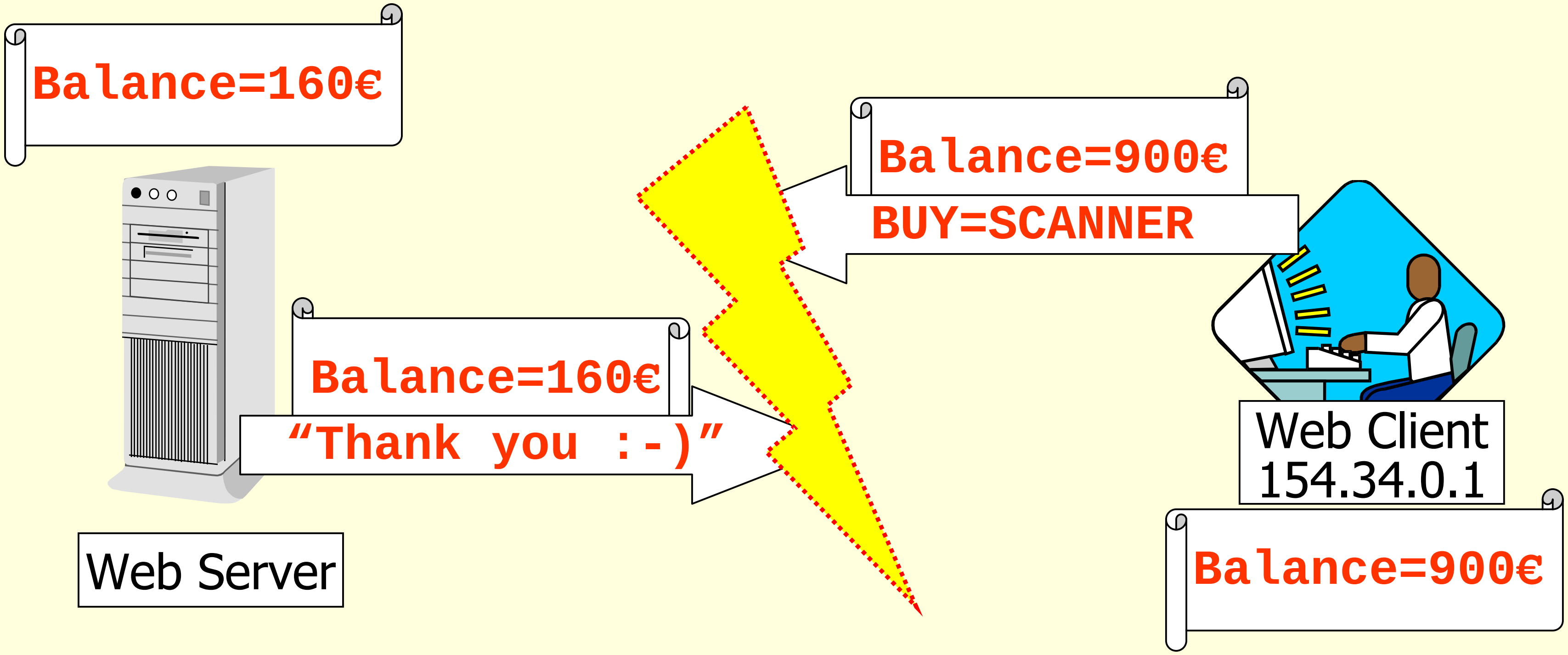
“Thank you :-)”

Web Client
154.34.0.1

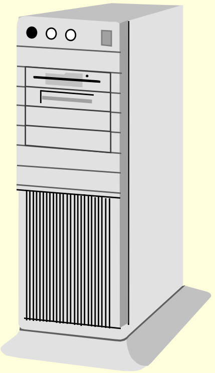
Web Server

Balance=900€

Problem: Race Conditions



Solution: Sequence Numbers



Web Server
154.34.0.1:**7665671**

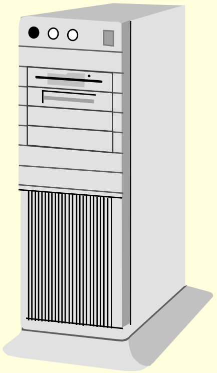
Web Client
154.34.0.1

SEQ=7665671
Balance=900€

Solution: Sequence Numbers

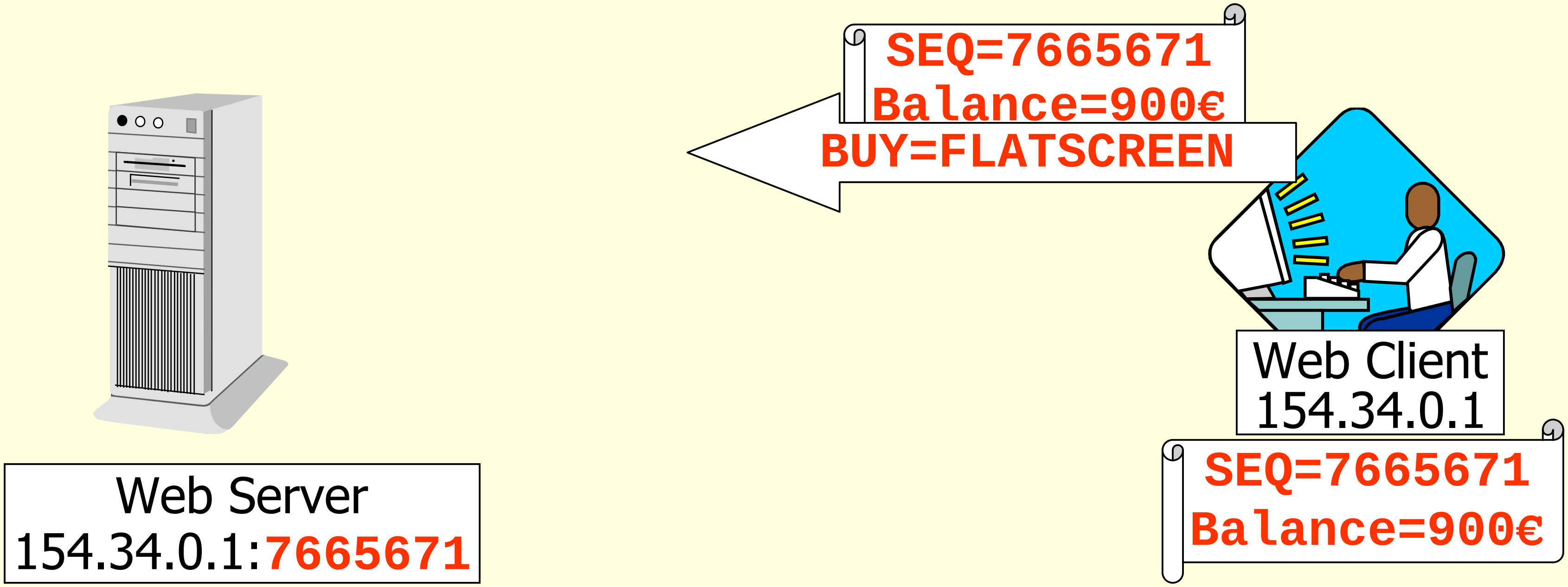
Web Client
154.34.0.1

SEQ=7665671
Balance=900€



Web Server
154.34.0.1:7665671

Solution: Sequence Numbers



SEQ=7665671
Balance=900€
BUY=FLATSCREEN

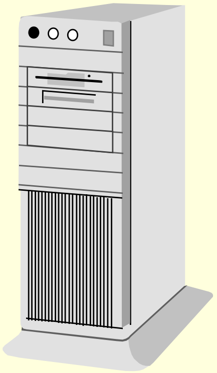
Web Server
154.34.0.1:7665671

Web Client
154.34.0.1

SEQ=7665671
Balance=900€

Solution: Sequence Numbers

Balance=160€

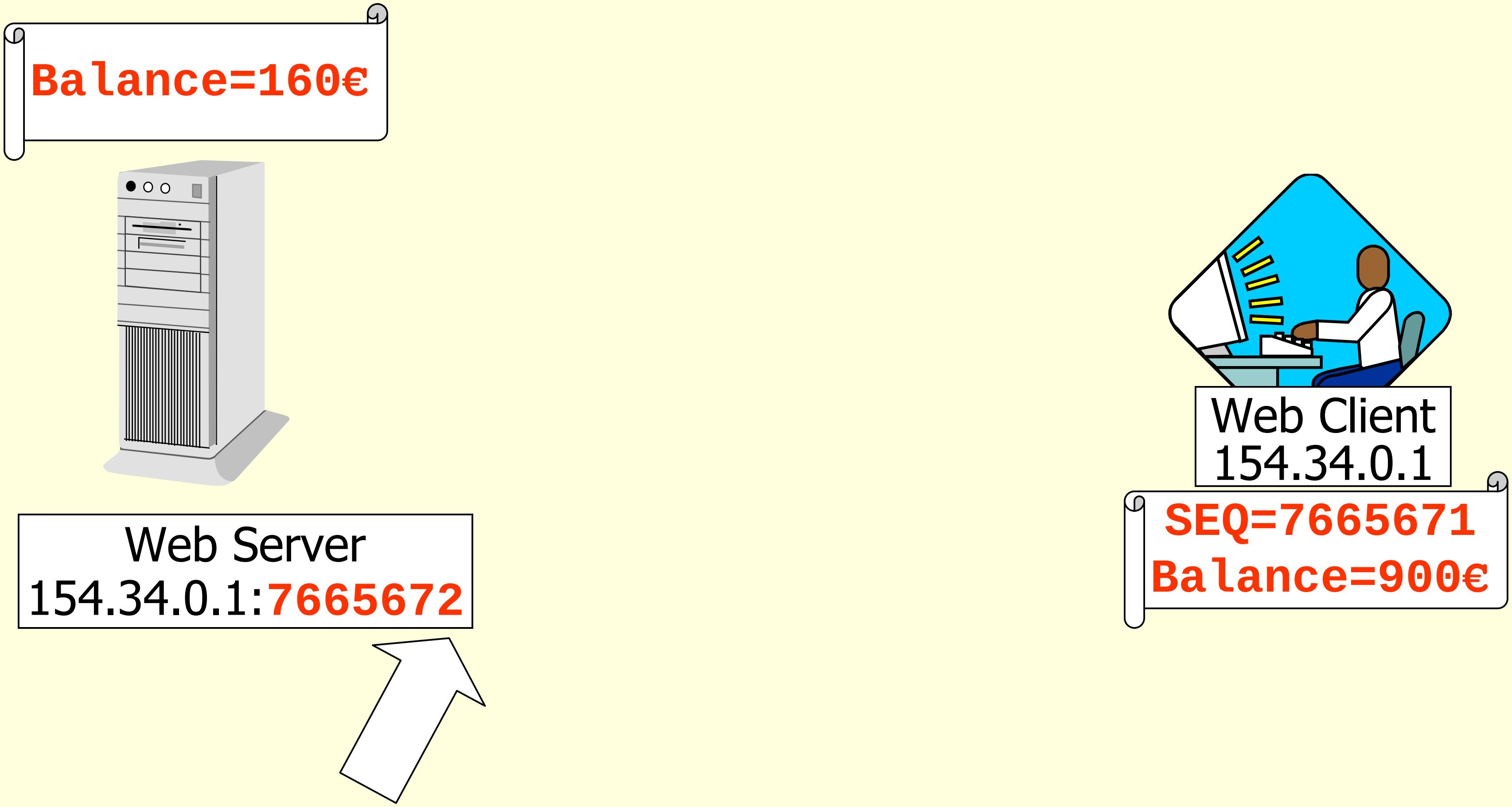


Web Server
154.34.0.1:7665671

Web Client
154.34.0.1

SEQ=7665671
Balance=900€

Solution: Sequence Numbers



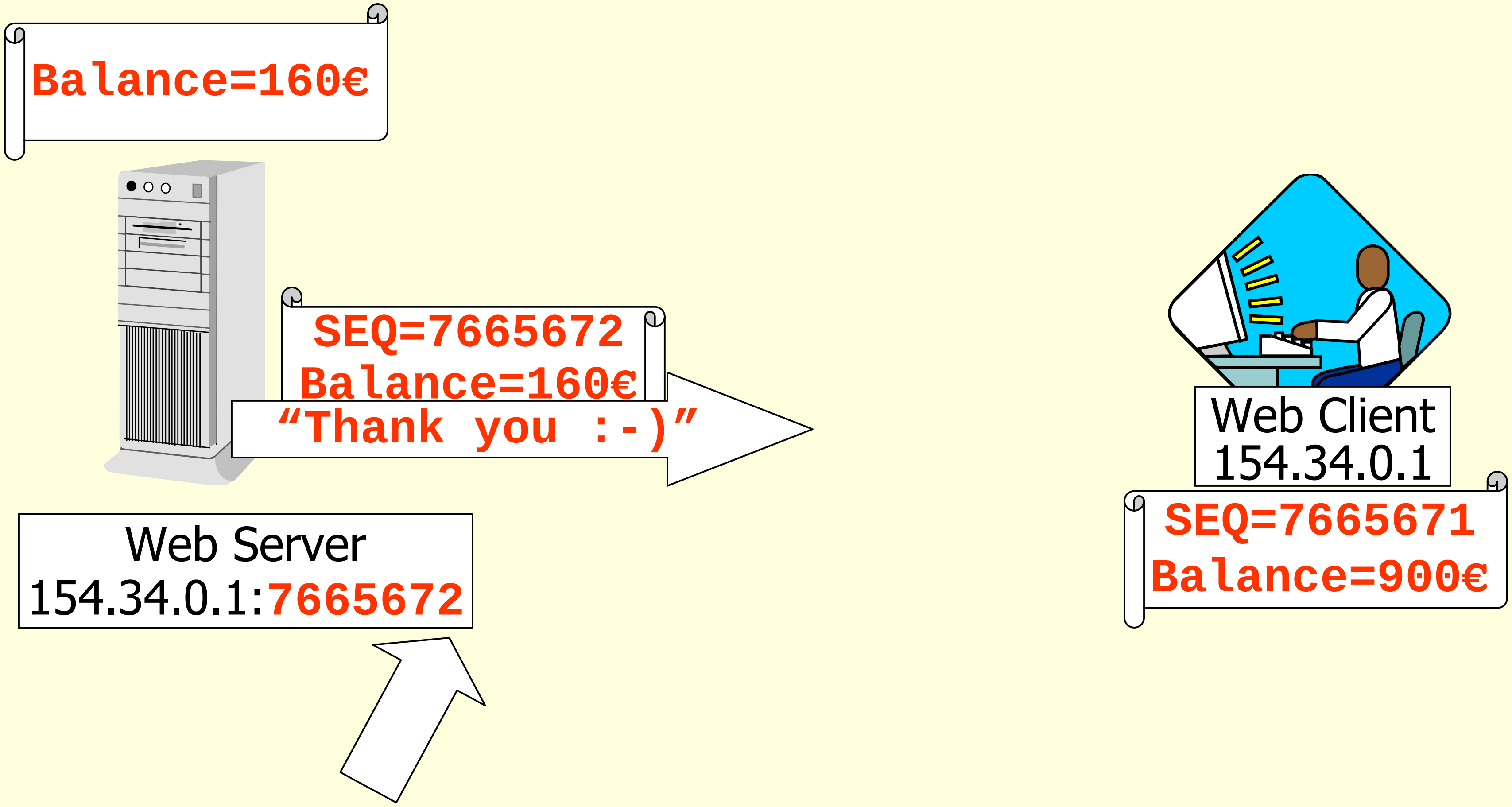
Balance=160€

Web Server
154.34.0.1:7665672

Web Client
154.34.0.1

SEQ=7665671
Balance=900€

Solution: Sequence Numbers



Balance=160€

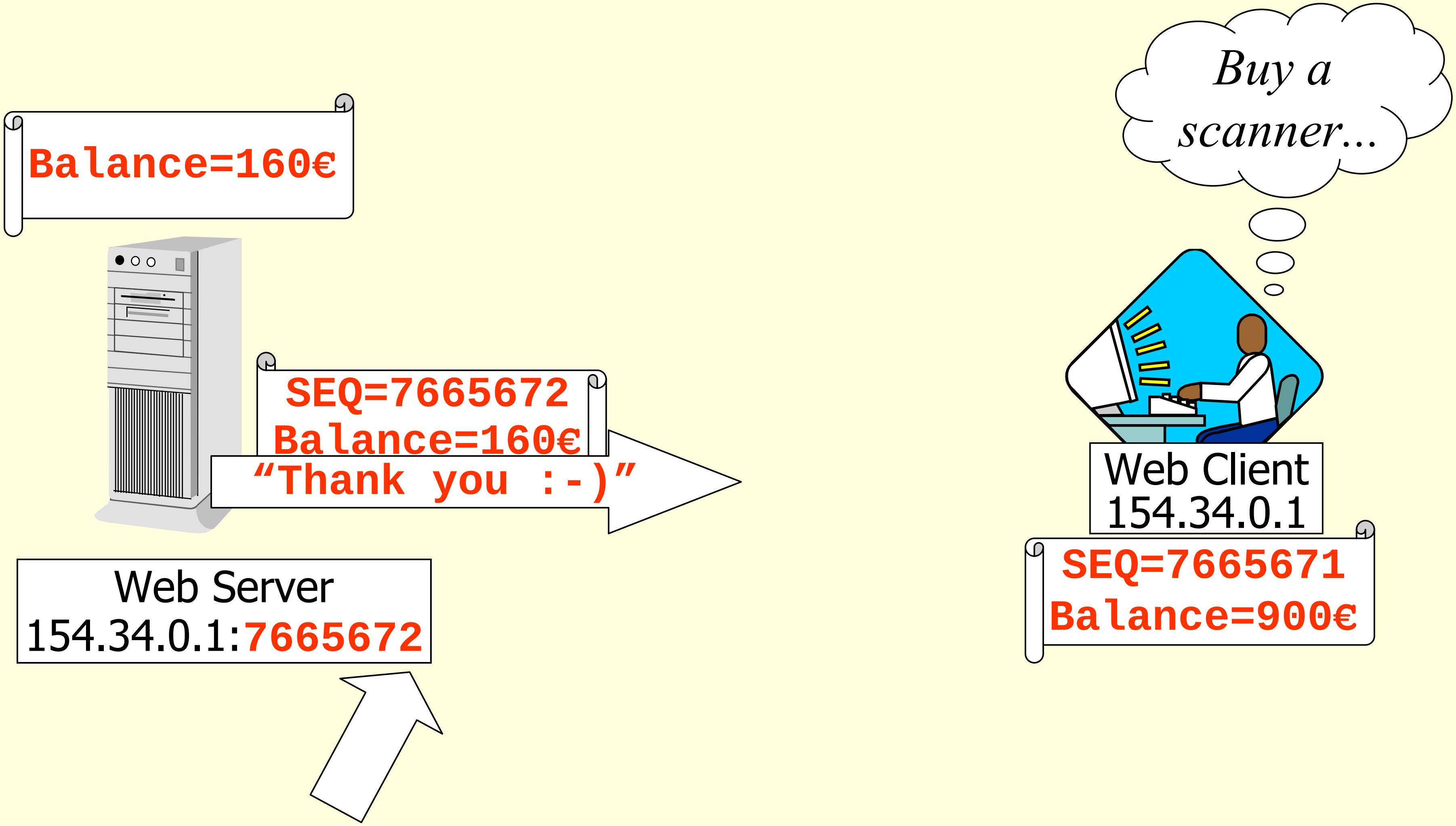
SEQ=7665672
Balance=160€
“Thank you :-)”

Web Server
154.34.0.1:7665672

Web Client
154.34.0.1

SEQ=7665671
Balance=900€

Solution: Sequence Numbers



Balance=160€

SEQ=7665672
Balance=160€
“Thank you :-)”

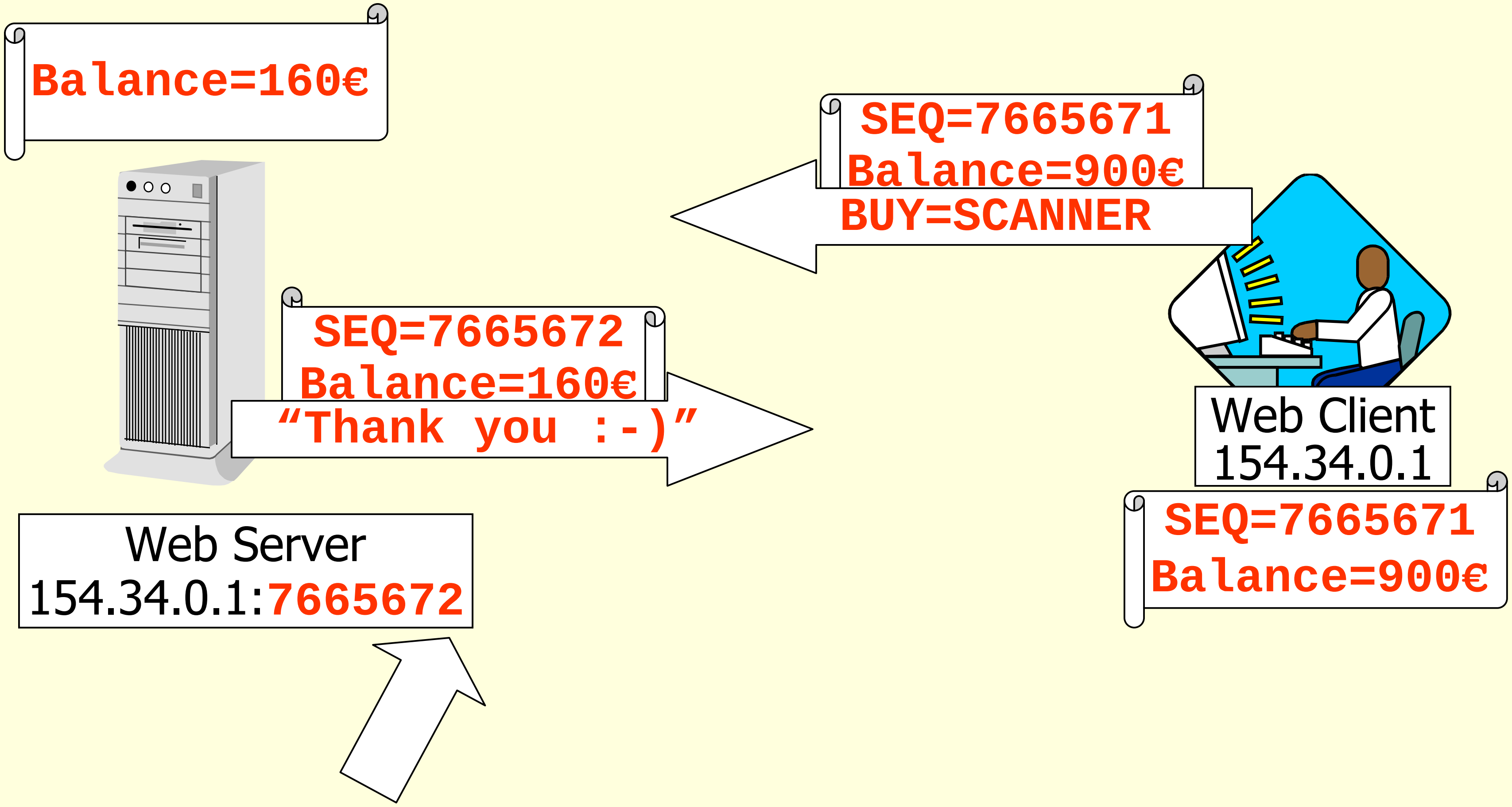
Web Server
154.34.0.1:7665672

Buy a scanner…

Web Client
154.34.0.1

SEQ=7665671
Balance=900€

Solution: Sequence Numbers



Balance=160€

SEQ=7665671
Balance=900€
BUY=SCANNER

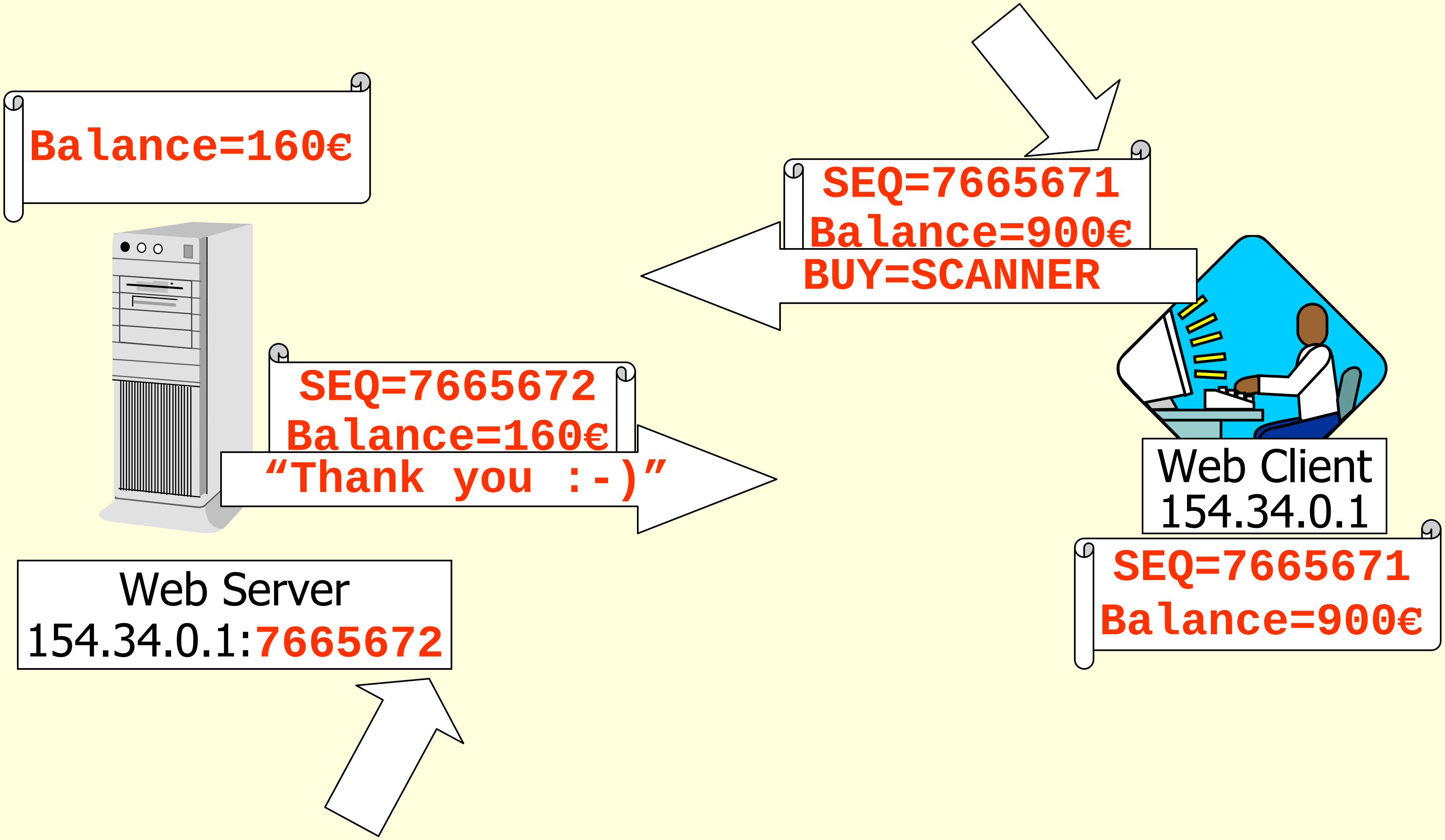
SEQ=7665672
Balance=160€
“Thank you :-)”

Web Client
154.34.0.1

SEQ=7665671
Balance=900€

Web Server
154.34.0.1:7665672

Solution: Sequence Numbers



Balance=160€

SEQ=7665671
Balance=900€
BUY=SCANNER

SEQ=7665672
Balance=160€
“Thank you :-)”

Web Client
154.34.0.1

SEQ=7665671
Balance=900€

Web Server
154.34.0.1:7665672

(Reintroduces the server side storage management problem.)

Security/Efficiency

Problem:

a malicious user may forge the continuation;

a curious user may inspect sensitive data.

Solution: encrypt/sign continuation.

Problem: continuations may be large pieces of data.

Solution: compress them.

Problem: Debugging

Executed code bears little resemblance to programmer's code. How do you debug it?

Answer: still an open question...

Ad hoc solution for PL's with `call/cc`:

- **don't preprocess;**
- **reimplement the `input` function to store continuations and send HTML to Web browser;**
- **reimplement the main function to resume current continuation.**

Literature

Paul Graunke, Shriram Krishnamurthi, Robert Bruce Findler, Matthias Felleisen (2001): *Automatically Restructuring Programs for the Web.*

Jacob Matthews, Paul Graunke, Shriram Krishnamurthi, Robert Bruce Findler, Matthias Felleisen (2004): *Automatically Restructuring Programs for the Web.*

Christian Queinnec (2000): *The influence of browsers on evaluators or, continuations to program Web servers.* In: ACM SIGPLAN International Conference on Functional Programming.

Andrew W. Appel (1992): *Compiling with Continuations.* Cambridge University Press.

Thomas Johnson (1985): *Lambda Lifting: Transforming Programs to Recursive Equations*. In: Proceedings of the Conference on Functional Programming Languages and Computer Architecture. Nancy, France.

John C. Reynolds (1972): Definitional Interpreters for Higher-Order Programming Languages. In: Proceedings of the 25th ACM National Conference. pp. 717–740.

Conclusion

Automatic restructuring of programs for the Web enables programmers to:

- **use existing paradigms and tools;**
- **structure programs in a natural way;**
- **be more productive.**

Thank you!