A compiler providing incremental scalability for web application

Etienne Brodu
etienne.brodu@insa-lyon.fr

Stéphane Frénot
stephane.frenot@insa-lyon.fr

Fabien Cellier
fabien.cellier@worldline.com

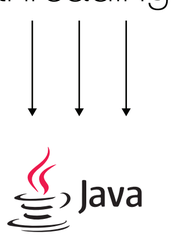
Frédéric Oblé
frederic.oble@worldline.com

To develop a **web application**, one have to **choose**

features
scalability

Monolithic models

threading



Java

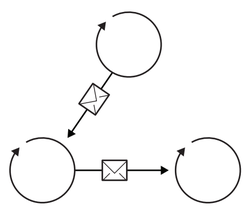
event-loop

node.js

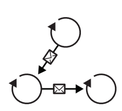
or

performance
scalability

Distributed models



SEDA[4], **System S**[3], **Storm**

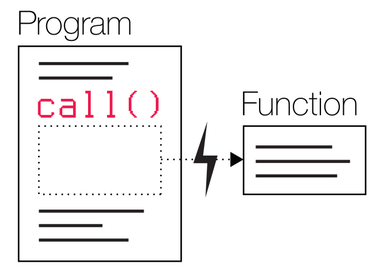
We want to **compile**  into 

The compiler extracts **autonomous parts** by searching for **rupture points** marking them out

A **Rupture point** is a call of a loosely coupled function

In node.js, an **asynchronous call** is a **rupture point**.

I/O Operation like `fs` and `express`.



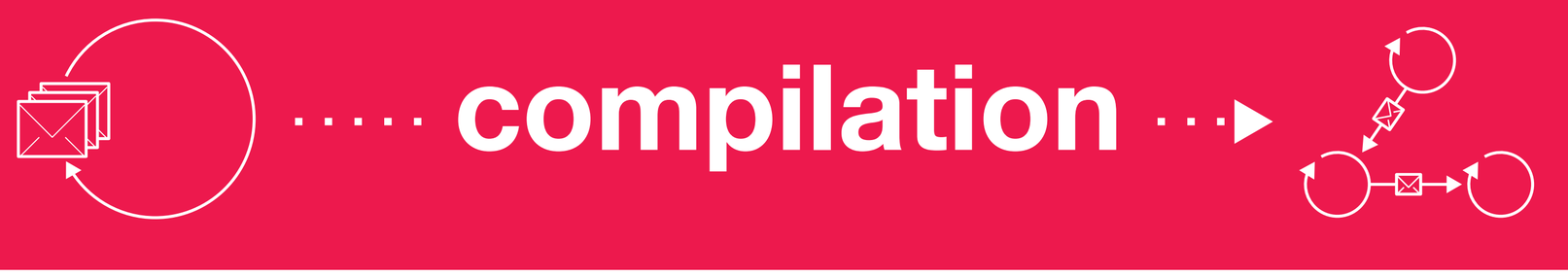

```
var app = require('express')();
app.get('/', function handler(req, res){
  fs.readFile(__filename, reply);
});
```

Application parts are enveloped into **fluxions**.

A **fluxion** is an autonomous with :

```
flx handler-1000 {fs}
-> reply-1001 <
  function handler(req, res) {<
    fs.readFile(..., -> reply-1001);<
  }
```

+ a unique **name**,
+ a persisted **memory**,
+ and a **function**
+ which outputs **streams**



source[1]

```
var app = require('express')(),
    fs = require('fs'),
    count = 0;

app.get('/', function handler(req, res){
  fs.readFile(__filename, function reply(err, data){
    count += 1;
    var code = ('' + data)
                    .replace(/\n/g, '<br>')
                    .replace(/ /g, ' ');
    res.send(err
           || 'downloaded ' + count +
              ' times<br><br><code>' +
              code + '</code>');
  });
});

app.listen(8080);
console.log('>> listening 8080');
```

target[1]

```
flx source.js {}
>> handler-1000 [res]
  var app = require('express')(),
      fs = require('fs'),
      count = 0;
  app.get('/', >> handler-1000);
  app.listen(8080);
  console.log('>> listening 8080');
```

req
res

```
flx handler-1000 {fs}
-> reply-1001 [res]
  function handler(req, res) {
    fs.readFile(__filename, -> reply-1001);
  }
```

res
err
data

```
flx reply-1001 {count}
-> null
  function reply(err, data) {
    count += 1;
    var code = ('' + data)
                    .replace(/\n/g, '<br>')
                    .replace(/ /g, ' ');
    res.send(err
           || 'downloaded ' + count +
              ' times<br><br><code>' +
              code + '</code>');
  }
```


[1] flx-example: https://github.com/etnbrd/flx-example/tree/1.0. Accessed: 2014-08-22.

[2] Fox, A. et al. 1997. Cluster-based scalable network services.

[3] Jain, N. et al. 2006. Design, implementation, and evaluation of the linear road benchmark on the stream processing core. SIGMOD '06 Proceedings of the 2006 ACM SIGMOD international conference on Management of data.

[4] Welsh, M. et al. 2000. A design framework for highly concurrent systems.

worldline e-payment services

Inria INVENTORS FOR THE DIGITAL WORLD

citi lab

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

Région RhôneAlpes