The RPC Calculus

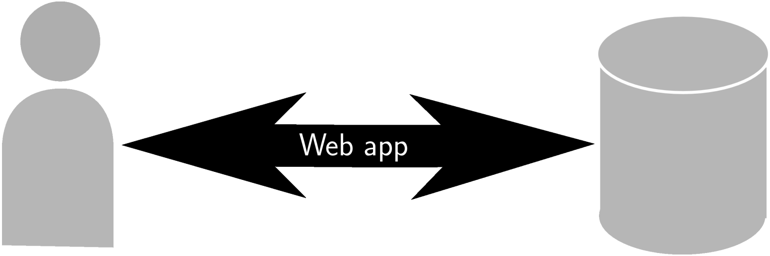
Symmetrical RPC in an Asymmetrical World

Ezra Cooper   Philip Wadler

September 8, 2009

Dream of unified web programming

Dream of unified web programming



What we want

Reality of web programming

Reality of web programming

It's a bit more fiddly.

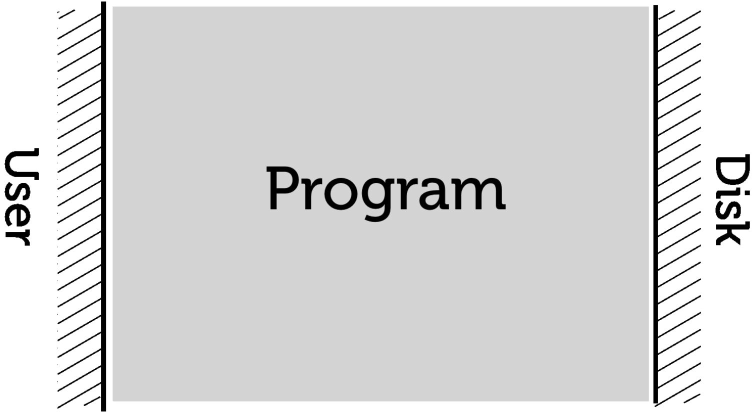
Reality of web programming

It’s a bit more fiddly.

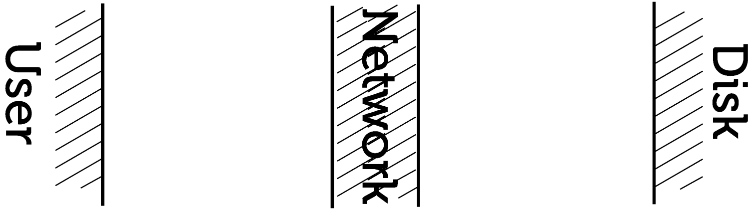
Here’s why:

Traditional program

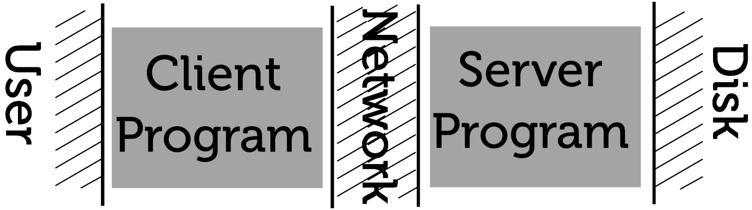
Traditional program



Traditional web program



Traditional web program



Unified program

Unified program

Simplifies the work of web programming

Unified program

Simplifies the work of web programming

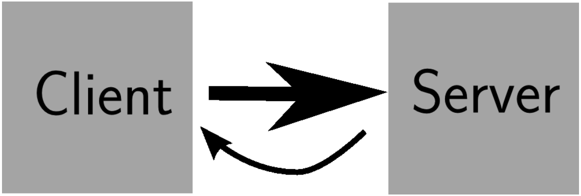
The Links language:
http://groups.inf.ed.ac.uk/links

Dream of a unified language

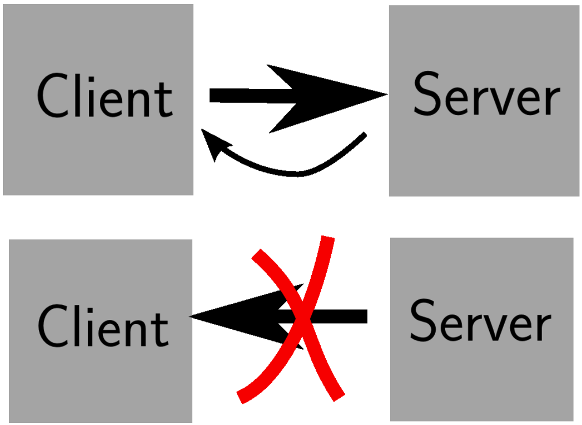
Waking up:

- Desire to control location explicitly, with a light touch;
- Need control for performance and security reasons;
- Tricky because of asymmetrical client/server relationship.

Roadblock: Asymmetrical client/server relationship



Roadblock: Asymmetrical client/server relationship



Stateless server

Web applications should not store control state at the server.

Stateless server

Web applications should not store control state at the server.

Server should encode all state and give it to client.

Stateless server

Web applications should not store control state at the server.

Server should encode all state and give it to client.

For this talk, *state* = call stack.

Example program

```
fun checkPassword(name, password) {
    # load this user's row from database & check password
    var u = lookupUser(name);
    u.password == password
}

fun validate() {
    var auth = checkPassword(fieldValue("name"),
                             fieldValue("password")));
    if (auth)
        displaySecretDocument();
    else
        displayErrorMessage();
}
```

Example located program

```
fun checkPassword(name, password) server {
  # load this user's row from database & check password
  var u = lookupUser(name);
  u.password == password
}

fun validate() client {
  var auth = checkPassword(fieldValue("name"),
                           fieldValue("password"));
  if (auth)
    displaySecretDocument();
  else
    displayErrorMessage();
}
```

Example located program: server push

```
fun findFlights(flightQuery) server {
  # Query each vendor for its own matching flights
  for (vendor ← airlines()) {
    var flights = queryVendor(vendor, flightQuery);
    # Send this vendor's flights to the browser
    displayFlights(flights);
  }
}

fun displayFlights(flights) client {
  # Add each flight to the page
  for (flight ← flights)
    addToPage(flight);
}
```

Example: higher-order functions

How should this code behave?

$$\begin{array}{l}\textbf{fun } \mathit{usernameMap}(f)\ \textbf{server}\ \{\\ \quad \textbf{var } \mathit{users} = \mathit{getUsersFromDatabase}();\\ \quad \textbf{for}\ (u \leftarrow \mathit{users})\,[f(u.\mathrm{name})]\\ \}\end{array}$$

$$\begin{array}{l}\textbf{fun } \mathit{userNameFirstThree}()\ \textbf{client}\ \{\\ \quad \mathit{usersMap}(\mathit{fun}(\mathit{name})\{\mathit{take}(3, \mathit{name})\});\\ \}\end{array}$$

Example: higher-order functions

How should this code behave?

$$
\begin{array}{l}
\textbf{fun}\ \mathit{usernameMap}(f)\ \textbf{server}\ \{ \\
\quad \textbf{var}\ \mathit{users} = \mathit{getUsersFromDatabase}(); \\
\quad \textbf{for}\ (u \leftarrow \mathit{users})\,[f(u.\mathrm{name})] \\
\} \\
\\
\textbf{fun}\ \mathit{userNameFirstThree}()\ \textbf{client}\ \{ \\
\quad \mathit{usersMap}(\mathit{fun}(\mathit{name})\{\mathit{take}(3, \mathit{name})\}); \\
\}
\end{array}
$$

▷ Functions in *lexical* client-context execute on client.

What I want to show you

- How to compile this language for the asymmetrical client-server model,

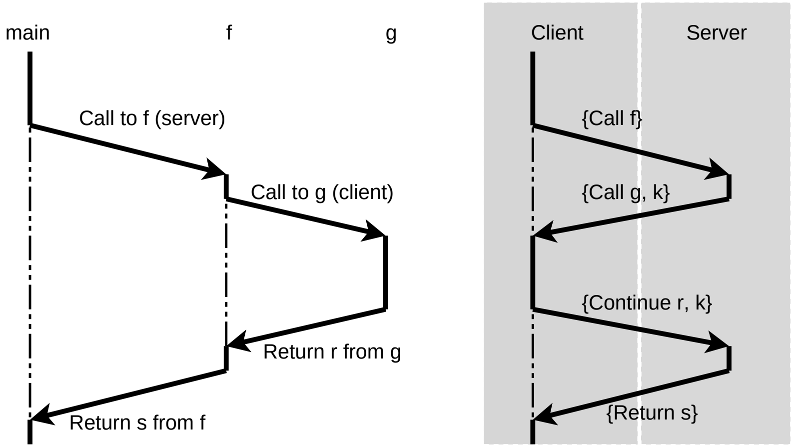
What I want to show you

- How to compile this language for the asymmetrical client-server model,
- How the compilation factors into standard techniques,

What I want to show you

- How to compile this language for the asymmetrical client-server model,
- How the compilation factors into standard techniques,
- How these these techniques can be presented formally and concisely.

How it's done



Source language:
call/return style

Implementation:
request/response style

Getting technical

Source language: the located lambda calculus

Source language: the located lambda calculus

$$\begin{aligned} L, M, N &::= LM \mid \lambda^{a} x.N \mid \lambda x.N \mid x \mid c \\ a, b &::= \mathbf{c} \mid \mathbf{s} \end{aligned}$$

Source language: the located lambda calculus

$$\begin{array}{rcl} L, M, N & ::= & LM \mid \lambda^a x.N \mid \lambda x.N \mid x \mid c \\ a, b & ::= & \mathbf{c} \mid \mathbf{s} \end{array}$$

We eliminate un-located forms $\lambda x.N$ by explicitly copying the location of their lexical context.

So $\lambda^{\mathbf{c}} x.L(\lambda y.N)$ becomes $\lambda^{\mathbf{c}} x.L(\lambda^{\mathbf{c}} y.N)$

Source language: the located lambda calculus

$$
\begin{array}{rcl}
L, M, N & ::= & L M \mid \lambda^{a} x. N \mid x \mid c \\
a, b & ::= & \mathbf{c} \mid \mathbf{s}
\end{array}
$$

Semantics

Read $M \Downarrow_a V$ as "M evaluates, starting at a, to V."

$$V \Downarrow_a V \qquad \text{(Value)}$$

$$\frac{L \Downarrow_a \lambda^b x.N \qquad M \Downarrow_a W \qquad N\{W/x\} \Downarrow_b V}{LM \Downarrow_a V} \qquad \text{(Beta)}$$

$$\frac{L \Downarrow_a c \qquad M \Downarrow_a W \qquad \delta_a(c, W) \Downarrow_a V}{LM \Downarrow_a V} \qquad \text{(Delta)}$$

Translation to a client-server system

Three techniques:

- CPS translation:
  reifies the control state
- Defunctionalization:
  turns higher-order functions into data (serializable)
- Trampolining:
  inverts control, so state resides at client.

CPS translation (due to Fischer, 1972, via Sabry and Wadler, 1997)

Source:

$$
\begin{array}{rcl}
L, M, N & ::= & LM \mid V \\
V & ::= & \lambda x.N \mid x
\end{array}
$$

CPS translation:

$$
\begin{array}{rcl}
(LM)^\dagger K & = & L^\dagger(\lambda f.M^\dagger(\lambda x.fxK)) \\
V^\dagger K & = & KV^\circ
\end{array}
$$

$$
\begin{array}{rcl}
(\lambda x.N)^\circ & = & \lambda x.\lambda k.N^\dagger k \\
x^\circ & = & x
\end{array}
$$

Defunctionalization

Defunctionalization target

$$
\begin{array}{rcl}
\mathcal{D} & ::= & \textbf{letrec}\ D\ \textbf{and}\ \cdots\ \textbf{and}\ D \\
D & ::= & f(\vec{x}) = \textbf{case}\ x\ \textbf{of}\ \mathcal{A} \\
\mathcal{A} & ::= & \text{a set of } A \text{ items} \\
A & ::= & F(\vec{c}) \Rightarrow M \\
M & ::= & f(\vec{M}) \mid F(\vec{M}) \mid x \mid c
\end{array}
$$

Defunctionalization target

$$
\begin{array}{rcl}
\mathcal{D} & ::= & \textbf{letrec}\ D\ \textbf{and}\ \cdots\ \textbf{and}\ D \\
D & ::= & f(\vec{x}) = \textbf{case}\ x\ \textbf{of}\ \mathcal{A} \\
\mathcal{A} & ::= & \text{a set of } A \text{ items} \\
A & ::= & F(\vec{c}) \Rightarrow M \\
M & ::= & f(\vec{M}) \mid F(\vec{M}) \mid x \mid c
\end{array}
$$

$$
\begin{array}{rl}
\text{function names} & f, g \\
\text{constructor names} & F, G
\end{array}
$$

Defunctionalization (orig. Reynolds, 1972)

$$
\begin{array}{rcl}
[\![M]\!]^{\text{top}} & = & \textbf{letrec}\ \mathit{apply}(\mathit{fun}, \mathit{arg}) = \textbf{case}\ \mathit{fun}\ \textbf{of}\ [\![M]\!]^{\text{fun}^*} \\
 & & \quad \textbf{in}\ [\![M]\!] \\
[\![\lambda x.N]\!]^{\text{fun}} & = & \ulcorner \lambda x.N \urcorner(\vec{y}) \Rightarrow [\![N]\!]\{\mathit{arg}/x\} \\
 & & \hfill \text{where } \vec{y} = \mathrm{FV}(\lambda x.N) \\
 & & \\
[\![LM]\!] & = & \mathit{apply}([\![L]\!], [\![M]\!]) \\
[\![V]\!] & = & V^{\circ} \\
 & & \\
(\lambda x.N)^{\circ} & = & \ulcorner \lambda x.N \urcorner(\vec{y}) \qquad \text{where } \vec{y} = \mathrm{FV}(\lambda x.N) \\
x^{\circ} & = & x
\end{array}
$$

The operation $\ulcorner M \urcorner$ gives an opaque identifier for the term M.

Trampolining (due to Ganz, Friedman and Wand)

- Continually returns control to a top-level *trampoline*;
- Works on any tail-form program,
  including CPS programs;
- Choice of the trampoline modifies the behavior.

Trampolining

$$
\begin{array}{rcl}
(LM)^{\mathrm{T}} & = & \mathit{Bounce}(\lambda z.L^{\mathrm{t}} M^{\mathrm{t}}) \\
 & & \qquad\qquad\text{(where } z \text{ is a dummy)} \\
V^{\mathrm{T}} & = & \mathit{Return}(V^{\mathrm{t}}) \\
 & & \\
(\lambda x.N)^{\mathrm{t}} & = & \lambda x.N^{\mathrm{T}} \\
x^{\mathrm{t}} & = & x
\end{array}
$$

Behavior depends on our choice of *tramp*.

Example trampolines

Trivial trampoline:

$$\begin{aligned} \mathit{tramp}(x) = \textbf{case } x \textbf{ of} \quad & \mathit{Bounce}(\mathit{thunk}) \Rightarrow \mathit{tramp}(\mathit{thunk}()) \\ \mid\; & \mathit{Return}(x) \Rightarrow x \end{aligned}$$

Example trampolines

Trivial trampoline:

$$
\begin{aligned}
\mathit{tramp}(x) = \textbf{case } x \textbf{ of} \quad & \\
& \mathit{Bounce}(\mathit{thunk}) \Rightarrow \mathit{tramp}(\mathit{thunk}()) \\
\mid\ & \mathit{Return}(x) \Rightarrow x
\end{aligned}
$$

Step-counting trampoline:

$$
\begin{aligned}
\mathit{tramp}(n, x) = \textbf{case } x \textbf{ of} \quad & \\
& \mathit{Bounce}(\mathit{thunk}) \Rightarrow \mathit{print}(n); \mathit{tramp}(n + 1, \mathit{thunk}()) \\
\mid\ & \mathit{Return}(x) \Rightarrow x
\end{aligned}
$$

Our trampoline

Since the control state is reified, *tramp* can split the computation into a client- and a server-side piece.

$$
\begin{array}{rcl}
\mathit{tramp}(x) & = & \textbf{case } x \textbf{ of} \\
& & \quad \mid \mathit{Bounce}(f, x, k) \Rightarrow \\
& & \qquad \mathit{tramp}(\textbf{req } \mathit{cont}\ (k, \mathit{apply}(f, x))) \\
& & \quad \mid \mathit{Return}(x) \Rightarrow x
\end{array}
$$

(This shouldn't make sense yet; don't worry.)

Our trampoline

Since the control state is reified, *tramp* can split the computation into a client and a server-side piece.

$$
\begin{array}{rcl}
\mathit{tramp}(x) & = & \textbf{case } x \textbf{ of} \\
& & \quad \mid \mathit{Call}(f, x, k) \Rightarrow \\
& & \qquad \mathit{tramp}(\textbf{req } \mathit{cont}\ (k, \mathit{apply}(f, x))) \\
& & \quad \mid \mathit{Return}(x) \Rightarrow x
\end{array}
$$

(This shouldn't make sense yet; don't worry.)

The Big Transformation

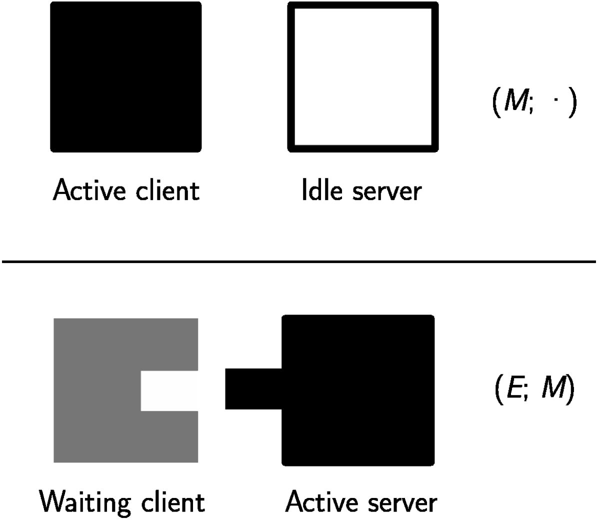
First, the target: first-order client-server calculus

The client-server calculus

Syntax

$$
\begin{array}{rrcl}
\text{configurations} & \mathcal{K} & ::= & (M;\ \cdot) \mid (E;\ M) \\
\text{terms} & L, M, N & ::= & x \mid c \mid F(\vec{M}) \mid f(\vec{M}) \mid \mathbf{req}\ f\ (\vec{M}) \\
\text{definition set} & \mathcal{D}, \mathcal{C}, \mathcal{S} & ::= & \mathbf{letrec}\ D\ \mathbf{and} \cdots \mathbf{and}\ D \\
\text{function definitions} & D & ::= & f(\vec{x}) = \mathbf{case}\ M\ \mathbf{of}\ \mathcal{A} \\
\text{alternative sets} & \mathcal{A} & & \text{a set of } A \text{ items} \\
\text{case alternatives} & A & ::= & F(\vec{x}) \Rightarrow M \\
\text{function names} & f, g & & \\
\text{constructor names} & F, G & &
\end{array}
$$

Configurations of the machine



Semantics

Client:

$$
\begin{array}{rcl}
(E[f(\vec{V})];\cdot) & \longrightarrow_{\mathcal{C},\mathcal{S}} & (E[M\{\vec{V}/\vec{x}\}];\cdot) \\
& & \quad \text{if } (f(\vec{x}) = M) \in \mathcal{C} \\
(E[\textbf{case } (F(\vec{V})) \textbf{ of } \mathcal{A}];\cdot) & \longrightarrow_{\mathcal{C},\mathcal{S}} & (E[M\{\vec{V}/\vec{x}\}];\cdot) \\
& & \quad \text{if } (F(\vec{x}) \Rightarrow M) \in \mathcal{A}
\end{array}
$$

Server:

$$
\begin{array}{rcl}
(E;\ E'[f(\vec{V})]) & \longrightarrow_{\mathcal{C},\mathcal{S}} & (E;\ E'[M\{\vec{V}/\vec{x}\}]) \\
& & \quad \text{if } (f(\vec{x}) = M) \in \mathcal{S} \\
(E;\ E'[\textbf{case } (F(\vec{V})) \textbf{ of } \mathcal{A}]) & \longrightarrow_{\mathcal{C},\mathcal{S}} & (E;\ E'[M\{\vec{V}/\vec{x}\}]) \\
& & \quad \text{if } (F(\vec{x}) \Rightarrow M) \in \mathcal{A}
\end{array}
$$

Communication:

$$
\begin{array}{rcl}
(E[\textbf{req } f\ (\vec{V})];\ \cdot) & \longrightarrow_{\mathcal{C},\mathcal{S}} & (E;\ f(\vec{V})) \\
(E;\ V) & \longrightarrow_{\mathcal{C},\mathcal{S}} & (E[V];\ \cdot)
\end{array}
$$

Now, the translation

Transformation on terms

$$
\begin{aligned}
(\lambda^a x.N)^\circ &= \ulcorner \lambda^a x.N \urcorner(\vec{y}) && \vec{y} = \mathrm{FV}(\lambda^a x.N) \\
x^\circ &= x \\
c^\circ &= c
\end{aligned}
$$

$$
\begin{aligned}
V^* &= V^\circ \\
(LM)^* &= \mathit{apply}(L^*,\ M^*)
\end{aligned}
$$

$$
\begin{aligned}
V^\dagger[\ ] &= \mathit{cont}([\ ],\ V^\circ) \\
(LM)^\dagger[\ ] &= L^\dagger(\ulcorner M \urcorner(\vec{y}, [\ ])) && \text{where } \vec{y} = \mathrm{FV}(M)
\end{aligned}
$$

Transformation to definitions (client-side)

$$
\begin{aligned}
[\![M]\!]^{\mathsf{c},\text{top}} \quad = \quad & \textbf{letrec}\, \mathit{apply}(\mathit{fun}, \mathit{arg}) = \textbf{case}\ \mathit{fun}\ \textbf{of}\ [\![M]\!]^{\mathsf{c},\text{fun}} \\
& \textbf{and}\ \mathit{tramp}(x) = \textbf{case}\ x\ \textbf{of} \\
& \qquad \mid \mathit{Call}(f, x, k) \Rightarrow \\
& \qquad\qquad \mathit{tramp}(\textbf{req}\ \mathit{cont}\ (k, \mathit{apply}(f, x))) \\
& \qquad \mid \mathit{Return}(x) \Rightarrow x
\end{aligned}
$$

$$
\begin{aligned}
[\![\lambda^{\mathsf{c}} x.N]\!]^{\mathsf{c},\text{fun}} \quad = \quad & \ulcorner \lambda^{\mathsf{c}} x.N \urcorner(\vec{y}) \Rightarrow N^{*}\{\mathit{arg}/x\} \\
& \qquad\qquad\qquad\qquad \text{where } \vec{y} = \mathrm{FV}(\lambda x.N) \\
[\![\lambda^{\mathsf{s}} x.N]\!]^{\mathsf{c},\text{fun}} \quad = \quad & \\
\ulcorner \lambda^{\mathsf{s}} x.N \urcorner(\vec{y}) \Rightarrow \ & \mathit{tramp}(\textbf{req}\ \mathit{apply}\ (\ulcorner \lambda^{\mathsf{s}} x.N \urcorner(\vec{y}), \mathit{arg}, \mathit{Fin}())) \\
& \qquad\qquad\qquad\qquad \text{where } \vec{y} = \mathrm{FV}(\lambda x.N)
\end{aligned}
$$

Transformation to definitions (server-side)

$$
\begin{aligned}
[\![M]\!]^{\mathbf{s},\mathsf{top}} \;=\; & \mathbf{letrec}\; apply(fun,\, arg,\, k) = \mathbf{case}\; fun\; \mathbf{of}\; [\![M]\!]^{\mathbf{s},\mathsf{fun}} \\
& \mathbf{and}\; cont(k,\, arg) = \mathbf{case}\; k\; \mathbf{of} \\
& \qquad\qquad [\![M]\!]^{\mathbf{s},\mathsf{cont}} \\
& \qquad\quad \mid App(fun,\, k) \Rightarrow apply(fun,\, arg,\, k) \\
& \qquad\quad \mid Fin() \Rightarrow Return(arg)
\end{aligned}
$$

$$
\begin{aligned}
[\![\lambda^{\mathbf{s}} x.N]\!]^{\mathbf{s},\mathsf{fun}} &= \ulcorner \lambda^{\mathbf{s}} x.N \urcorner(\vec{y}) \Rightarrow (N^{\dagger} k)\{arg/x\} \\
&\qquad\qquad\qquad \text{where } \vec{y} = \mathrm{FV}(\lambda x.N) \\
[\![\lambda^{\mathbf{c}} x.N]\!]^{\mathbf{s},\mathsf{fun}} &= \ulcorner \lambda^{\mathbf{c}} x.N \urcorner(\vec{y}) \Rightarrow Call(\ulcorner \lambda^{\mathbf{c}} x.N \urcorner(\vec{y}),\, arg,\, k) \\
&\qquad\qquad\qquad \text{where } \vec{y} = \mathrm{FV}(\lambda x.N) \\
[\![LM]\!]^{\mathbf{s},\mathsf{cont}} &= \ulcorner M \urcorner(\vec{y}, k) \Rightarrow M^{\dagger}(App(arg, k)) \\
&\qquad\qquad\qquad \text{where } \vec{y} = \mathrm{FV}(M)
\end{aligned}
$$

Correctness: Bisimulation

$$
\begin{tikzcd}
M \arrow[r, "\Downarrow"] \arrow[d, dash] & V \arrow[d, dash, dashed] \\
M' \arrow[r, dashed, "\longrightarrow\!\!\!\!\rightarrow"] & V'
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
M \arrow[r, dashed, "\Downarrow"] \arrow[d, dash] & V \arrow[d, dash, dashed] \\
M' \arrow[r, "\longrightarrow\!\!\!\!\rightarrow"] & V'
\end{tikzcd}
$$

Summary

We can

Summary

We can

- Enrich a functional programming language with *location annotations*,

Summary

We can

- Enrich a functional programming language with *location annotations*,
- which designate execution location of their contents lexically,

Summary

We can

- Enrich a functional programming language with *location annotations*,
- which designate execution location of their contents lexically,
- and whose semantics are straightforward,

Summary

We can

- Enrich a functional programming language with *location annotations*,
- which designate execution location of their contents lexically,
- and whose semantics are straightforward,
- and we can execute these programs on an asymmetrical client-server system with a “stateless server”

Summary

We can

- Enrich a functional programming language with *location annotations*,
- which designate execution location of their contents lexically,
- and whose semantics are straightforward,
- and we can execute these programs on an asymmetrical client-server system with a "stateless server"
- using a combination of classic transformations.

Summary

We can

- Enrich a functional programming language with *location annotations*,
- which designate execution location of their contents lexically,
- and whose semantics are straightforward,
- and we can execute these programs on an asymmetrical client-server system with a "stateless server"
- using a combination of classic transformations.

Thank you