Hop and HipHop: Multitier Web Orchestration

Gérard Berry*, Manuel Serrano**

*Collège de France
Chair Algorithms, Machines and Langages

**Inria Sophia-Antipolis

Indian Imagination : pbase.com/gberry

ICDCIT, Bhubaneswar, Feb. 6, 2014

COLLÈGE DE FRANCE
1530

informatiques mathématiques
Inria


2013
NBC NEWS
Michael Sohn / AP


2005
Luca Bruno / AP

A Darwinian Technology Space

transport display program navigator

- 1988-1992 (CERN): http, html, url
- 1994: cgi, netscape, javascript s., **cookie**, ssl / https
- 1995: ie, http1.0, html2, **javascript client**, php
- 1997: flash, gecko, apache, opera, css, http1.1, auth, dhtml
- 1999: css2, mathml, svg, html4, **soap**, httpu, **ajax**
- 2001: (xhtml), dom, wsdl
- 2002-08: webkit, canvas, html5, chrome, **json**, jquery
- 2010: mobile, **websocket**

The Primitive Web

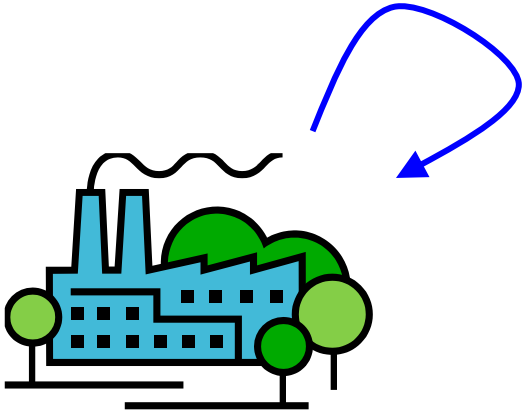
client

http

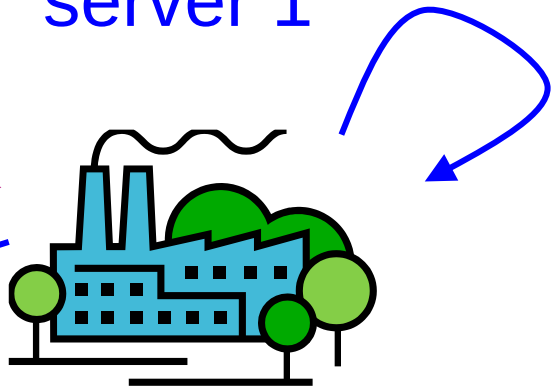
html

http

html



server 1



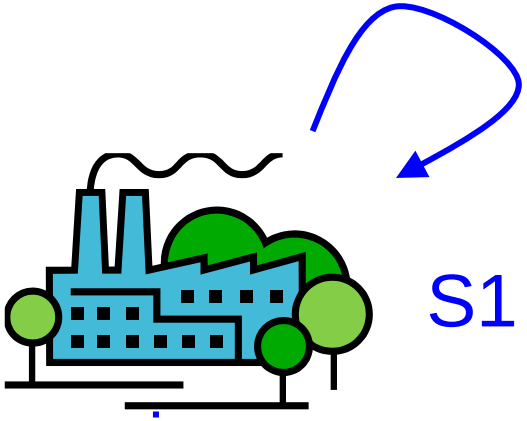
server 2

The program is the server

The Web 1.0: chatting with the server

client

http



S1

html

cookie

...

The program
is in the server

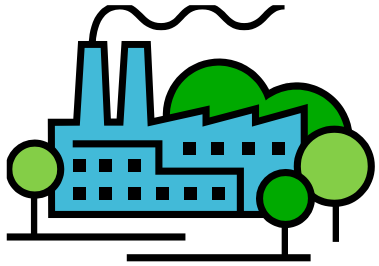
Main difficulty: make sense of the BACK button!

The Web 2.0: distributed programming

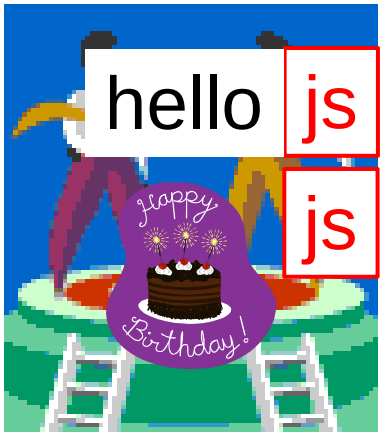
client

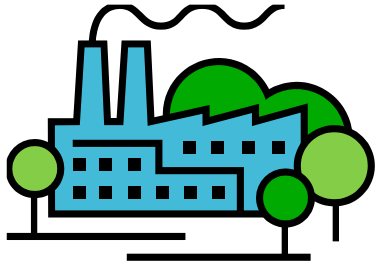
http

html



server 1

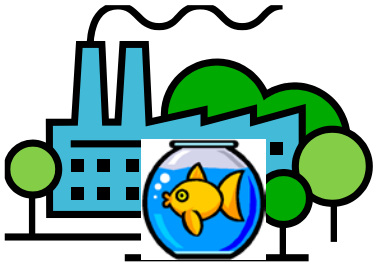




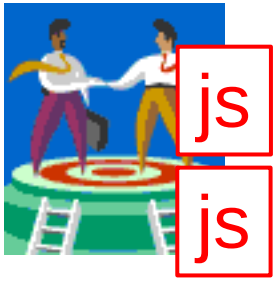
server 2

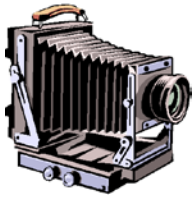
The program is distributed between the server(s) and client(s)

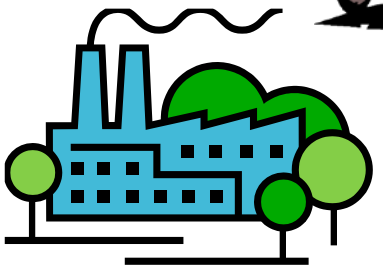
Future Web : Diffuse Programming



server 1







server 2

Distributed programs everywhere

Web Programming is Currently Too Difficult

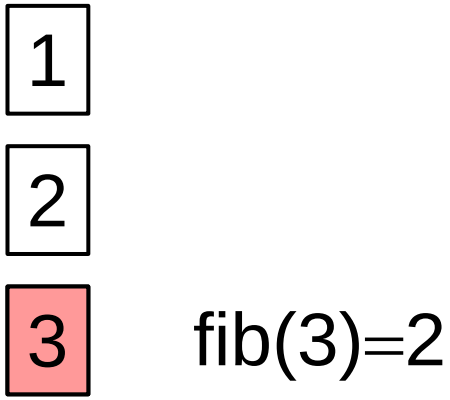
- Lots of technologies and languages to master
  ⇒ heterogeneous ad-hoc programming ☹
- Fundamental programming difficulties
  - distributed programming is hard
  - thread-based programming is error-prone
    (non-determinism, deadlocks, difficulty to debug)
  - idem for handling events by event-handlers
  - orchestrating asynchronous activities is problematic

Our solution:
A single programming framework: Hop
A Hop-embedded orchestration DSL: HipHop

Hop's Features

- Single language for all web programming needs
- Full algorithmic language: functional, Scheme-based
- Embeds HTML, makes it extensible
- Classical multithreading (indispensable)
- Multitier single code for client(s) / server(s)
- Server client code generation and shipping
  (client programs are server values)
- Automatic data communication
- Direct definition and use of web services

HTML → Hop Example



HTML / Javascript → Hop

```
<DOCTYPE HTML PUBLIC “-//W3C//DTD HTML 4.01//EN”
”http://www/w3/org/TR/htm14/strict.dtd”>
<HTML>
  <HEAD>
    <META http-equiv=”Contents-Type” content=”text/html”>
    <SCRIPT src=”fib.js” type=”application/javascript/”/>
  </HEAD>
  <BODY>
    <TABLE>
      <TR> <TD onclick=”alert(’fib(1)=’+fib(1))”>1</TD> </TR>
      <TR> <TD onclick=”alert(’fib(2)=’+fib(2))”>2</TD> </TR>
      <TR> <TD onclick=”alert(’fib(3)=’+fib(3))”>3</TD> </TR>
     </TABLE>
</BODY>
</HTML>
```

Hop Multi-Tier Programming

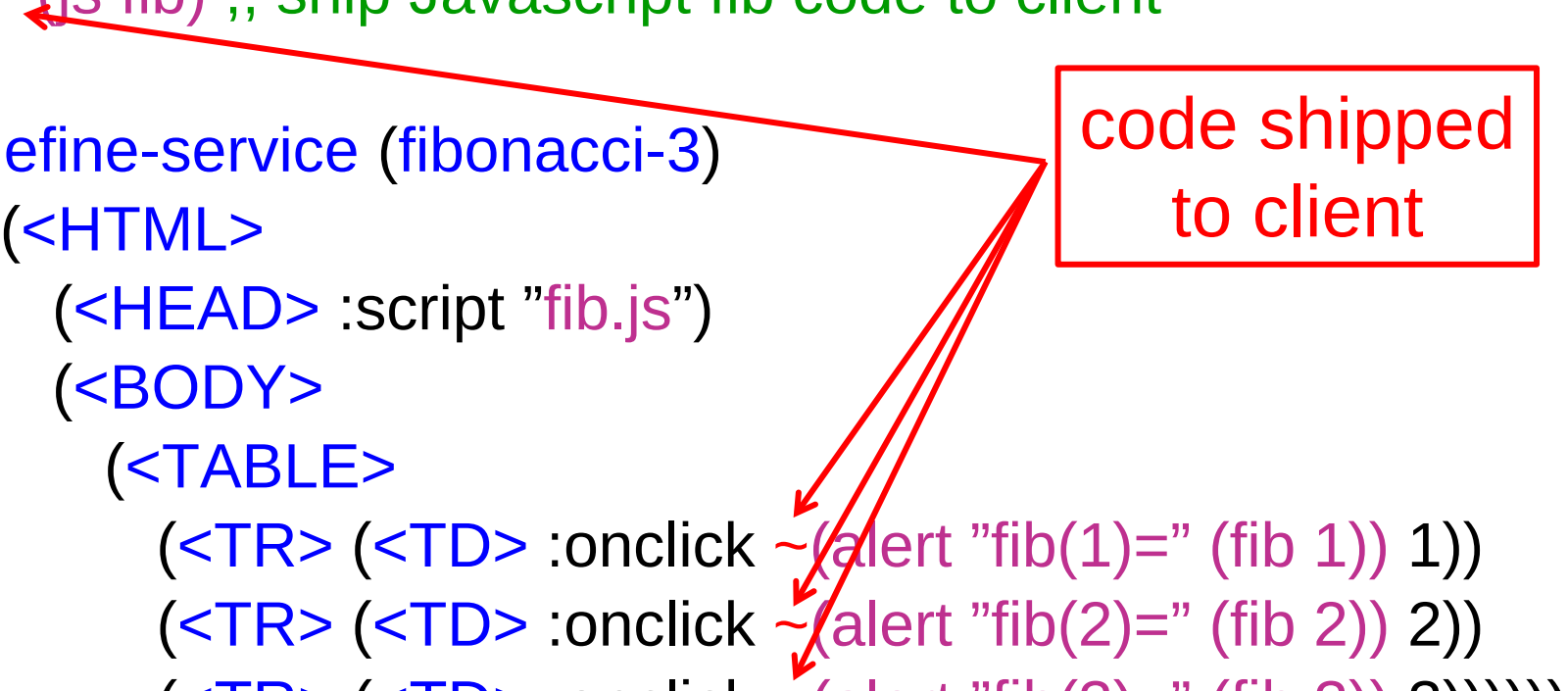
```
<DOCTYPE HTML PUBLIC “-//W3C//DTD HTML 4.01//EN”
“http://www/w3/org/TR/htm14/strict.dtd”>
(<HTML> ;; server-side function
  (<HEAD> :script ”fib.js”)
    <META http-equiv=“Contents-Type” content=“text/html”>
    <SCRIPT src=“fib.js” type=“application/javascript/”/>
  </HEAD>
  (<BODY>
    (<TABLE>
      (<TR> (<TD> :onclick ~(alert ”fib(1)=” (fib 1)) 1))
      (<TR> (<TD> :onclick ~(alert ”fib(2)=” (fib 2)) 2))
      (<TR> (<TD> :onclick ~(alert ”fib(3)=” (fib 3)) 3)))))
> </TR>
```

client-side mark

Hop Structured Programming (1)

```
(module Fibo
   ~(js fib) ;; ship Javascript fib code to client

(define-service (fibonacci-3)
   (<HTML>
      (<HEAD> :script "fib.js")
      (<BODY>
         (<TABLE>
            (<TR> (<TD> :onclick ~(alert "fib(1)=" (fib 1)) 1))
            (<TR> (<TD> :onclick ~(alert "fib(2)=" (fib 2)) 2))
            (<TR> (<TD> :onclick ~(alert "fib(3)=" (fib 3)) 3))))))
```



http://myserver:8080/hop/fibonacci-3

Hop Structured Programming (2)

```
(module Fibo
   ~(js fib) ;; ship Javascript fib code to client

(define-service (fibonacci-3)
   (<HTML>
      (<HEAD> :script ”fib.js”)
      (<BODY>
         (<TABLE>
            (<TR> (<TD> :onclick ~(alert ”fib(1)=” (fib 1)) 1))
            (<TR> (<TD> :onclick ~(alert ”fib(2)=” (fib 2)) 2))
            (<TR> (<TD> :onclick ~(alert ”fib(3)=” (fib 3)) 3))))))

(define-tag <TR-FIB> (i)
         (<TR> (<TD> :onclick ~(alert ”fib” $i “=” (fib $i)) i)))
```

server value exported to client

Hop Structured Programming (3)

```
(module Fibo
   ~(js fib) ;; ship Javascript fib code to client

(define-service (fibonacci-3)
   (<HTML>
      (<HEAD> :script ”fib.js”)
      (<BODY>
         (<TABLE>
            (<TR-FIB> 1))
            (<TR-FIB> 2))
            (<TR-FIB> 3)))))

(define-tag <TR-FIB> (i)
         (<TR> (<TD> :onclick ~(alert ”fib” $i “=” (fib $i)) i)))
```

Hop Functional Programming (1)

```
(module Fibo
   ~(js fib) ;; ship Javascript fib code to client

(define-service (fibonacci-3)
   (<HTML>
      (<HEAD> :script ”fib.js”)
      (<BODY>
         (<TABLE>
            (map <TR-FIB> ’(1 2 3)))))

(define-tag <TR-FIB> (i)
         (<TR> (<TD> :onclick ~(alert ”fib” $i “=” (fib $i)) i)))
```

Hop Functional Programming (2)

```
(module Fibo
   ~(js fib) ;; ship Javascript fib code to client

(define-service (fibonacci-3)
   (<HTML>
      (<HEAD> :script “fib.js”)
      (<BODY>
         (<TABLE>
            (map <TR-FIB> (iota 3 1))))))

(define-tag <TR-FIB> (i)
         (<TR> (<TD> :onclick ~(alert ”fib” $i “=” (fib $i)) i)))
```

Hop Web Service Definition

```
(module Fibo
  ~(js fib) ;; ship Javascript fib code to client

(define-service (fibonacci-list n) ;; arbitrary input list
  (<HTML>
    (<HEAD> :script “fib.js”)
    (<BODY>
      (<TABLE>
        (map <TR-FIB> (iota (string->number n) 1)))))

(define-tag <TR-FIB> (i)
        (<TR> (<TD> :onclick ~(alert ”fib” $i “=” (fib $i)) i)))
```

http://myserver:8080/hop/fibonacci-list?n=10

Exchanging data using services

1	1
2	1
3	2
4	3
5	5
6	8

```
;; build table of fib values as a list
(define-service (fibonacci-list2 n)
   (map (lambda (i)
          (cons i (fib i)))
        (iota n)))

;; client code
~(with-hop ($fibonacci-list2 8)
   (lambda (lst)
     (<TABLE>
       (map (lambda (i)
                 (<TR> (<TD> (car i)) (<TD> (cdr i))))
            lst))))
```

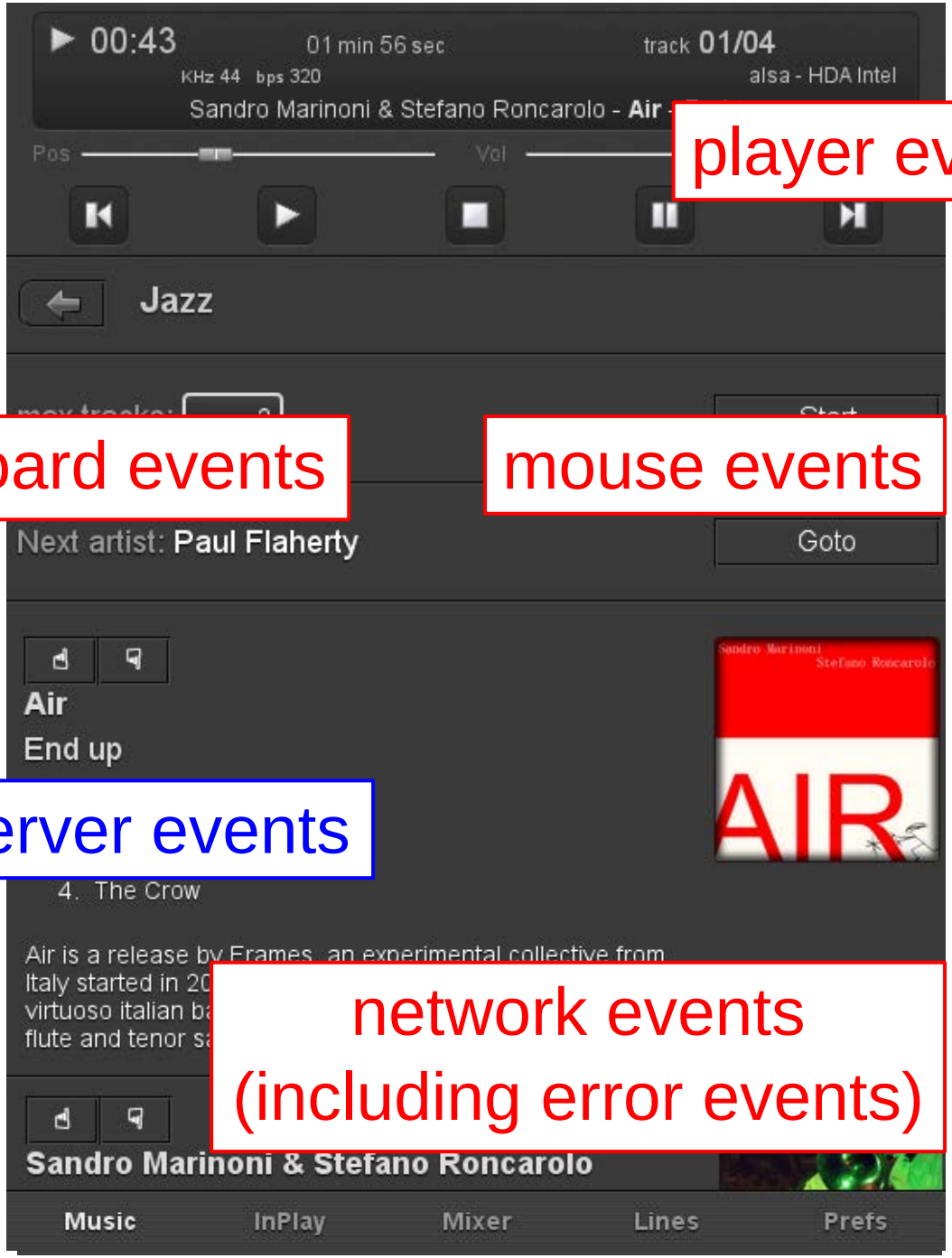
builds server url with argument

builds HTML on the client

The Web Orchestration Problem

- Building complex applications by combining existing web services
- Making them portable to any device
  - computer, smartphone, tablet, car, coffemaker, etc.

1. Handling lots of events of various kinds
   the main goal of Hiphop
2. Handling data streams
   currenly not done by HipHop
   the main goal of Orc / FRP / Flapjax



player events

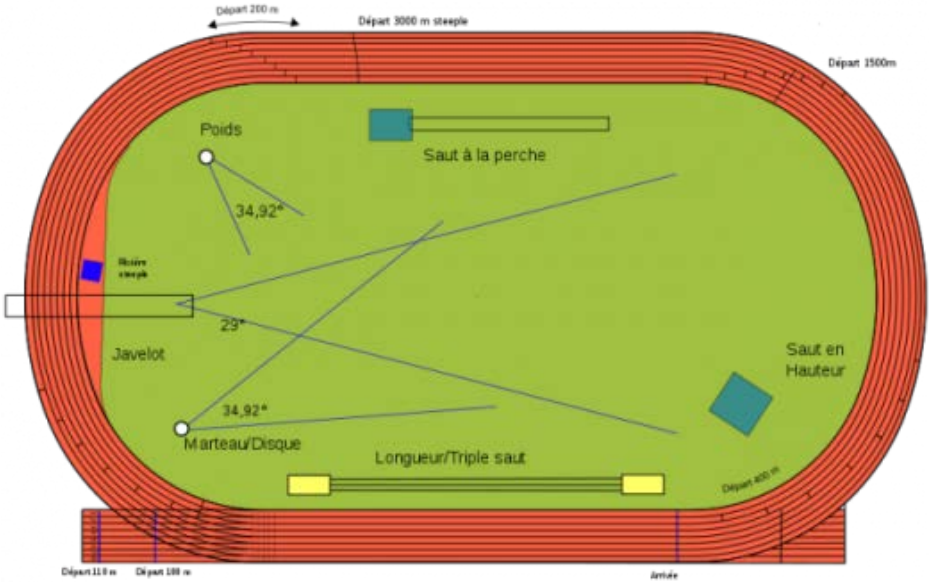
keyboard events

mouse events

hop server events

network events
(including error events)

Esterel : Reactive Event Handling



Second → Hour → Morning

Meter → Lap

Step

HeartBeat → HeartAttack

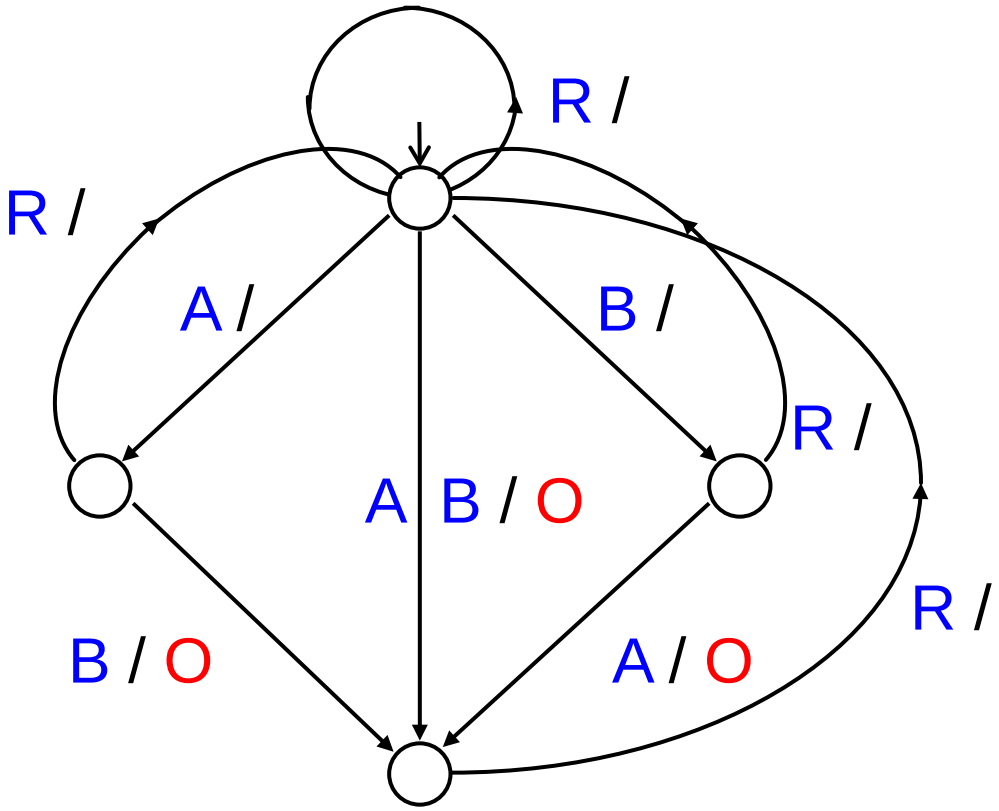
The Esterel Runner

```
trap HeartAttack in
    every Morning do
        abort
            loop
                abort  run Slowly  when 100 Meter ;
                abort
                    every Step do
                        run Jump || run Breathe || CheckHeart
                    end every
                when 15 Second ;
                run FullSpeed
            each  Lap
        when  4 Lap
    end every
handle HeartAttack do
   run RushToHospital
end trap
```

exit HeartAttack

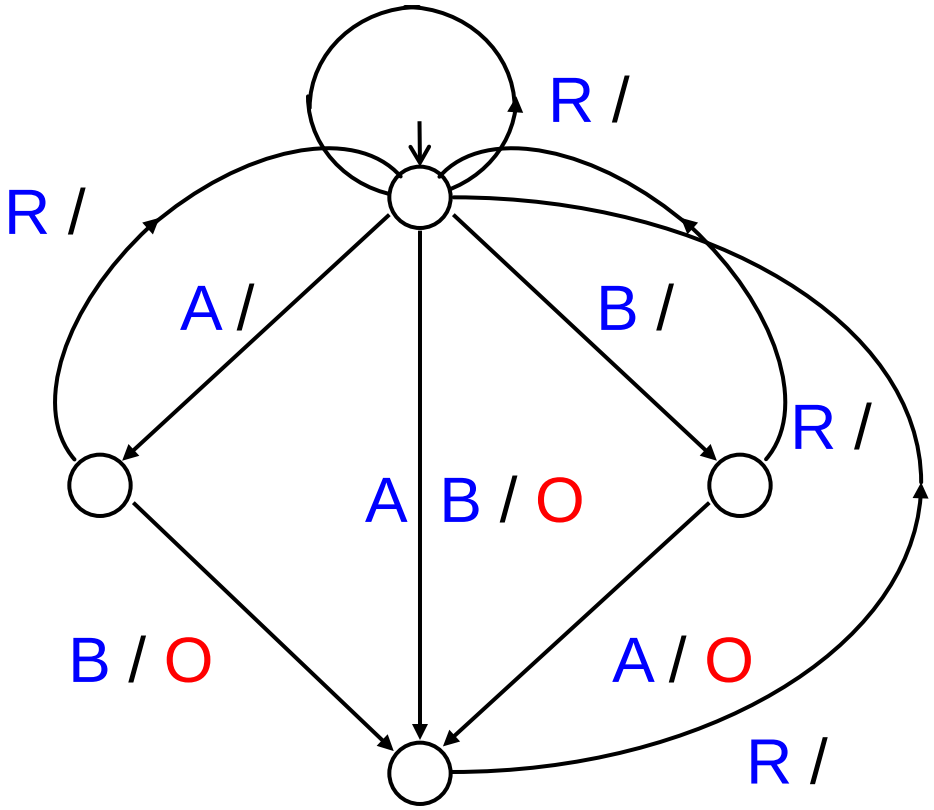
The ABRO Example

Emit O as soon as A and B have arrived
Reset behavior each time R is received



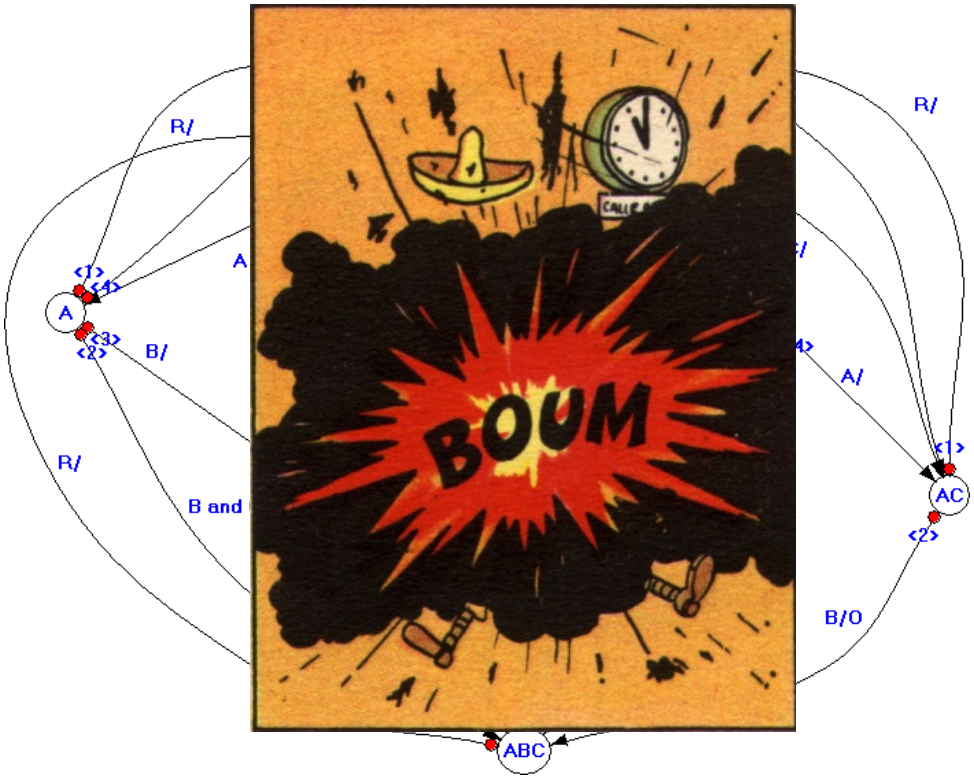
Get artists
R : new artist name
A : music
B : picture
O : play & display

Esterel = Linear Specification




```
loop
   abort
        { await A || await B };
        emit O;
        halt
   when R
end loop
```

Linear ⇒ No Source Code Explosion !



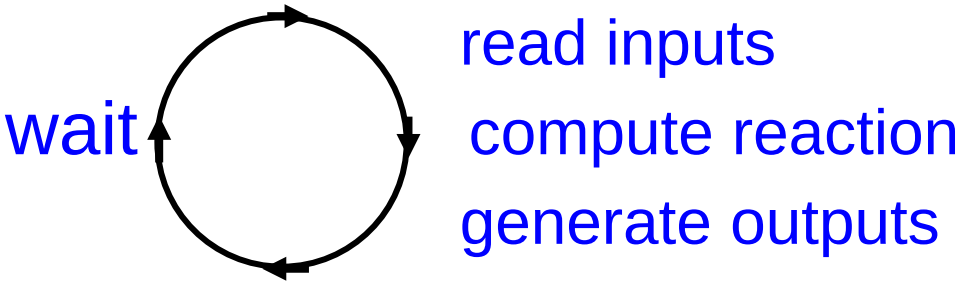
ABCRO

```
loop
   abort
        {    await A
          || await B
          || await C
        };
        emit O;
        halt
   when R
end loop
```

source: l'oreille cassée, Hergé

Cycle-Based Software Synchrony

Cyclic execution: static scheduling + 4-stroke engine



Synchronous = Zero-delay = within the same cycle

parallel propagation of control

parallel propagation of signals

Concurrency resolved at compile-time – no threads!

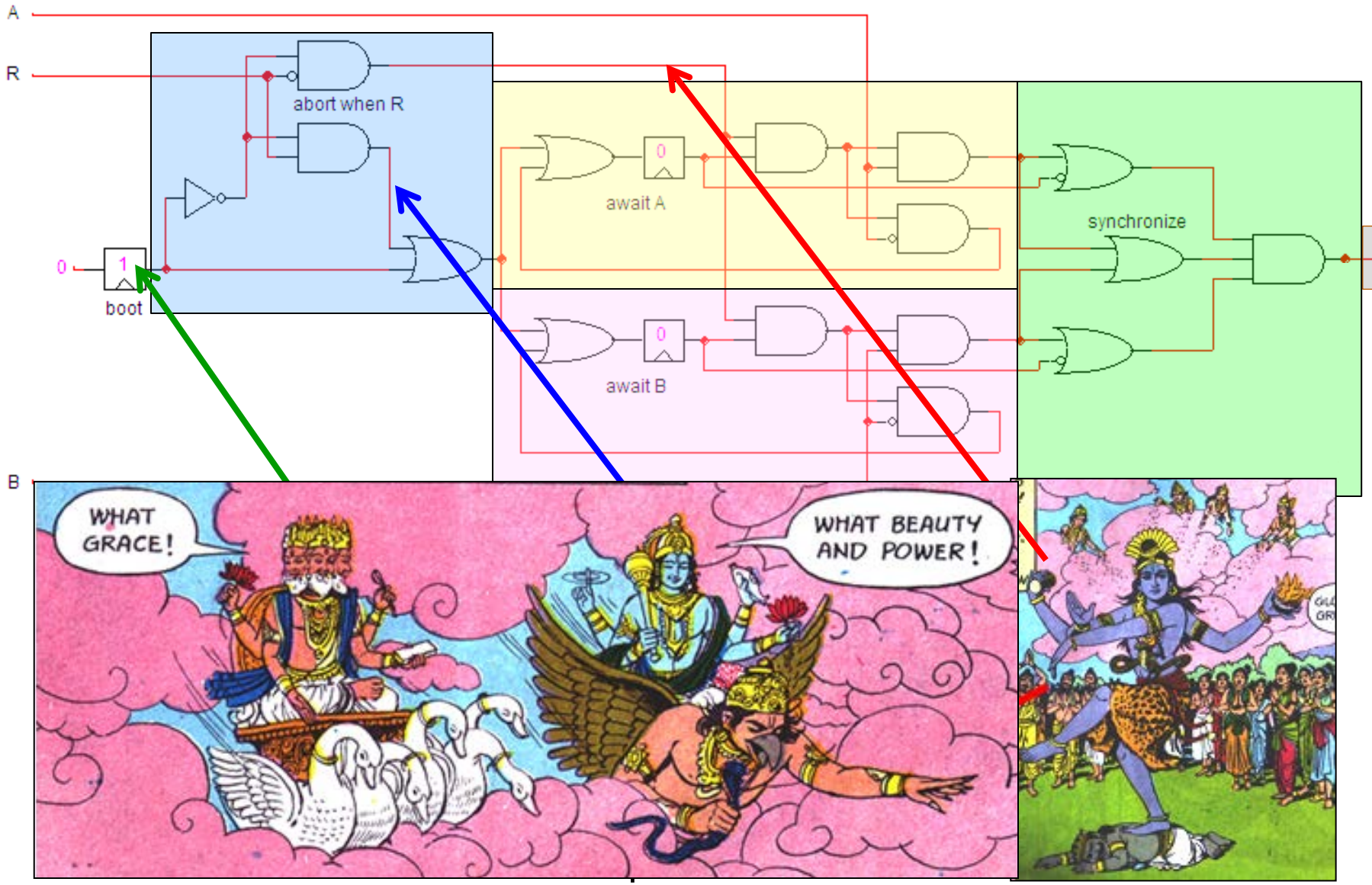
$\Rightarrow$ determinism, no event / computation interference

Synchronous Languages in Practice

- Avionics : Rafale, Airbus, Embraer, CARERI, etc,
- Trains, heavy industry, robotics
- Telecom
- Simulation of physical processes
- Circuit design, synthesis, and verification
- Music composition (!)

Both in industrial and academic contexts

ABRO en circuits



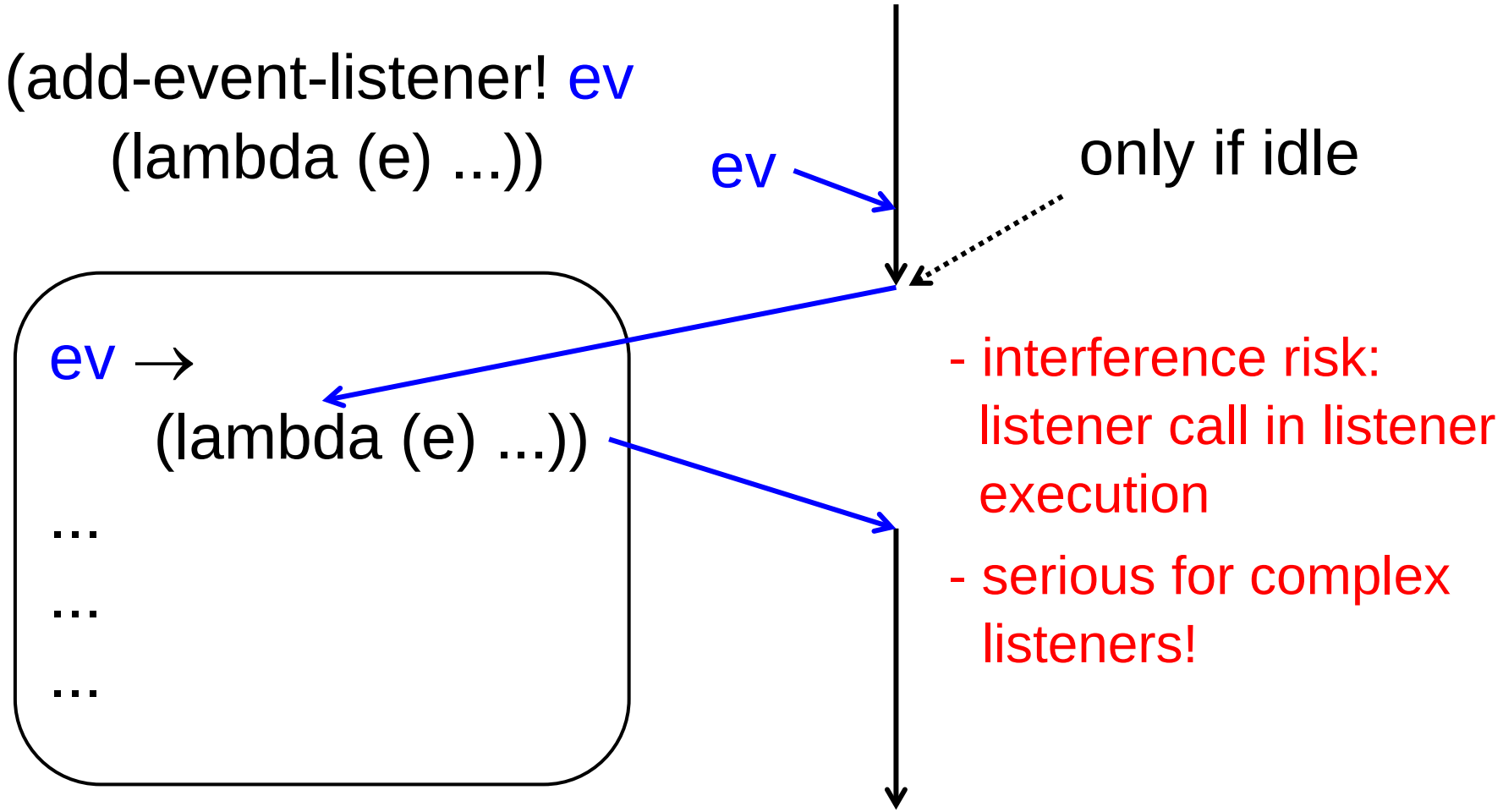
```
module Vishnu-3 :
loop
  trap T1 in
    signal S1 in
      pause; emit S1; exit T1
    ||
      loop
        trap T2 in
          signal S2 in
            pause; emit S2; exit T2
          ||
            loop
              present S1 and S2 then emit X end;
              present S1 and not S2 then emit Y end;
              present not S1 and not S2 then emit Z end;
              pause
            end loop
          end signal
        end trap
      end loop
    end signal
  end trap
end loop
end module
```

HipHop's Goal w.r.t. Orchestration

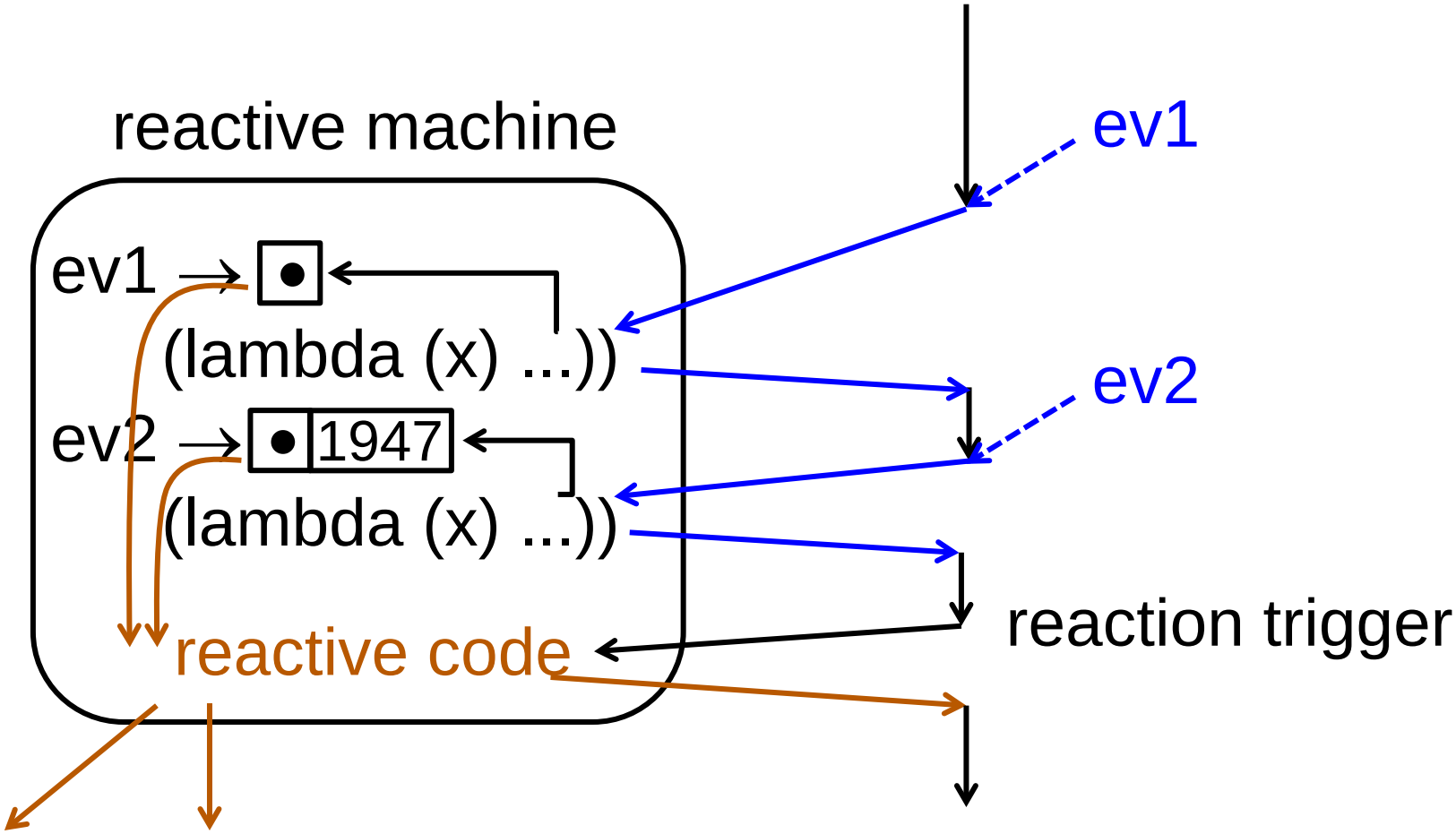
- Provide the user with a unified vision of event handling
  - GUI client events
  - events raised by services
  - events raised by web-connected objects
  - temporal events (e.g., timer expired)
  - Hop-generated events
- Define how to react to events in function of the memory of past events => orchestration
- Avoid all threads / event-listeners synchronization problems using Esterel's synchronous programming style

Exactly Esterel's goal, but more ambitious:
integration within Hop, dynamicity, multitier, etc.

Classical Javascript / HOP event Handling



The Synchronous Approach



sequence, parallelism, communication, preemption

synchronous, conceptually 0-delay

HipHop Events

```
(class HipHopEvent ...)  ;; similar to Esterel signals
                         ;; but first-class objects as in ReactiveML
```

- Boolean presence / absence status, unique at each instant

```
(now& e)      ;; Hop boolean expression
(pre& e)      ;; status at previous instant
```

- value of arbitrary type, unique at each instant

```
(val& e)      ;; Hop expression
(preval& e)   ;; value at previous instant
```

- combination functions for multiple synchronous emissions

```
(class CountEvent::HipHopEvent
    (status (default #f))
    (init (default 0))
    (op (default +))
  ... )
```

ABRO in HipHop

Emit O as soon as both A et B have arrived
Reset behavior each R

```
// Esterel module
module ABRO :
   input A, B, R;
   output O;
   loop
      { await A || await B };
      emit O;
   each R
end module
```

```
;; Hop function defining HipHop code
(define (ABRO& A B R O)
   (loop-each& (now& R)
      (par&
         (await& (now& A))
         (await& (now& B)))
      (emit& O)))
```

Variant of ABRO

After first R and every subsequent R,
emit O as soon as A and B have arrived.
Terminate if A et B occur simultaneously

```
(define (ABRObis& A B R O)
  (trap& Done
    (every& (now& R)
      (par&
        (await& (now& A))
        (await& (now& B)))
      (emit& O)
      (if& (and (now& A) (now& B))
        (exit& Done)))))
```

*HipHop Kernel: AST constructors**

* AST = Abstract Syntax Tree

stmt& :

(nothing&)

(emit& e hop*)

(atom& hop)

(pause&)

(if& hop *stmt& stmt&*)

(seq& *stmt&*$^+$)

(loop& *stmt&*$^+$)

(par& *stmt&*$^+$)

(suspend& hop *stmt&*$^+$)

(trap& trap-ident stmt&$^+$)

(exit& trap-ident)

(local& (local-e$^+$) stmt+)

HipHop Language Extensibility

Statements derived from primitive ones:

(halt&)

(sustain& e hop)

(await& [:immediate bool] delay stmt*)

(abort& [:immediate bool] delay stmt^{+})

(until& [:immediate bool] delay stmt^{+}) ;; Esterel weak abort

(loop-each& delay stmt^{+})

(every& [:immediate bool] delay stmt^{+})

```
(define (sustain& e . hop-list)
  (loop&
     (emit&  (cons e hop-list)
     (pause&)))
```

Build sustain&'s AST using Hop

HipHop Language Extensibility

```
(define (await-last-of& . e-list)
    (par& (map await& e-list)))
```

```
(await-last-of& A B C)
```

```
(par& (await& A) (await& B) (await& C))
```

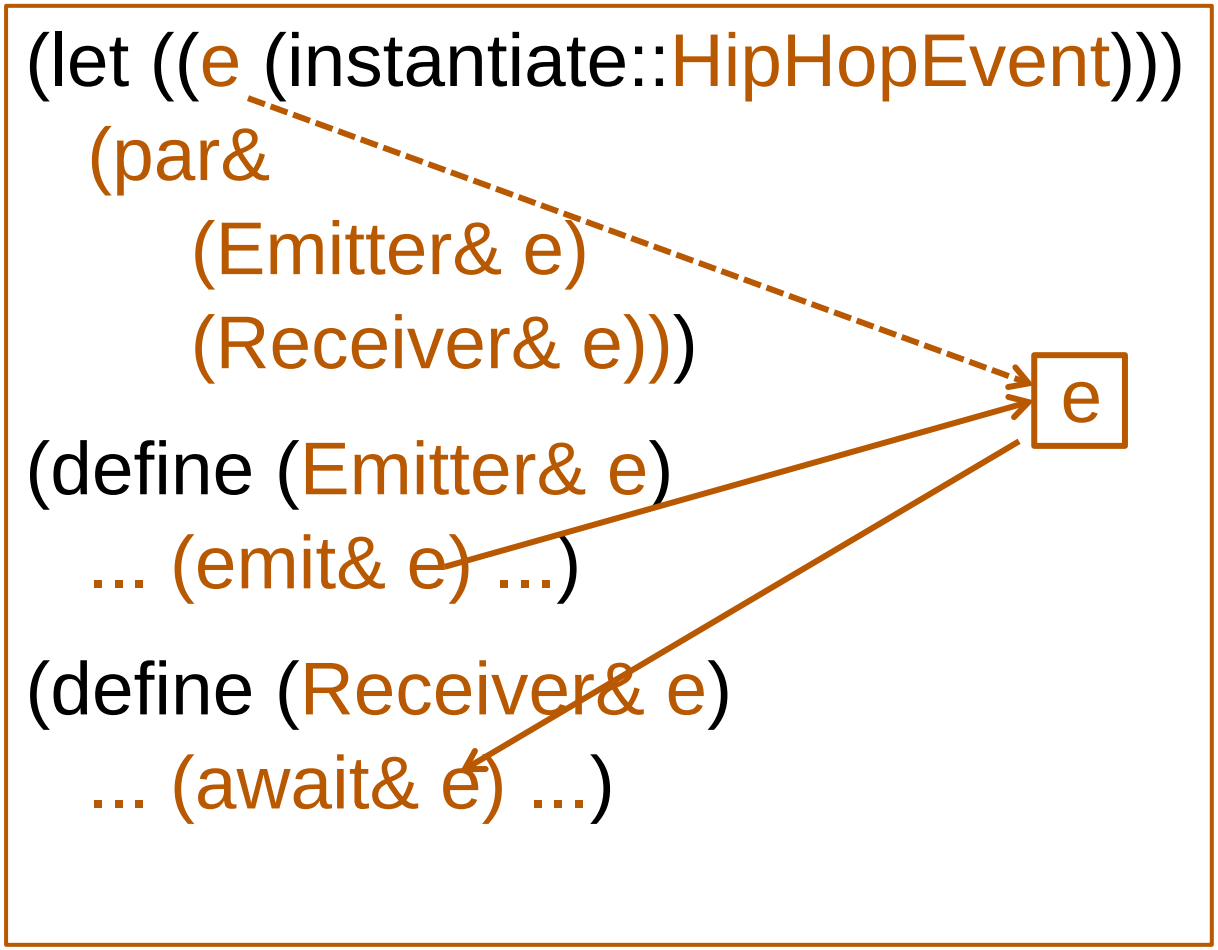
Extension : dynamic reactive behavior definition
genpar&, dyngenpar& : dynamically compute the par& instruction during the reaction

HipHop Language Extensibility

```
(define (repeat& N::int stmt)
   ;; declare a fresh private Hop counter
   (let ((count::int N))
      ;; install a trap to exit after N steps
      (trap& end
         ;; reset the local counter
         (atom& (set! count N))
         ;; loop until exit&
         (loop&
            ;; execute the user stmt
            stmt
            ;; increment the Hop counter
            (atom& (set! count (–fx count 1)))
            (if& (= count 0)
               ;; the end, escape from the loop
               (exit& end))))))
```

Note:
true function
not ugly macro!

Modularity inherited from Hop



Execution Machine

- Goal : handle input and output HipHop events
  triggers the HipHop reaction when asked to

(define M (instantiate::HipHopMachine (program P&)))

Execution Machine: Input and Reaction

HipHop input: Calling the hiphop-input! Hop function in the main Hop code or in an event handler


```
(hiphop-input! M A)
...
(hiphop-input! M B 1947)
...
(hiphop-react!) ;; atomic !
```

```
(hiphop-input-and-react! M A)

(hiphop-input-and-react! M B 1947)
```

Execution Machine: Gathering Inputs

- handling inputs beween two reactions

A = **STOP PLEASE**: one press is sufficient to set the bit

...

B = Toc : please count the Tocs!

Toc Toc Toc → (hiphop-input! M Toc 3)

Execution Machine : Output

- Define an HipHop event-listener for each output, called by M if the reaction emits the signal


```
(hiphop-add-event-listener! M X
                            (lambda () (action)))

(hiphop-add-event-listener! M Y
                            (lambda (V) (action V)))
```

hopfm& : Five Synchronous Activities

```
(define (hopfm&)
   (until& (memq (val& musicstate) '(stop ended))
      (par& ;; running all the components in synchronous parallel
         ;; peek a random playlist
         (random-playlist& catalog genre playlist)
         ;; manage a playlist
         (playlist& playlist)
         ;; deal with new tracks
         (track& track album artist)
         ;; manage new artists
         (artist-info& catalog genre artist bio discog similar playlist)
         ;; update the gui
         (gui& musicstate track album artist bio discog similar))))
```

Finding a Random Artist With Music

```
(define (random-playlist& catalog genre playlist)
   (trap& found
      ;; start looping
      (loop&
         ;; creates two local events
         (local& ((local-artist (instantiate::HipHopEvent))
                  (local-playlist (instantiate::HipHopEvent)))
            ;; get a random artist from FMA
            (with-hop& ($hopfm/genre/artist/random genre catalog) local-artist)
            ;; get the tracks of that artist
            (with-hop& ($hopfm/artist/tracks (val& local-artist)) local-playlist)
            (if& (pair? (val& local-playlist))
            ;; an artist with music is found
            (seq&
               (emit& playlist (val& local-playlist))
               (exit& found)))))))
```

From Specification To HipHop Code

The artist-info& component searches in parallel an image of the current artist, information about that artist, and a similar artist with a playlist. It outputs bio, discog, similar, and playlist towards the other components as soon as the corresponding information has been found.

```
(define (artist-info& catalog genre artist bio discog similar playlist)
   (every& (now& artist) ;; we have a playlist for that artist
      (par&
         ;; request a similar artist list
         (similar-artist& catalog genre playlist artist similar)
         ;; fetch artist biography and discography
         (artist/bio& artist bio discog)
         ;; fetch the artist images
         (artist/image& artist))))
```

Fetching an Image From Two Sites

```
(define (artist/image& artist)
;; find the first image out of two services
   (let ((el (dom-get-element-by-id "hophifi-internet-artist-image")))
      (local& ((img (instantiate::HipHopEvent (name "image"))))
         ;; try finding one image on two different servers
         ;; abort the pending request as soon as one returns
         (trap& (done)
            (par&
               (seq&
                  (with-hop& ($hopfm/artist/image (val& artist)) img)
                  (if& (now& img) (exit& done)))
               (seq&
                  (with-hop& ($hopfm/artist/image/echonest (val& artist)) img)
                  (if& (now& img) (exit& done)))))
            (if& (now& img)
               ;; update the GUI with the new image
               (atom&
                  (node-style-set! el :visibility "visible")
                  (set! el.src (val& img)))
               ;; no image was found, hides the current one
               (atom& (node-style-set! el :visibility "hidden"))))))
```

parallel search, immediately terminates by exit& as soon as one search returns an image

```
(trap& (done)
   (par&
      (seq&
         (with-hop& ($hopfm/artist/image (val& artist)) img)
         (if& (now& img) (exit& done)))
      (seq&
         (with-hop& ($hopfm/artist/image/echonest (val& artist)) img)
         (if& (now& img) (exit& done)))))
      (if& (now& img)
         <then>
         <else>)))
```

but parallel may also terminate if both searches return no image

HipHop fix for Chromium Bug

```
(define  (onended& play ended stop)
   (every& (now& play) ;; new music in the player
      (until& (or (now& stop) (now& ended))
         (loop& ;; slow loop polling music duration
            (let& ((music (val& play)))
               (if& (music-duration? music)
                  (await& (timer& (music-duration music))
                     (loop& ;; fast loop polling music-ended?
                        (if& (music-ended? music)
                           (emit& ended music)
                           (await& (timer& 10)))))
                  (await& (timer& 100))))))))
```

no threads, no event loop

Conclusion

- The web becomes the universal diffuse platform
- Hop greatly simplifies and integrates web programming
- HipHop greatly simplifies reaction to events and global service orchestration
- But stream orchestration *à la* ORC not yet available

More research and experimentation needed!

Related Reactive Work

- Esterel, Lustre, Signal
  - environment integration not part of the language
- Esterel Execution Machines (C. André, D. Gaffé)
- ReactiveC (F. Boussinot) ,Reactive ML (L. Mandel) integration within C / ML
  - successors : Junior (Java), SugarCubes, FunLoft, etc.
- Timed CCP, V. Saraswat :
  - integration within Concurrent Constraint Programming
  - much more implicit control
- Quartz, SCL in Germany