Accessor Tutorial

Bringing Sanity to IoT's use of Callbacks

Edward A. Lee

Robert S. Pepper Distinguished Professor
UC Berkeley

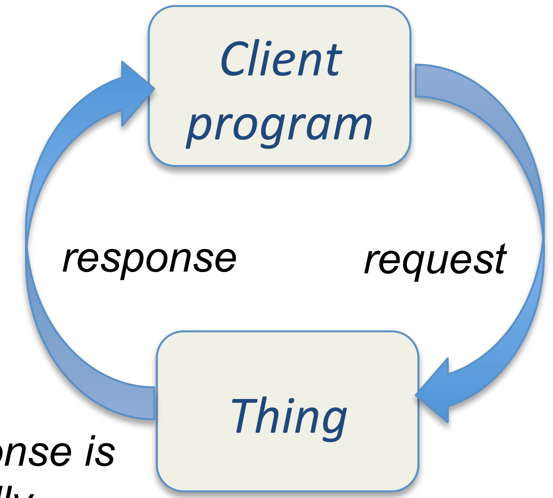
TerraSwarm E-Workshop

February 24, 2016 – Berkeley, CA

A Common Design Pattern: Asynchronous Atomic Callbacks (AAC)

Asynchronous Atomic Callbacks (AAC) (also called the *Reactor Pattern*) is a pattern where short atomic actions are interleaved with atomic invocation of response handlers.

In the Web, AAC is widely used. It is central to many popular internet programming frameworks such as Node.js & Vert.x, and to CPS frameworks such as TinyOS.



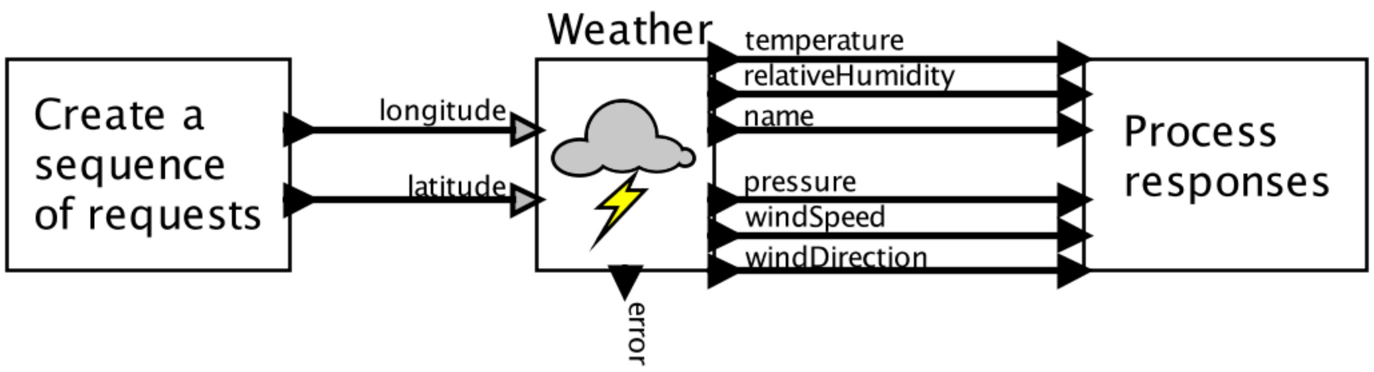
Response is typically asynchronous to avoid blocking the client program.

Response handler executes atomically.

Request encodes all state info (credentials, commands, etc.)

Another Common Design Pattern: *Actors*

Streaming requests:

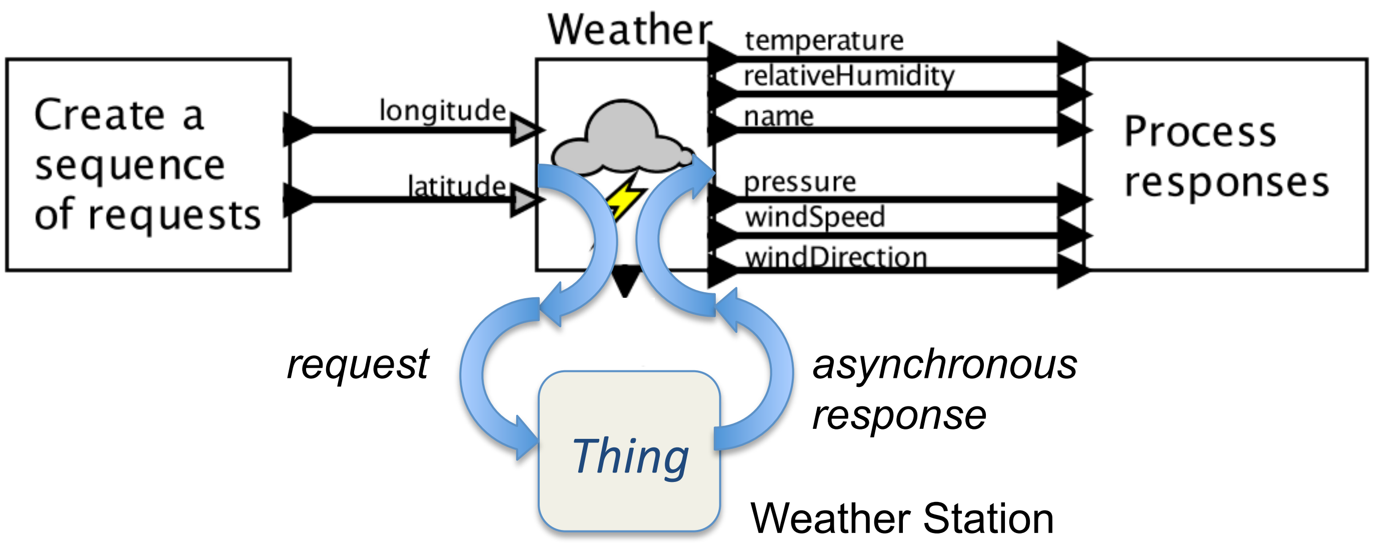


Sequence of requests for a service (a *stream*) triggers a sequence of responses.

Actors embrace concurrency and scale well.

Actors and AAC

Streaming requests:

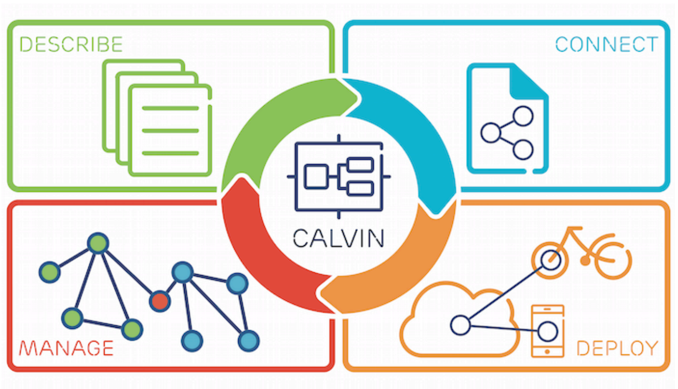


This is the essence of *accessors*, a design pattern for IoT that embraces concurrency, asynchrony, and atomicity.

We are not alone pursuing this approach

Notable efforts:

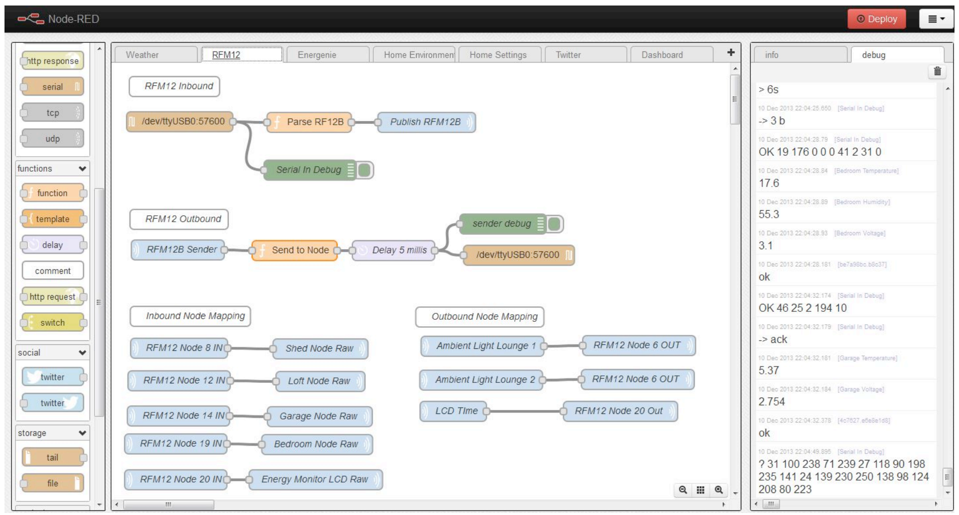
- Node Red (IBM)
- Calvin (Ericsson)



Our emphasis is on rigorous contracts for interactions.

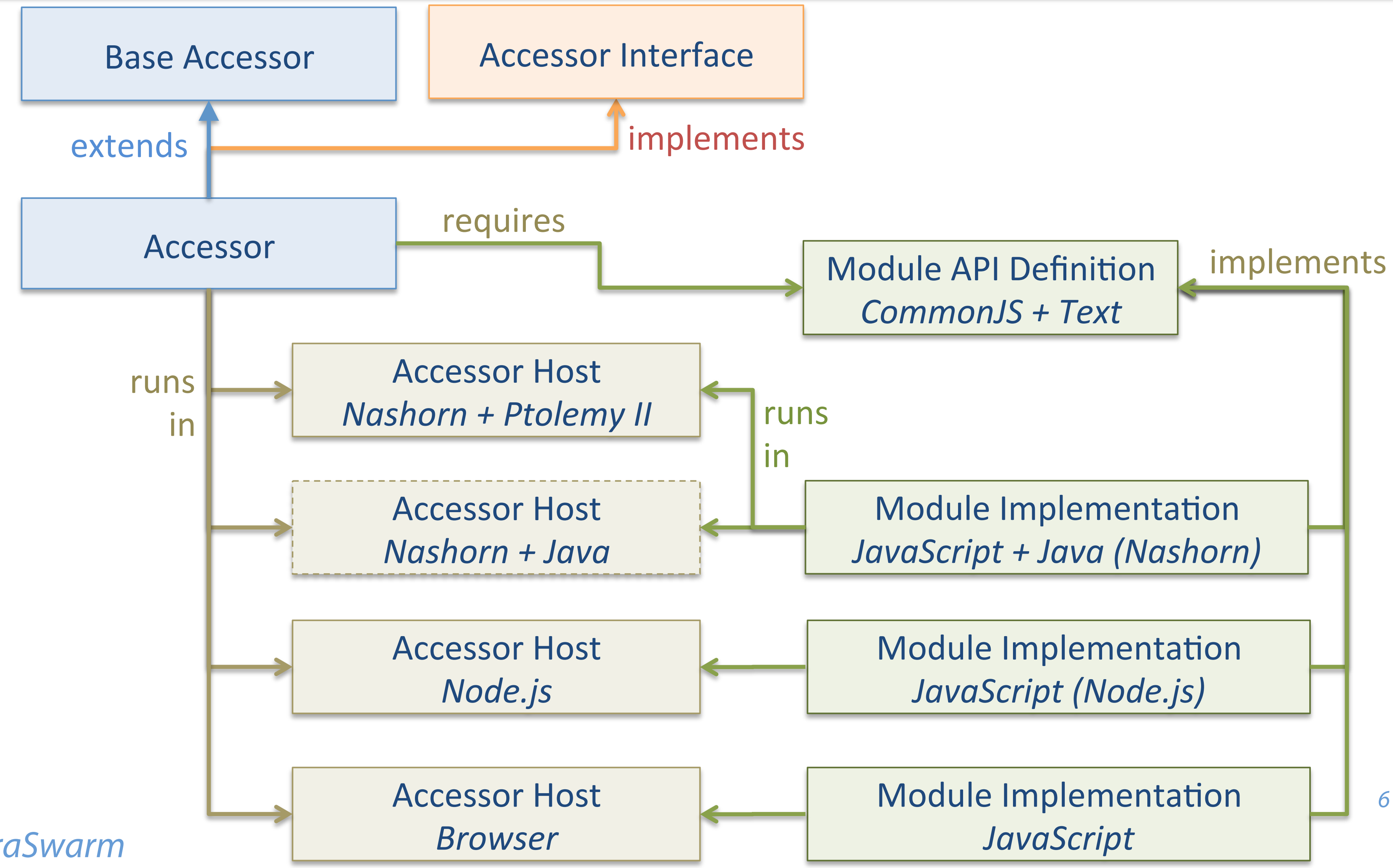
Node-RED

From: “Home Automation with Node Red, JeeNodes and Open Energy Monitor,” Dom Bramley's Blog of Maximo and the 'Internet of Things’, IBM Developer Works, Dec., 2013.



```
> 6s
-> 3 b
OK 19 176 0 0 0 41 2 31 0
17.6
55.3
3.1
ok
OK 46 25 2 194 10
-> ack
5.37
2.754
ok
? 31 100 238 71 239 27 118 90 198
235 141 24 139 230 250 138 98 124
208 80 223
```

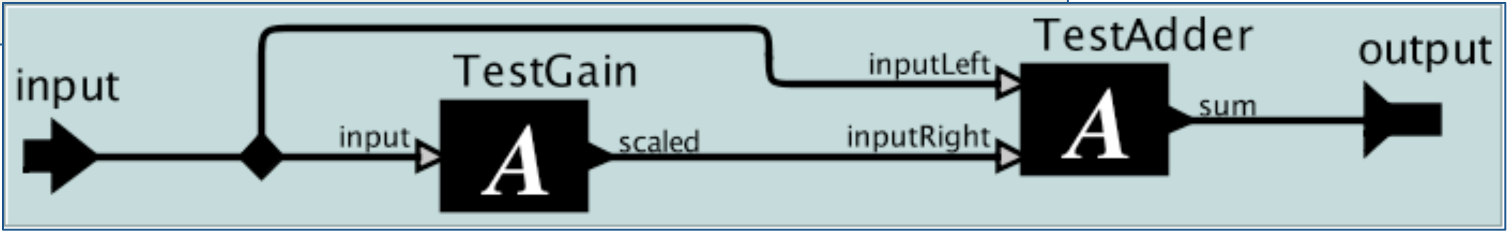
Accessor Architecture Version 1.0
http://accessors.org



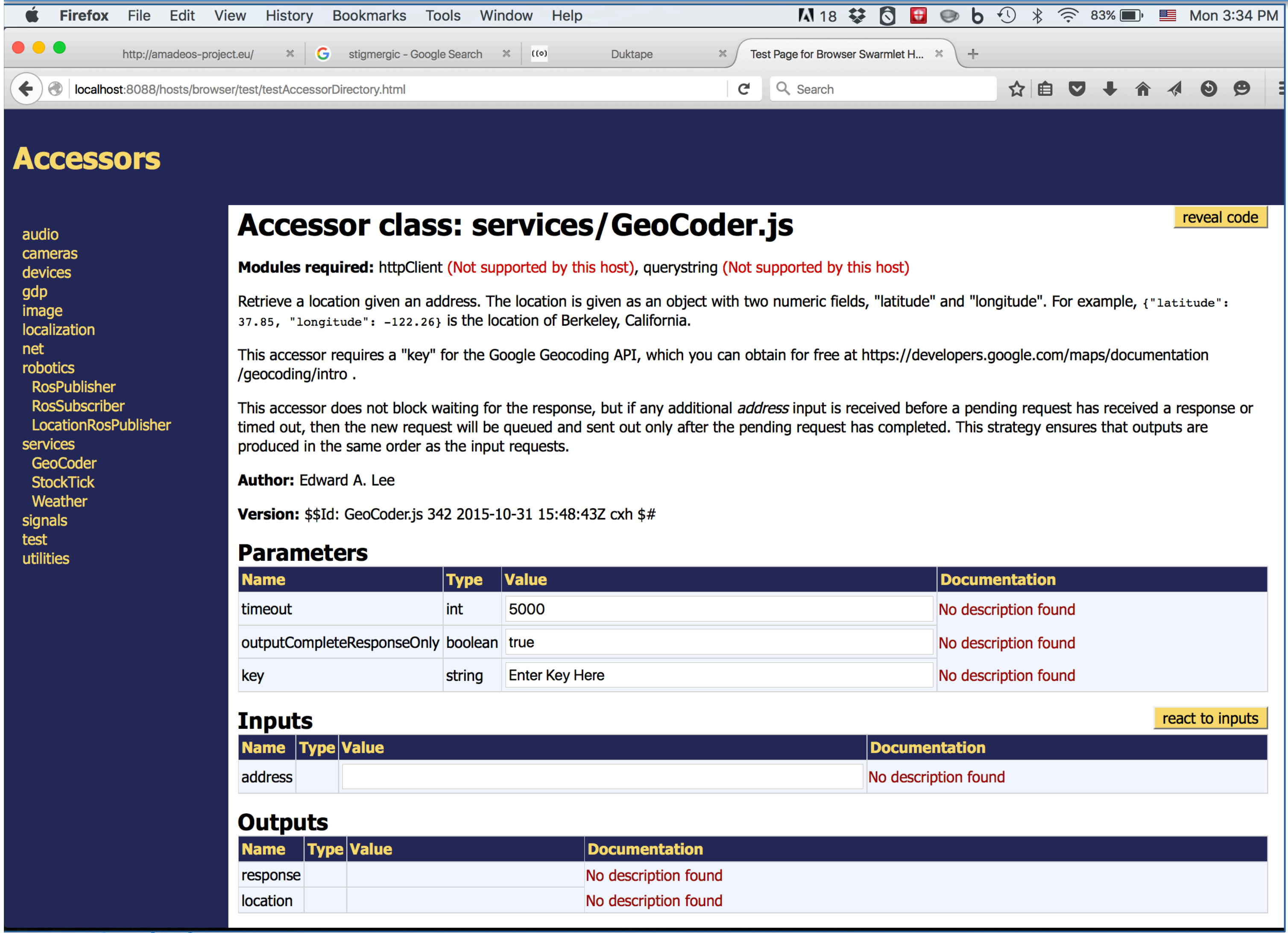
Recent Developments

- Common JavaScript core for three hosts:
  - Browser, Node.js, Cape Code
- Composite accessors (w/ a DE-like MoC)

```
exports.setup = function() {
    input('input', {'type':'number', 'value':0});
    output('output', {'type':'number'});
    var gain = instantiate('TestGain', 'test/TestGain');
    gain.setParameter('gain', 4);
    var adder = instantiate('TestAdder', 'test/TestAdder');
    connect('input', adder, 'inputLeft');
    connect('input', gain, 'input');
    connect(gain, 'scaled', adder, 'inputRight');
    connect(adder, 'sum', 'output');
}
```



Browser Host


Firefox File Edit View History Bookmarks Tools Window Help
Mon 3:34 PM
http://amadeos-project.eu/
stigmergic - Google Search
Duktape
Test Page for Browser Swarmlet H...
localhost:8088/hosts/browser/test/testAccessorDirectory.html
Search

Accessors

- audio
- cameras
- devices
- gdp
- image
- localization
- net
- robotics
  - RosPublisher
  - RosSubscriber
  - LocationRosPublisher
- services
  - GeoCoder
  - StockTick
  - Weather
- signals
- test
- utilities

Accessor class: services/GeoCoder.js reveal code

Modules required: httpClient (Not supported by this host), querystring (Not supported by this host)

Retrieve a location given an address. The location is given as an object with two numeric fields, "latitude" and "longitude". For example, `{"latitude": 37.85, "longitude": -122.26}` is the location of Berkeley, California.

This accessor requires a "key" for the Google Geocoding API, which you can obtain for free at https://developers.google.com/maps/documentation /geocoding/intro .

This accessor does not block waiting for the response, but if any additional *address* input is received before a pending request has received a response or timed out, then the new request will be queued and sent out only after the pending request has completed. This strategy ensures that outputs are produced in the same order as the input requests.

Author: Edward A. Lee

Version: $$Id: GeoCoder.js 342 2015-10-31 15:48:43Z cxh $#

Parameters

Name	Type	Value	Documentation
timeout	int	5000	No description found
outputCompleteResponseOnly	boolean	true	No description found
key	string	Enter Key Here	No description found

Inputs react to inputs

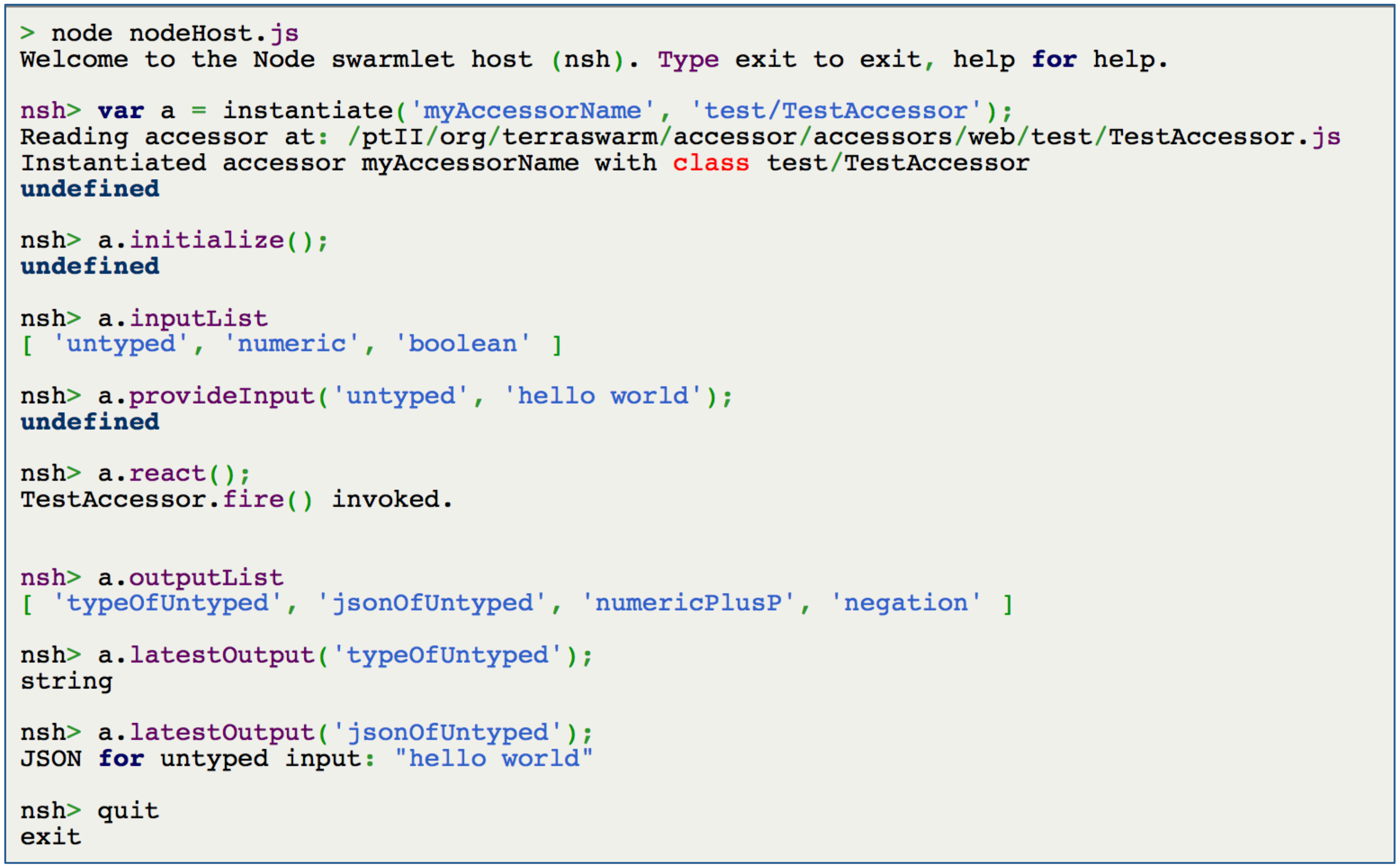
Name	Type	Value	Documentation
address			No description found

Outputs

Name	Type	Value	Documentation
response			No description found
location			No description found

Key challenge: Many accessors require modules that cannot be supported in a browser due to security constraints.

Node.js Host



```
> node nodeHost.js
Welcome to the Node swarmlet host (nsh). Type exit to exit, help for help.

nsh> var a = instantiate('myAccessorName', 'test/TestAccessor');
Reading accessor at: /ptII/org/terraswarm/accessor/accessors/web/test/TestAccessor.js
Instantiated accessor myAccessorName with class test/TestAccessor
undefined

nsh> a.initialize();
undefined

nsh> a.inputList
[ 'untyped', 'numeric', 'boolean' ]

nsh> a.provideInput('untyped', 'hello world');
undefined

nsh> a.react();
TestAccessor.fire() invoked.


nsh> a.outputList
[ 'typeOfUntyped', 'jsonOfUntyped', 'numericPlusP', 'negation' ]

nsh> a.latestOutput('typeOfUntyped');
string

nsh> a.latestOutput('jsonOfUntyped');
JSON for untyped input: "hello world"

nsh> quit
exit
```

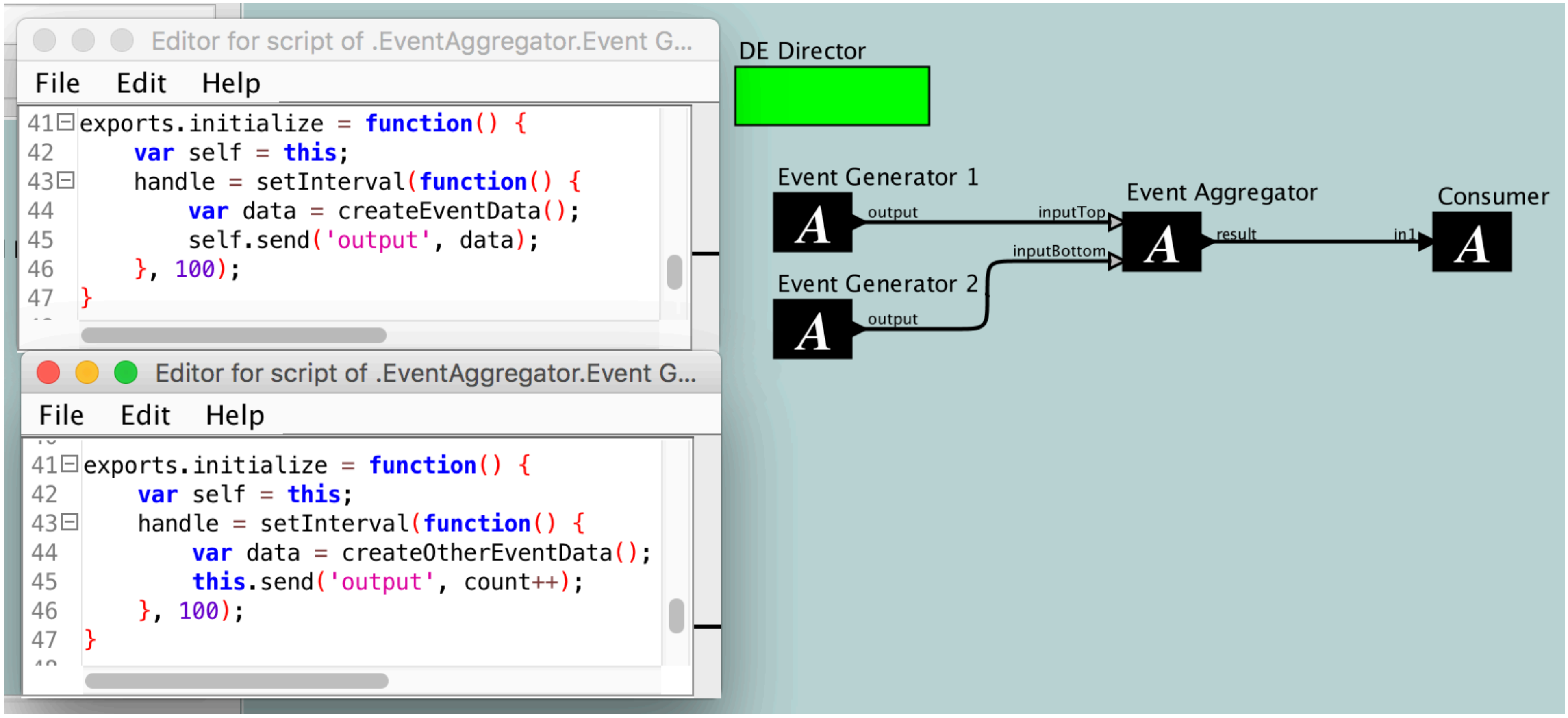
Key challenges:

Deterministic timed orchestration and coordination.

Maintain compatibility between modules supported by this host and the others.

Coordinated Timing

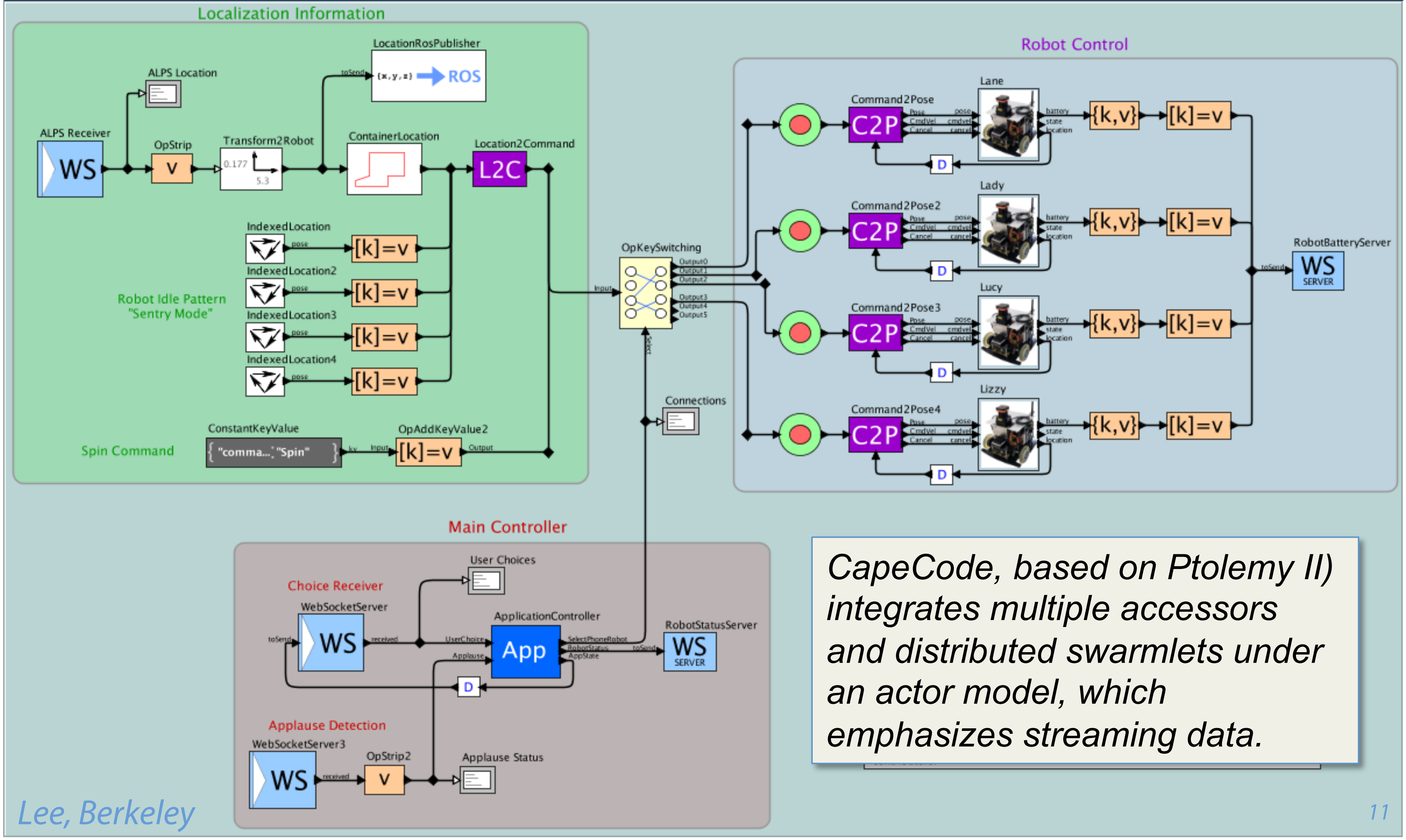
If we use the timed action support in browsers and in Node.js, we will not get deterministic interaction.



```
exports.initialize = function() {
    var self = this;
    handle = setInterval(function() {
        var data = createEventData();
        self.send('output', data);
    }, 100);
}
```

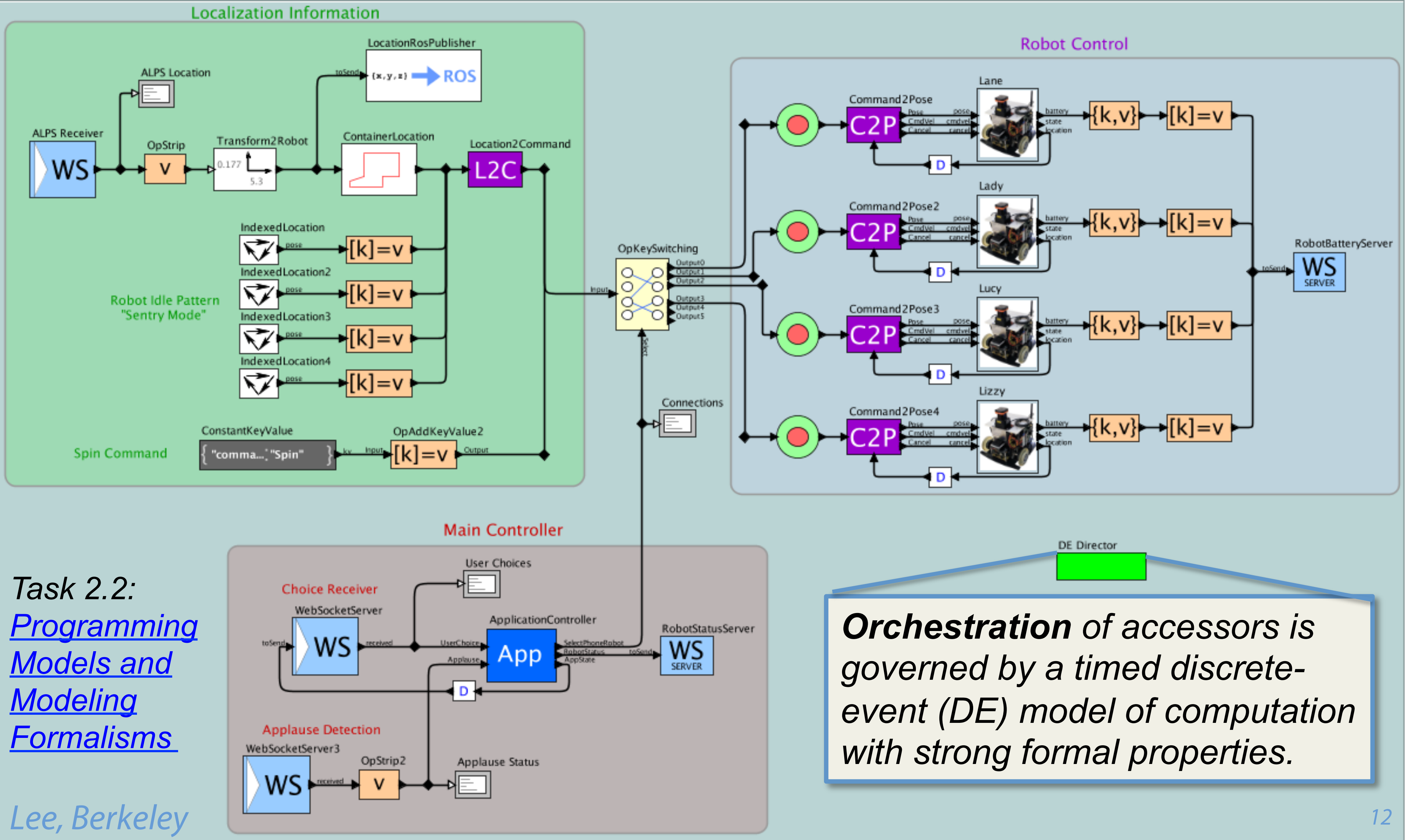
```
exports.initialize = function() {
    var self = this;
    handle = setInterval(function() {
        var data = createOtherEventData();
        this.send('output', count++);
    }, 100);
}
```

CapeCode Host



CapeCode, based on Ptolemy II) integrates multiple accessors and distributed swarmlets under an actor model, which emphasizes streaming data.

Key Element of the CapeCode Host: The DE Director

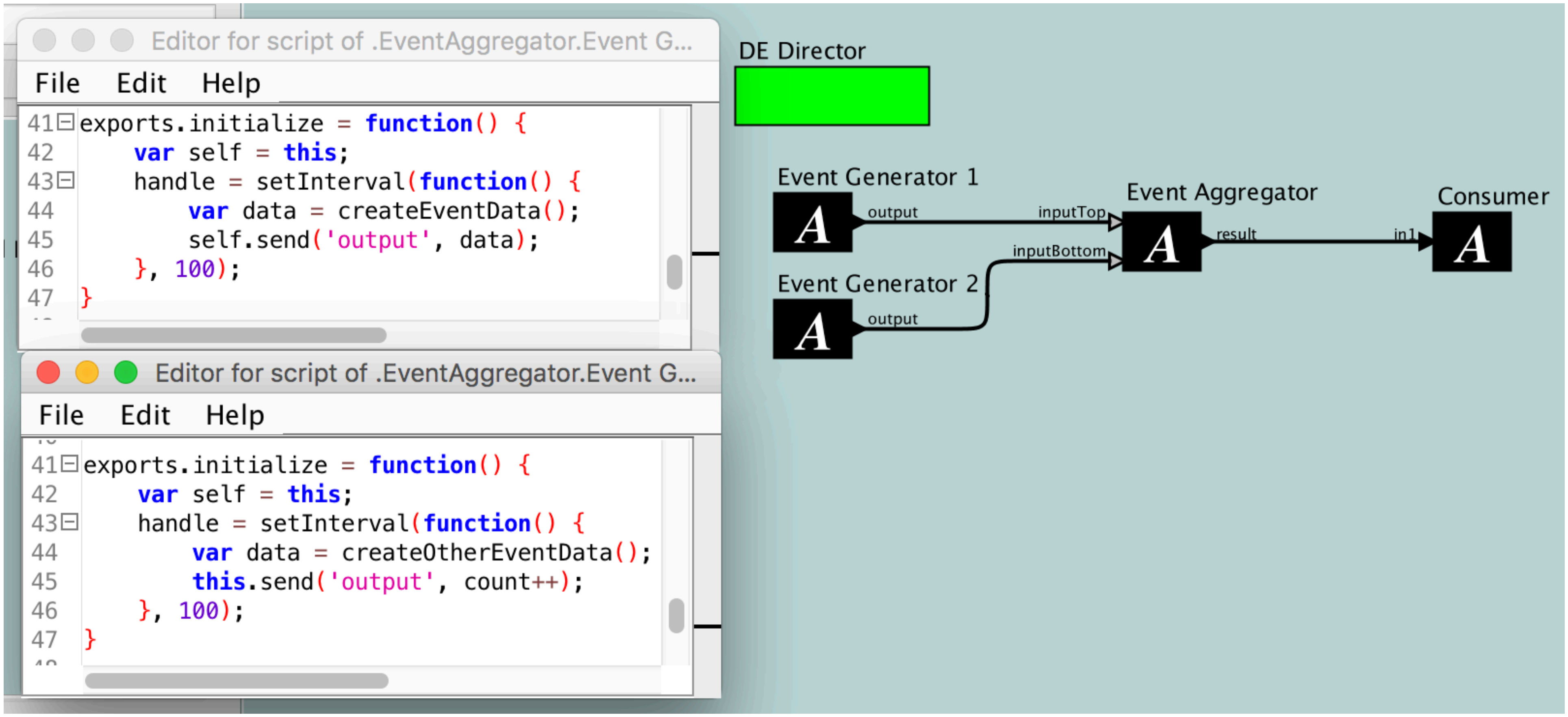


Task 2.2:
[Programming Models and Modeling Formalisms]

Orchestration *of accessors is governed by a timed discrete-event (DE) model of computation with strong formal properties.*

Coordinated Timing

In Cape Code, the Event Aggregator gets simultaneous events from the two generators.

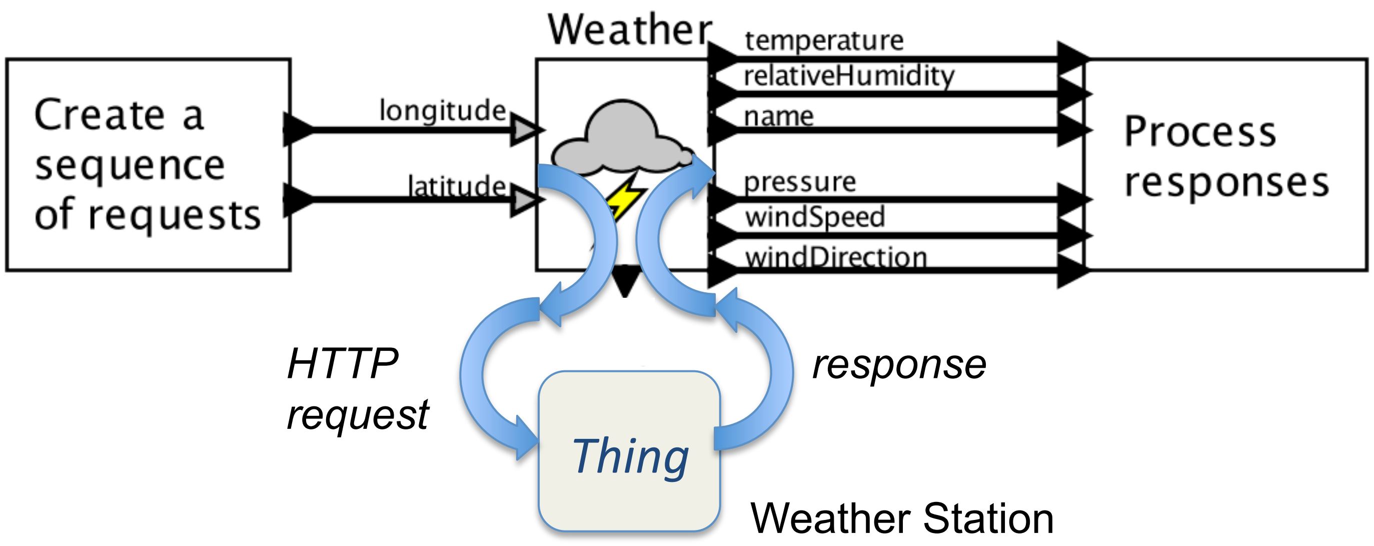


```
exports.initialize = function() {
    var self = this;
    handle = setInterval(function() {
        var data = createEventData();
        self.send('output', data);
    }, 100);
}
```

```
exports.initialize = function() {
    var self = this;
    handle = setInterval(function() {
        var data = createOtherEventData();
        this.send('output', count++);
    }, 100);
}
```

Another Timing Challenge with Actors and AAC

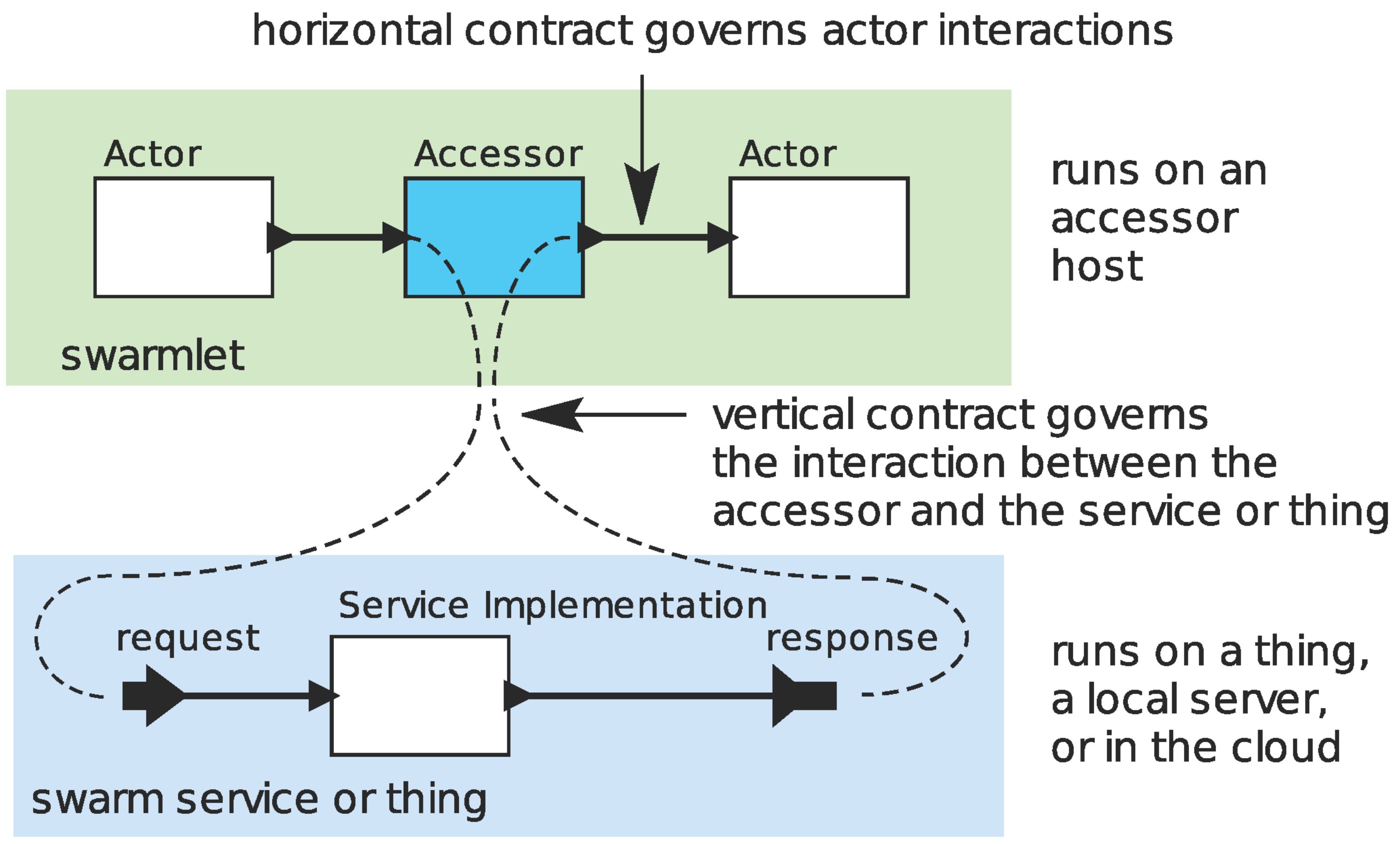
Example of a potential problem:



The responses may not come back in the same order as the requests!

CapeCode's realization of the httpClient module reorders responses to match the order of the requests.

Focus on Interfaces

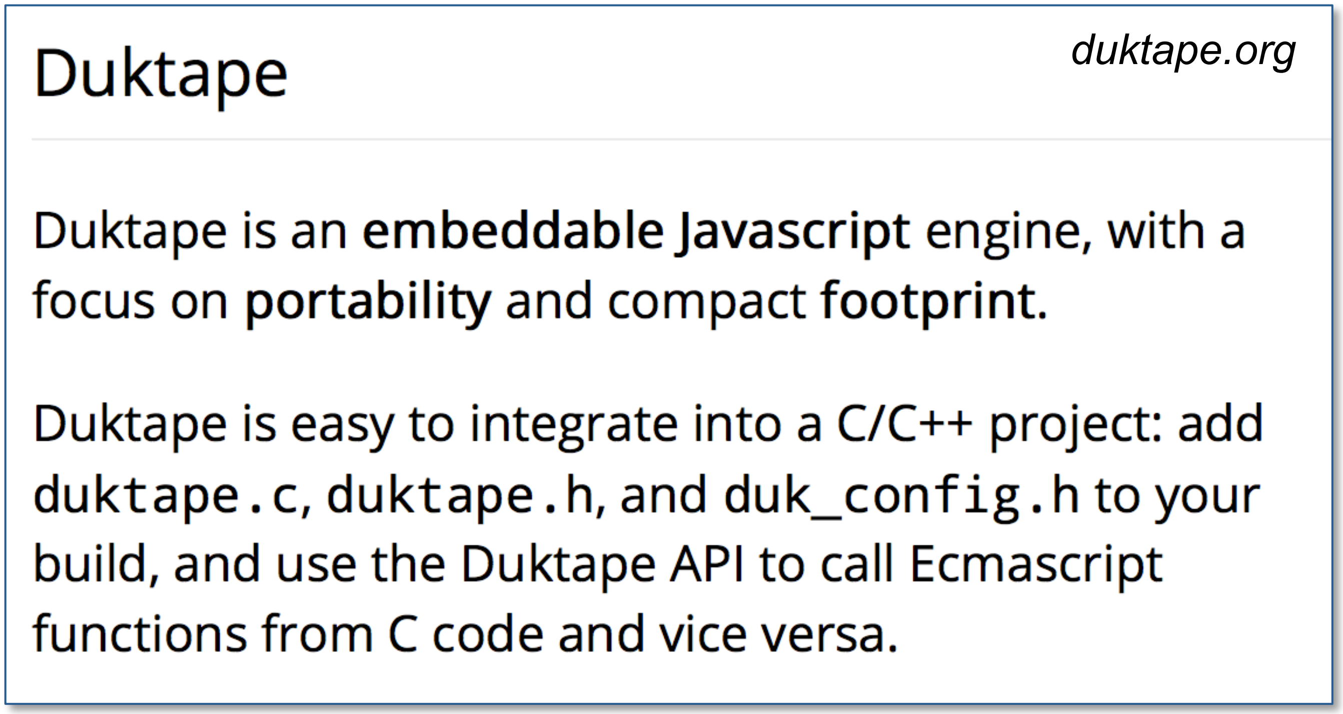

horizontal contract governs actor interactions
Actor
Accessor
Actor
swarmlet
runs on an accessor host
vertical contract governs the interaction between the accessor and the service or thing
Service Implementation
request
response
swarm service or thing
runs on a thing, a local server, or in the cloud

Sound timing semantics is part of the horizontal contract.

Reordering responses to match requests is part of the vertical contract.

An Opportunity

Several lightweight, embeddable JavaScript engines have appeared. A particularly attractive one is Duktape (from Samsung), which integrates nicely with embedded C code.



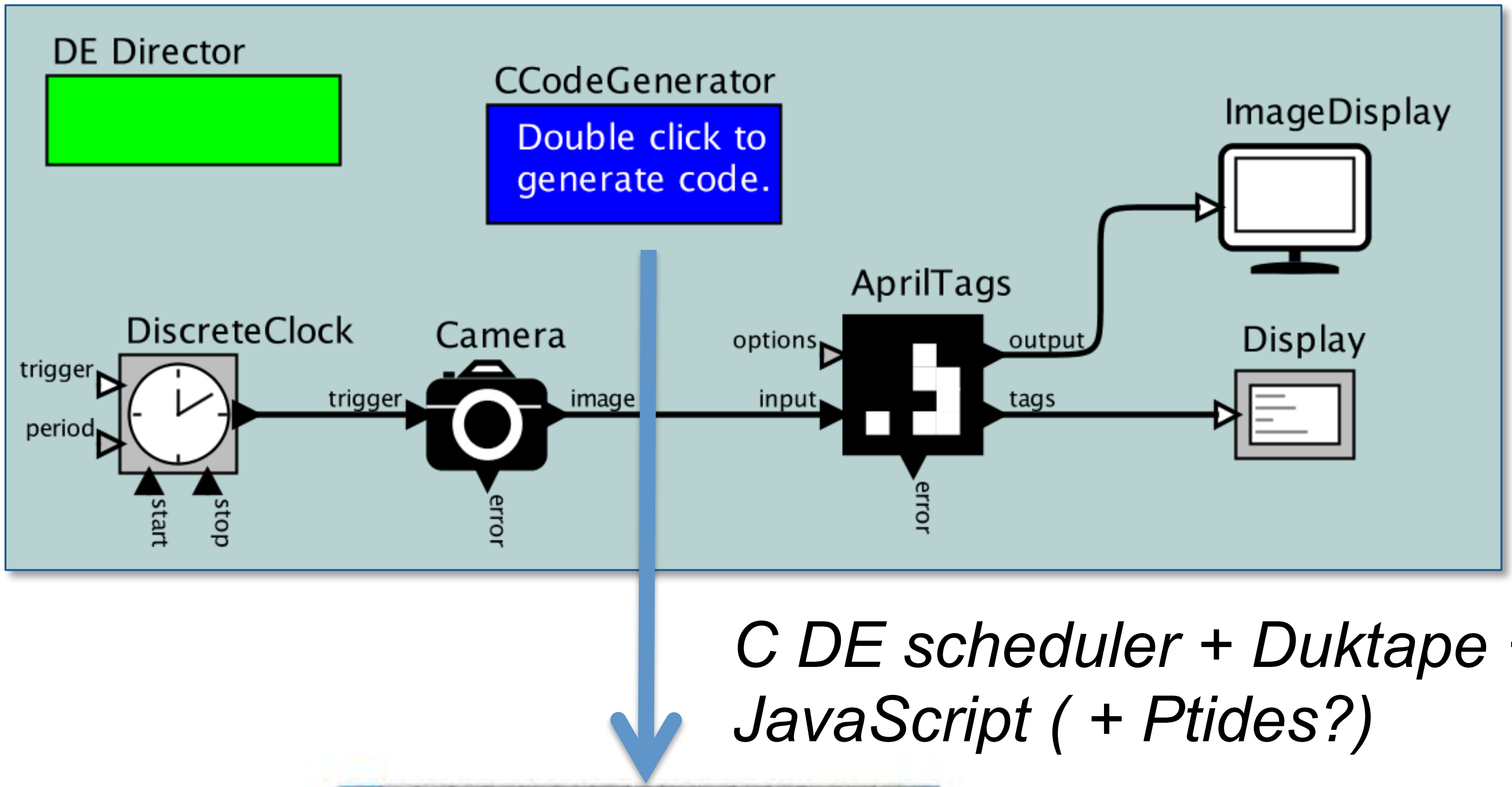
Duktape

duktape.org

Duktape is an **embeddable Javascript** engine, with a focus on **portability** and compact **footprint**.

Duktape is easy to integrate into a C/C++ project: add `duktape.c`, `duktape.h`, and `duk_config.h` to your build, and use the Duktape API to call Ecmascript functions from C code and vice versa.

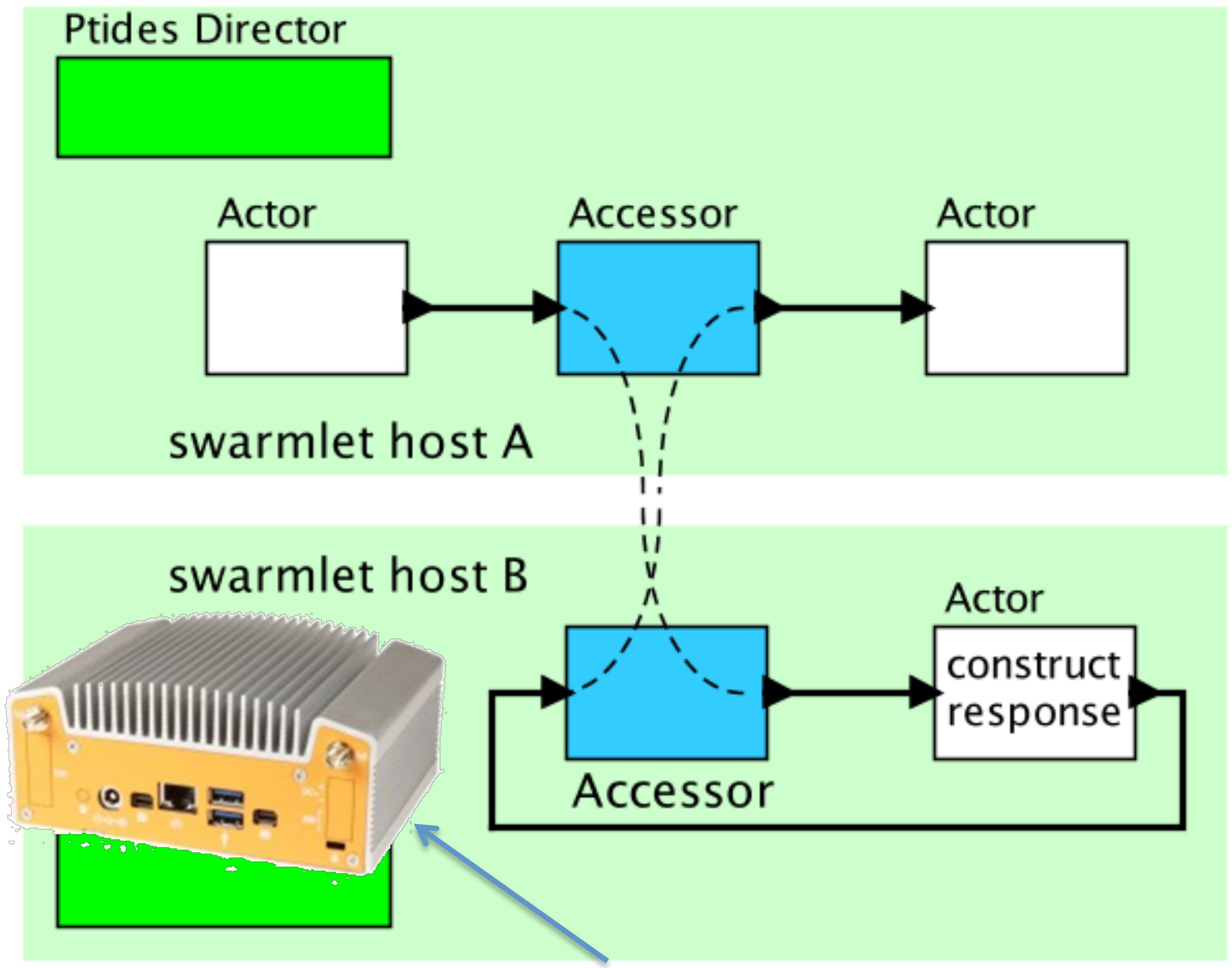
Code Generation?



C DE scheduler + Duktape + JavaScript (+ Ptides?)

Ptruly Ptiny Ptarget (PPP)

Another Opportunity: Distributed Swarmlets using Accessors



E.g., Intelligent gateway

Leveraging time stamps and synchronized clocks, we can achieve **deterministic** distributed MoCs.

See:

- PTIDES [2007]
- Google Spanner [2012]

The Biggest Problem

A lot of software needs to be written:

- Module implementations for hosts
- Duktape host needs to be designed and validated.
- Code generator needs to be developed (with point of departure being an existing, working, but limited C code generator for Ptolemy II).