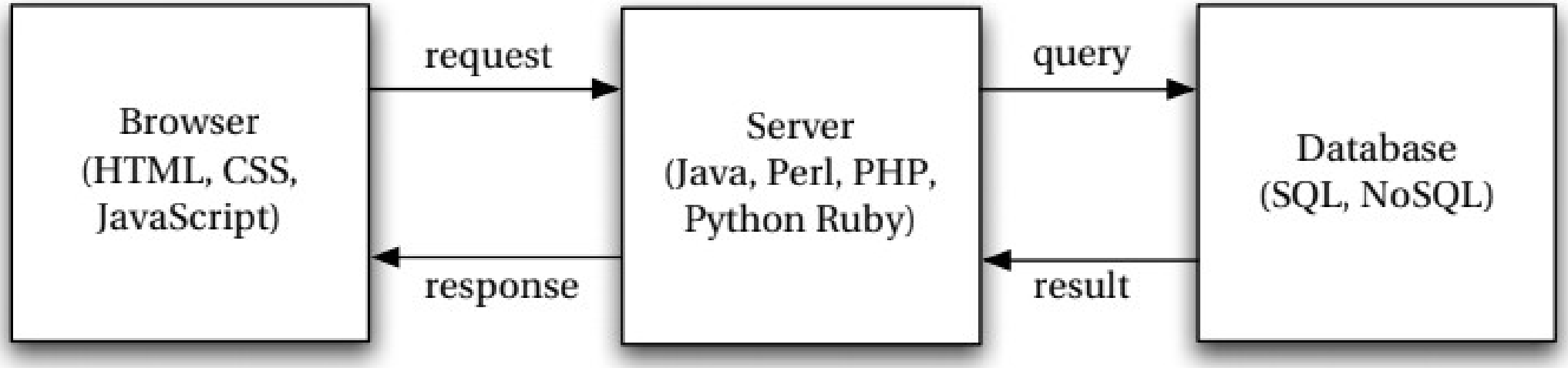
The Links Programming Language

Simon Fowler

PL4DS, Shonan



3 *tiers* of web applications, *each with their own languages*.

- Users have to learn three languages!
- Marshalling data is cumbersome and error-prone
- No cross-tier guarantees of correctness

Links: Web Programming Without Tiers*

Ezra Cooper, Sam Lindley, Philip Wadler, and Jeremy Yallop

University of Edinburgh

Abstract. Links is a programming language for web applications that generates code for all three tiers of a web application from a single source, compiling into JavaScript to run on the client and into SQL to run on the database. Links supports rich clients running in what has been dubbed *'Ajax'* style, and supports concurrent processes with statically-typed message passing. Links is *scalable* in the sense that session state is preserved in the client rather than the server, in contrast to other approaches such as Java Servlets or PLT Scheme. Client-side concurrency in JavaScript and transfer of computation between client and server are both supported by translation into continuation-passing style.

Links: First uniform language for client, server, database programming

...but it has come on a long way since 2006!

Links

http://www.links-lang.org

DATABASE INTEGRATION

- Relational Lenses
- Query Shredding
- Language-Integrated query
- Provenance

WEB DEVELOPMENT

- Typed HTML + antiquotes
- Form Abstractions
- Model-View-Update

CONCURRENCY & DISTRIBUTION

- RPC Calculus
- Distributed Session Types
- Session Exceptions

INTERACTIVE PROGRAMMING

- TryLinks

EFFECT HANDLERS

- CEK Machine (Server)
- CPS Translation (Client)
- Row-based Effects

From RPC to Session Types

RPC

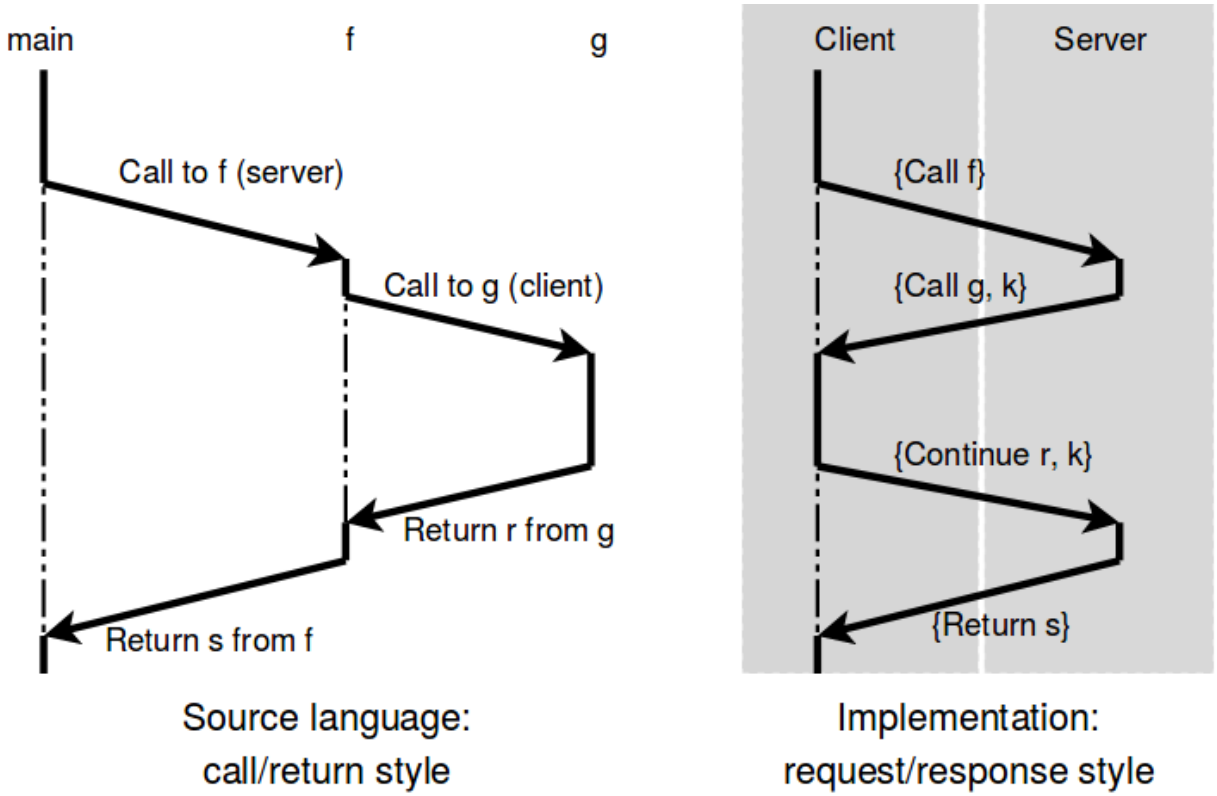
```
sig getServerYear : () ~> Int
fun getServerYear() server {
  intToDate(serverTime()).year
}
```

```
fun mainPage() {
  var _  = spawnClient {
    replacePlaceholder(getServerYear())
  };
  page
    <html><body>
      <div id="placeholder"></div>
    </body></html>
}
```

`getServerYear`: Function defined **only on server**
`mainPage`: Run on server, to generate page to serve to client.

- Spawns client process, which calls `getServerYear`
- Request sent to server, which returns year
- Placeholder element replaced with year

RPC

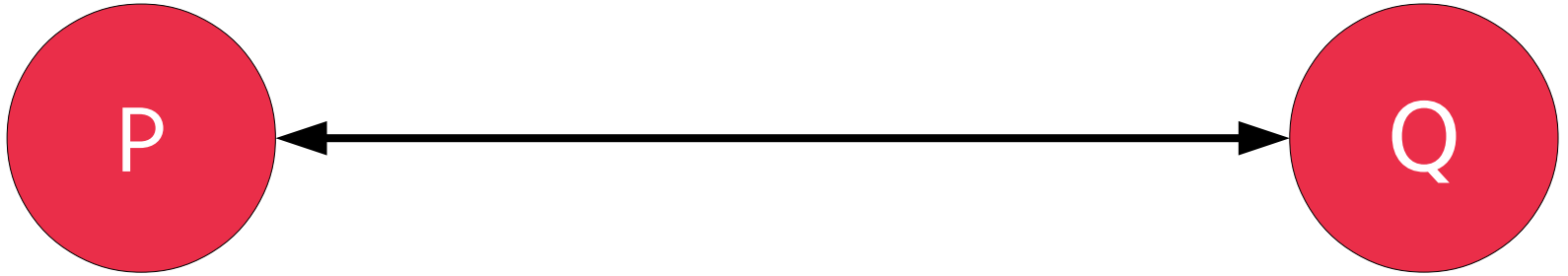


Source language:
call/return style

Implementation:
request/response style

Original Links RPC: Based on continuation-passing style, designed for repeated invocations of CGI-based server.

Session Types



`!Int.!Int.?Bool.End` `?Int.?Int.!Bool.End`

Session types: Type system for communication channel endpoints
Endpoints are **dual**, ensuring communication matches

Session Types

```
typename EqualityClient = !Int.!Int.?Bool.End;

sig equalityClient : (EqualityClient) ~> Bool
fun equalityClient(s) {
  var s = send(5, s);
  var s = send(5, s);
  var (res, s) = receive(s);
  close(s); res
}
```

Session Types

```
typename EqualityClient = !Int.!Int.?Bool.End;

sig equalityClient : (EqualityClient) ~> Bool
fun equalityClient(s) {
  var s = send(5, s);
  var s = send(“Hello!”, s);
  var (res, s) = receive(s);
  close(s); res
}
```

```
eqclientwrong.links:6: Type error: The function
    `send'
has type
    `(String, !(String).a::Session) ~b~> a::Session'
while the arguments passed to it have types
    `String'
and
    `!(Int).?(Bool).End'
and the currently allowed effects are
    `|wild|c'
In expression: send("Hello", s).
```

Session Types

```
typename EqualityClient = !Int.!Int.?Bool.End;

sig equalityClient : (EqualityClient) ~> Bool
fun equalityClient(s) {
  var s = send(5, s);
  var s = send(5, s);
  var (res, t) = receive(s);
  var (res, s) = receive(s);
  close(t); close(s); res
}
```

```
eqclientwrong.links:6: Type error: Variable s has
linear type
    `?(Bool).End'
but is used 2 times.
In expression: var s = send(5, s);.
```

Session Types

```
typename EqualityClient = !Int.!Int.?Bool.End;

sig equalityClient : (EqualityClient) ~> Bool
fun equalityClient(s) {
  var s = send(5, s);
  var s = send(5, s);
  true
}
```

```
eqclientwrong.links:6: Type error: Variable s has
linear type
    `?(Bool).End'
but is used 0 times.
In expression: var s = send(5, s);.
```

RPC using Sessions

```
typename YearServer = ?().!Int.End;
typename YearClient = ~YearServer;

fun servePage() {
  var c = fork(handleRequest);
  mainPage(c)
}

fun handleRequest(s) {
  var (_, s) = receive(s);
  var year = intToDate(serverTime()).year;
  var s = send(year, s);
  close(s)
}
```

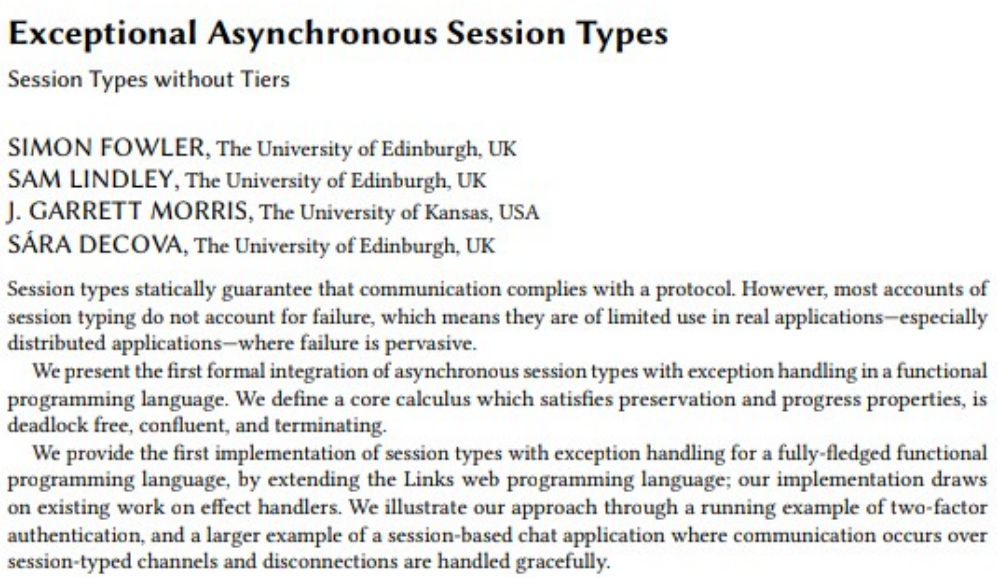
```
fun mainPage(c) {
  var _ = spawnClient {
    var c = send((), c);
    var (year, c) = receive(c);
    close(c);
    replacePlaceholder(year)
  };

  page
    <html><body>
      <div id="placeholder"></div>
    </body></html>
}
```

servePage: Run on server, to generate page to serve to client.

- Fork: Creates server process and channel; waits for (), and responds with year
- Client process sends (), waits for year, updates page

What if a client disconnects?



Exceptional Asynchronous Session Types

Session Types without Tiers

SIMON FOWLER, The University of Edinburgh, UK
SAM LINDLEY, The University of Edinburgh, UK
J. GARRETT MORRIS, The University of Kansas, USA
SÁRA DECOVA, The University of Edinburgh, UK

Session types statically guarantee that communication complies with a protocol. However, most accounts of session typing do not account for failure, which means they are of limited use in real applications—especially distributed applications—where failure is pervasive.

We present the first formal integration of asynchronous session types with exception handling in a functional programming language. We define a core calculus which satisfies preservation and progress properties, is deadlock free, confluent, and terminating.

We provide the first implementation of session types with exception handling for a fully-fledged functional programming language, by extending the Links web programming language; our implementation draws on existing work on effect handlers. We illustrate our approach through a running example of two-factor authentication, and a larger example of a session-based chat application where communication occurs over session-typed channels and disconnections are handled gracefully.

If a client disconnects (web page closed), server thread may wait forever!

- **Exceptional GV**
  - Core calculus extending linear lambda-calculus with exceptions, asynchronous session types, ability to “drop” channels

- **Implementation in Links**
  - First *implementation* combining session types & exceptions in a functional language

Conclusion

Links: Functional language for web application development

This talk: Overview of modern Links, with support for distributed session channels

Future work:

- *N*-tiers
- Functional multiparty session types
- (Ongoing): Integration with Model-View-Update (Elm) architecture













Core Links team, past and present:

Ezra Cooper, James Cheney, Frank Emrich, Stefan Fehrenbach, Simon Fowler, Daniel Hillerström, Rudi Horn, Sam Lindley, Garrett Morris, Wilmer Ricciotti, Jeremy Yallop