

JFLA ~~2005~~ 2014

# ReactiveML : un langage pour la programmation réactive en ML

Louis Mandel    Marc Pouzet

`{louis.mandel, marc.pouzet}@lip6.fr`

Laboratoire d'Informatique de Paris 6  
Université Pierre et Marie Curie

~~9 mars 2005~~

11 janvier 2014

# Historique

- 1998 : cours de programmation en Licence
  - programmation événementielle

# Programmation événementielle

```
class generate_new_plateforme = object(self)
  val mutable state = 0
  val mutable last_click = (0, 0)

  method on_click pos =
    match state with
    | 0 -> last_click <- pos;
           state <- 1
    | 1 -> emit new_plateforme (last_click, pos);
           state <- 0

  method on_key_down k =
    match k with
    | Key_ESC -> state <- 0
    | _ -> ()

end
```

# Historique

- 1998 : cours de programmation en Licence
  - programmation événementielle

# Historique

- 1998 : cours de programmation en Licence
  - programmation événementielle
- 2002 : Thèse sur la reconfiguration dynamique en Lucid Synchrone

# Historique

- 1998 : cours de programmation en Licence
  - programmation événementielle
- 2002 : DEA sur les SugarCubes [Boussinot et Susini]
- 2002 : Thèse sur la reconfiguration dynamique en Lucid Synchrone

# Historique

- 1998 : cours de programmation en Licence
  - programmation événementielle
- 2002 : DEA sur les SugarCubes [Boussinot et Susini]
- ~~● 2002 : Thèse sur la reconfiguration dynamique en Lucid Synchrone~~
- 2002 - 2003 : Implantations de SugarCubes (Junior)
  - Inspiration en particulier de Junior par L. Hazard

# Historique

- 1998 : cours de programmation en Licence
  - programmation événementielle
- 2002 : DEA sur les SugarCubes [Boussinot et Susini]
- ~~● 2002 : Thèse sur la reconfiguration dynamique en Lucid Synchrone~~
- 2002 - 2003 : Implantations de SugarCubes (Junior)
  - Inspiration en particulier de Junior par L. Hazard
- Fin 2003 : passage de Java à ML
  - lien entre la partie réactive et langage hôte
  - simplification de l'implantation

# Approche langages

- **fournir des constructions de haut niveau** pour composer/décrire des systèmes interactifs
- **alternative aux approches classiques** : impérative, programmation événementielle, concurrente (à base de thread), ...
- **la question de l'efficacité est centrale**, il n'y a pas de threads à l'exécution
- **mécanismes de sûreté** (e.g., typage)
- **parallélisme déterministe**
- **s'intégrer a un langage existant** (OCAML) sans réduire son pouvoir expressif

On fonde ce langage sur le modèle réactif introduit par F. Boussinot

# Retour sur l'exemple

```
let process generate_new_plateforme click key new_plateforme =  
  loop  
    await click (p1) in  
    do  
      await click (p2) in  
        emit new_plateforme (p1, p2)  
    until key(Key_ESC) done  
  end
```

# Approche langages

- **fournir des constructions de haut niveau** pour composer/décrire des systèmes interactifs
- **alternative aux approches classiques** : impérative, programmation événementielle, concurrente (à base de thread), ...
- **la question de l'efficacité est centrale**, il n'y a pas de threads à l'exécution
- **mécanismes de sûreté** (e.g., typage)
- **parallélisme déterministe**
- **s'intégrer a un langage existant** (OCAML) sans réduire son pouvoir expressif

On fonde ce langage sur le modèle réactif introduit par F. Boussinot

# Analyse de réactivité

## Boucle instantanée

```
let process instantaneous_loop =  
  loop () end
```

## Récursion instantanée

```
let rec process instantaneous_rec =  
  run instantaneous_rec
```

# Approche langages

- **fournir des constructions de haut niveau** pour composer/décrire des systèmes interactifs
- **alternative aux approches classiques** : impérative, programmation événementielle, concurrente (à base de thread), ...
- **la question de l'efficacité est centrale**, il n'y a pas de threads à l'exécution
- **mécanismes de sûreté** (e.g., typage)
- **parallélisme déterministe**
- **s'intégrer a un langage existant** (OCAML) sans réduire son pouvoir expressif

On fonde ce langage sur le modèle réactif introduit par F. Boussinot

# Programming Mixed Music in ReactiveML

- Collaboration entre
  - l'équipe Mutant de l'IRCAM
  - l'équipe Parkas de l'ENS

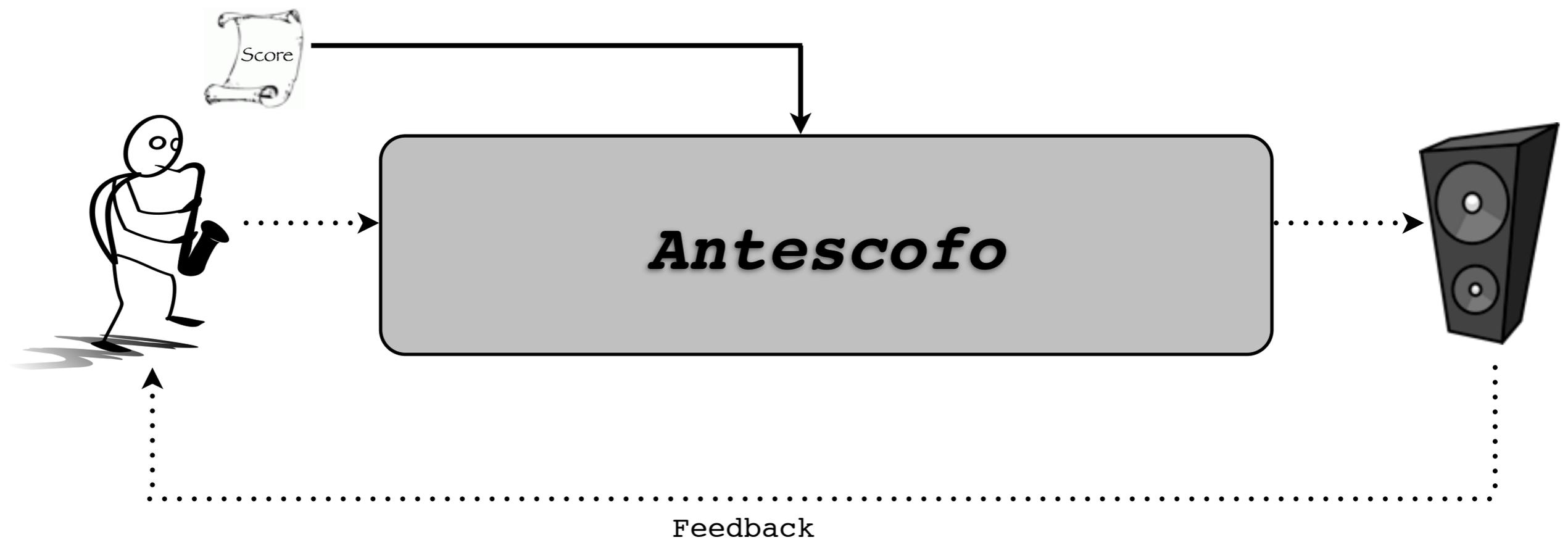
# Mixed Music and Antescofo

[Cont 2008]



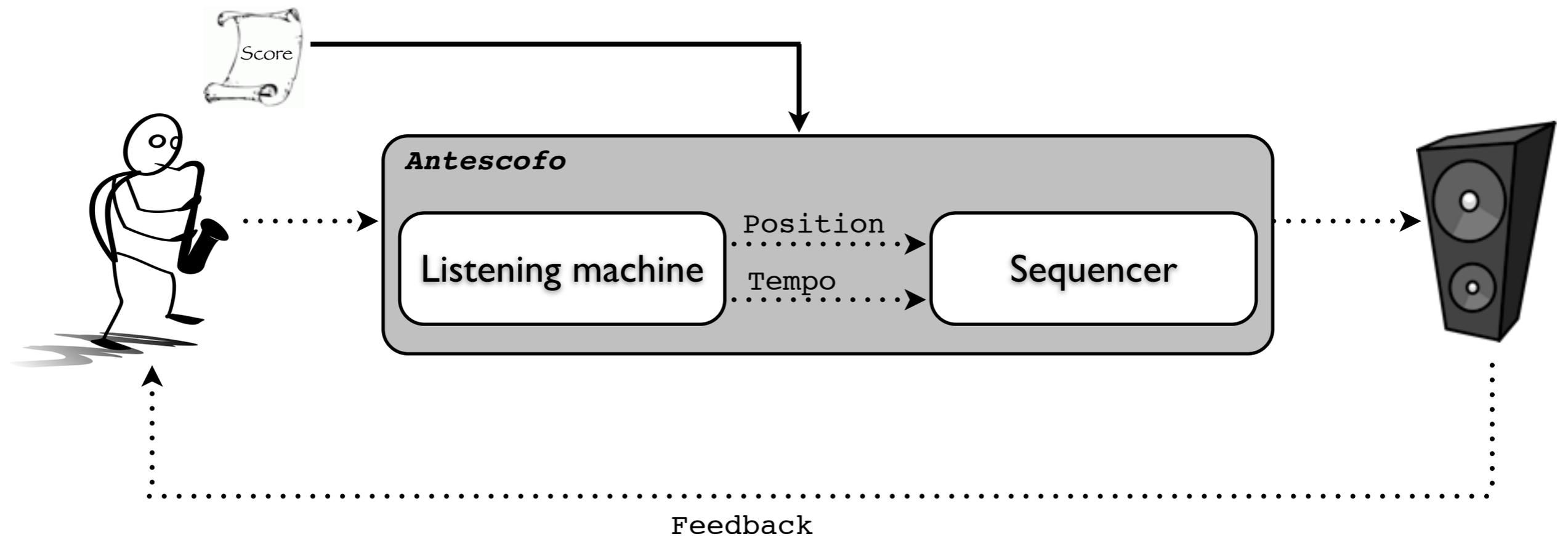
# Mixed Music and Antescofo

[Cont 2008]



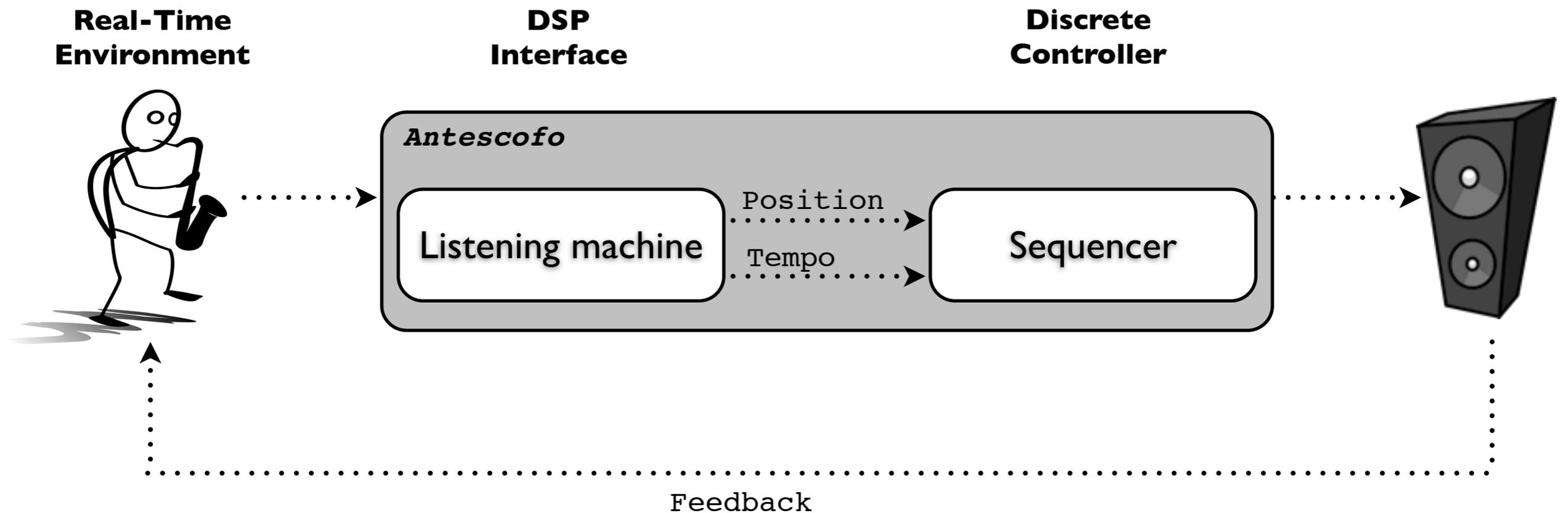
# Antescofo Architecture

[Cont 2008]



# Antescofo Architecture

[Cont 2008]



# The Antescofo Language

Goal: Jointly specify electronic and instrumental parts

Anthèmes II (1994)

**Libre brusque** (♩ = 92) **Pierre Boulez** (\*1925)

*f* *fff* *mf* *ff* *rall.* (♩ = 92) *batt. (archet normal)* (♩ = 66)

**Violon**

Spatialization: F -11/-18/-18/2.0

① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨

**Inf. Rev.**

reverb. time: 60"

Spatialization: F -11/-18/-18/2.0

♩ = 90 msec.

**Sampl. IR**

MIDI: 93 90 85 84 82 80 75 77 75 74

reverb. time: 60"

Spatialization: F -11/-18/-18/2.0

**Sampler**

♩ = 93 msec. *pizz.*

MIDI: [74 73 70 69 68 67 66 65]

[74 73] [74 71 70] [69 70 73 74] [74 73 72 69 68] [67 68 71 72 73 74] [63 64 67 68 69 70 71 74]

Spatialization: MR -4/-12/-24/2.0

**Freq. Shift.**

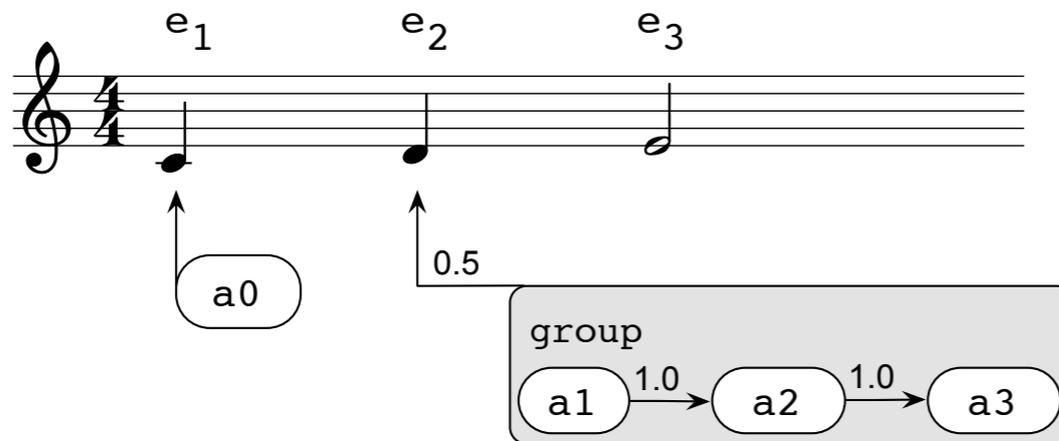
Spatialization:

New version using antescofo (2008)

# The Antescofo Language

Goal: Jointly specify electronic and instrumental parts

[Echeveste et al. 2012]



```
NOTE 60 1.0
0.0 'a_0'

NOTE 62 1.0
0.5 GROUP loose causal
    { 0.0 'a_1'
      1.0 'a_2'
      1.0 'a_3' }

NOTE 64 2.0
```

- Time is relative to the tempo
- Electronic actions are characterized by a delay
- Hierarchical structure: *groups and nested groups*
- Synchronization with the musician : *tight, loose*
- Error handling strategies : *partial, causal*

# Live Coding

Modify, correct and interact with the score during the performance

# Automatic Accompaniment

The house of the rising sun

The image displays a musical score for the song 'The house of the rising sun'. It consists of two staves of music in 4/4 time. The first staff contains measures 1 through 7, and the second staff contains measures 8 through 14. Chord progressions are indicated below the notes: Am, C, D, F, Am, C, E for the first staff; and Am, C, D, F, Am, E, Am for the second staff. The melody is written in treble clef.

- **Functional programming**  
modular definition of the accompaniment
- **Reactive programming**  
interaction with the score during the performance

# Definitions

## 1. Define the bass line

```
let bass = [0.0, (A, Min); 2.0, (C, Maj); ...]  
val bass: (delay * chord) list
```

## 2. Define the accompaniment style

```
let arpeggio chord =  
  ...  
  group Loose Local  
    [0.0, action_note (fond);  
     1.0, action_note (third);  
     2.0, action_note (fifth);}]  
val arpeggio: chord -> asco_event
```

## 3. Link with the performance

```
let process basic_accomp =  
  run (link asco 2 roots)  
val basic_accomp: unit process
```

# Kill a Process

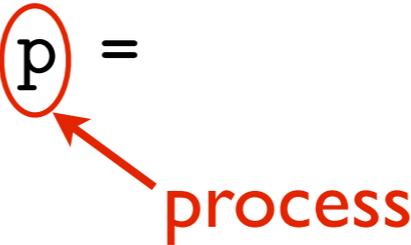
Example of a higher-order process

```
let process killable k p =  
  do  
    run p  
  until k done  
val killable:  
  (unit, unit) event -> unit process ->  
  unit process
```

# Kill a Process

Example of a higher-order process

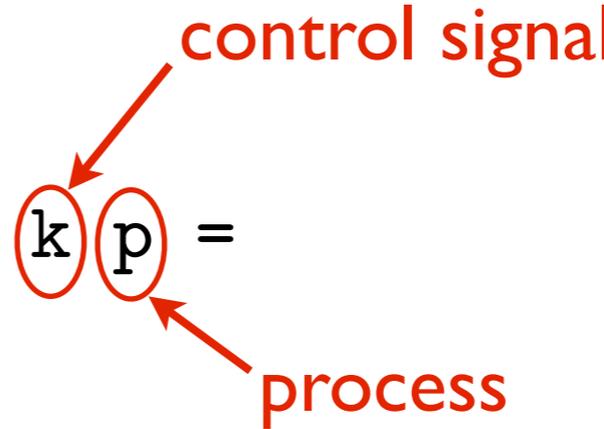
```
let process killable k (p) =  
  do  
    run p  
    until k done  
val killable:  
  (unit, unit) event -> unit process ->  
  unit process
```



# Kill a Process

Example of a higher-order process

```
let process killable (k) (p) =  
  do  
    run p  
    until k done  
val killable:  
  (unit, unit) event -> unit process ->  
  unit process
```



# Dynamic Changes

Example of a recursive higher-order process

```
let process rec replaceable replace p =  
  do  
    run p  
  until replace (q) ->  
    run (replaceable replace q)  
done  
val replaceable:  
(unit process, unit process) event ->  
unit process -> unit process
```

# Dynamic Changes

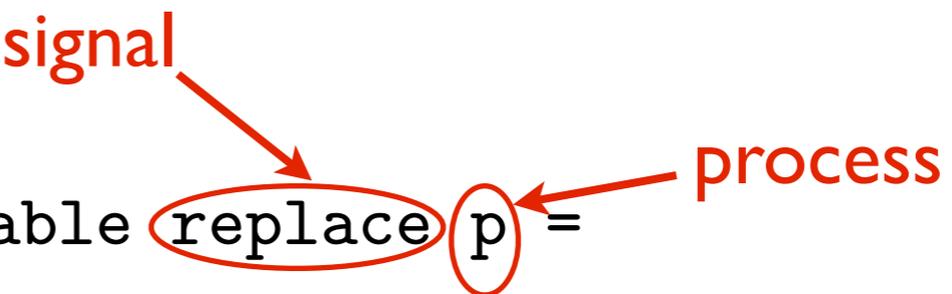
Example of a recursive higher-order process

```
let process rec replaceable replace (p) = 
do
  run p
until replace (q) ->
  run (replaceable replace q)
done
val replaceable:
  (unit process, unit process) event ->
  unit process -> unit process
```

# Dynamic Changes

Example of a recursive higher-order process

```
let process rec replaceable replace (p) = process
do
  run p
until replace (q) ->
  run (replaceable replace q)
done
val replaceable:
  (unit process, unit process) event ->
  unit process -> unit process
```



# Dynamic Changes

Example of a recursive higher-order process

```
let process rec replaceable replace (p) =  
  do  
    run p  
  until replace (q) ->  
    run (replaceable replace q)  
done  
val replaceable:  
  (unit process, unit process) event ->  
  unit process -> unit process
```

signal

process

new behavior

signal can carry processes

# New Reactive Behaviors

Example: Steve Reich's Piano Phase

# Piano Phase ...

## Piano Phase,

pour 2 pianos ou 2 marimbas

Steve Reich

$\text{♩} = \text{ca. } 72$

**Bob**

**Alice**

1 (x4-8) r.h. l.h. mf non legato

2 (x12-18) r.h. l.h. fade in non legato mf

3 (x4-16) (x4-16) hold tempo 1 (tempo 1)

4 (x16-24) (x4-16) (tempo 1)

5 (x16-24) (x4-16) (tempo 1)

6 (x16-24) (x4-16) (tempo 1)

accel very slightly hold tempo 1 a.v.s.

hold tempo 1 a.v.s. hold tempo 1 a.v.s. hold tempo 1 a.v.s.

# Piano Phase ...

## Piano Phase,

pour 2 pianos ou 2 marimbas

Steve Reich

$\text{♩} = \text{ca. } 72$

**Bob**

**Alice**

The musical score for 'Piano Phase' by Steve Reich is presented for two parts: Bob (top staff) and Alice (bottom staff). The score consists of six measures, each with a measure number and a range in parentheses: 1 (x4-8), 2 (x12-18), 3 (x16-24), 4 (x16-24), 5 (x16-24), and 6 (x16-24). The notation includes right-hand (r.h.) and left-hand (l.h.) parts, dynamics such as *mf* non legato, and performance instructions like 'fade in', 'hold tempo 1', and 'accel very slightly'. A red arrow points to a synchronization point in measure 2, where the two parts align. The score is written in treble clef with a key signature of one sharp (F#).

**Synchronization**

# Piano Phase ...

## Piano Phase,

pour 2 pianos ou 2 marimbas

Steve Reich

$\text{♩} = \text{ca. } 72$

**Bob**

**Alice**

1 (x4-8) 2 (x12-18) 3 (x4-16) 4 (x16-24) 5 (x4-16) 6 (x16-24)

r.h. l.h. mf non legato r.h. l.h. mf non legato

fade in non legato mf hold tempo 1 accel very slightly hold tempo 1 a.v.s. hold tempo 1 (tempo 1) a.v.s. hold tempo 1 a.v.s. hold tempo 1 a.v.s.

**Desynchronization**

# Piano Phase ...

**Bob**

**Alice**

Piano Phase,  
pour 2 pianos ou 2 marimbas

$\text{♩} = \text{ca. } 72$

1 (x4-8) 2 (x12-18)

r.h. l.h. mf non legato r.h. l.h. fade in non legato

4 (x16-24) 5 (x16-24)

(tempo1) (tempo1) (tempo1)

hold tempo 1 accel very slightly

(x4-16) (tempo1) a.v.s. (tempo1) (tempo1) (tempo1) a.v.s. a.v.s.

Steve Reich

**Desynchronization**

# Piano Phase ...

## Piano Phase,

pour 2 pianos ou 2 marimbas

$\text{♩} = \text{ca. } 72$

Steve Reich

**Bob**

**Alice**

1 (x4-8) r.h. l.h. mf non legato

2 (x12-18) r.h. l.h. fade in non legato mf

3 (x4-16) (x16-24) hold tempo 1 accel very slightly hold tempo 1 a.v.s.

4 (x16-24) (x4-16) (tempo 1)

5 (x16-24) (x4-16) (tempo 1)

6 (x16-24) (x4-16) (tempo 1)

hold tempo 1 a.v.s. hold tempo 1 a.v.s. hold tempo 1 a.v.s.

# Piano Phase ...

## Piano Phase,

pour 2 pianos ou 2 marimbas

Steve Reich

$\bullet = \text{ca. } 72$

**Bob**

**Alice**

1 (x4-8) 2 (x12-18) (x4-16) 3 (x16-24) (x4-16)

r.h. l.h. mf non legato r.h. l.h. accel very slightly hold tempo 1 (tempo 1) a.v.s.

fade in non legato mf hold tempo 1

4 (x16-24) (x4-16) 5 (x16-24) (x4-16) 6 (x16-24) (x4-16)

(tempo 1) (tempo 1) (tempo 1) (tempo 1)

hold tempo 1 a.v.s. hold tempo 1 a.v.s. hold tempo 1 a.v.s.

# Piano Phase ...

## Piano Phase,

pour 2 pianos ou 2 marimbas

Steve Reich

$\text{♩} = \text{ca. } 72$

**Bob**

**Alice**

1 (x4-8) 2 (x12-18) 3 (x4-16) 3 (x16-24) (x4-16)

r.h. l.h. hold tempo 1 (tempo 1)

*mf* non legato r.h. l.h. accel very slightly hold tempo 1 a.v.s.

fade in non legato *mf*

4 (x16-24) 5 (x16-24) 6 (x16-24)

(x4-16) (tempo 1) (tempo 1) (tempo 1)

hold tempo 1 a.v.s. hold tempo 1 a.v.s. hold tempo 1 a.v.s.

# Piano Phase ...

## Piano Phase,

pour 2 pianos ou 2 marimbas

Steve Reich

$\text{♩} = \text{ca. } 72$

**Bob**

**Alice**

1 (x4-8) 2 (x12-18) (x4-16) 3 (x16-24) (x4-16)

r.h. l.h. mf non legato r.h. l.h. hold tempo 1 (tempo 1) accel. very slightly hold tempo 1 a.v.s.

fade in non legato mf

4 (x16-24) (x4-16) (x16-24) (x4-16) 6 (x16-24) (x4-16)

(tempo 1) (tempo 1) (tempo 1)

hold tempo 1 a.v.s. hold tempo 1 a.v.s. hold tempo 1 a.v.s.

# Piano Phase ...

## Piano Phase,

pour 2 pianos ou 2 marimbas

Steve Reich

$\text{♩} = \text{ca. } 72$

**Bob**

**Alice**

1 (x4-8) 2 (x12-18) 3 (x16-24) (x4-16) (x4-16)

r.h. l.h. hold tempo 1 (tempo 1)

*mf* non legato r.h. l.h. accel very slightly hold tempo 1 a.v.s.

fade in non legato *mf*

4 (x16-24) 5 (x16-24) 6 (x16-24) (x4-16) (x4-16) (x4-16)

(tempo 1) (tempo 1) (tempo 1)

hold tempo 1 a.v.s. hold tempo 1 a.v.s. hold tempo 1 a.v.s.

# Piano Phase ...

## Piano Phase,

pour 2 pianos ou 2 marimbas

Steve Reich

$\text{♩} = \text{ca. } 72$

**Bob**

**Alice**

**Problem:**  
We do not want to compute a priori  
when resynchronizations will occur

# ... in Mixed Music

## Live musician

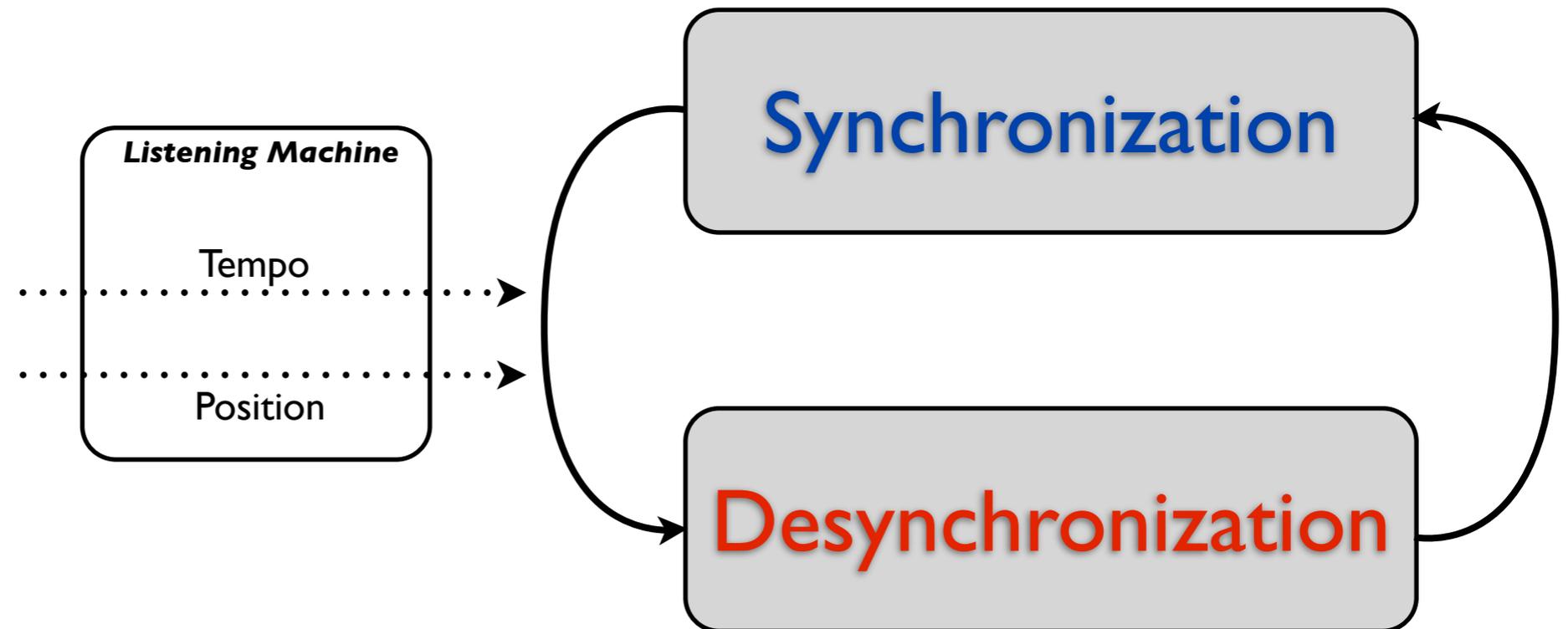
Plays the constant speed part



**Bob**

## Electronic

Handles the desynchronization



**Alice**

# ... in Mixed Music

## Live musician

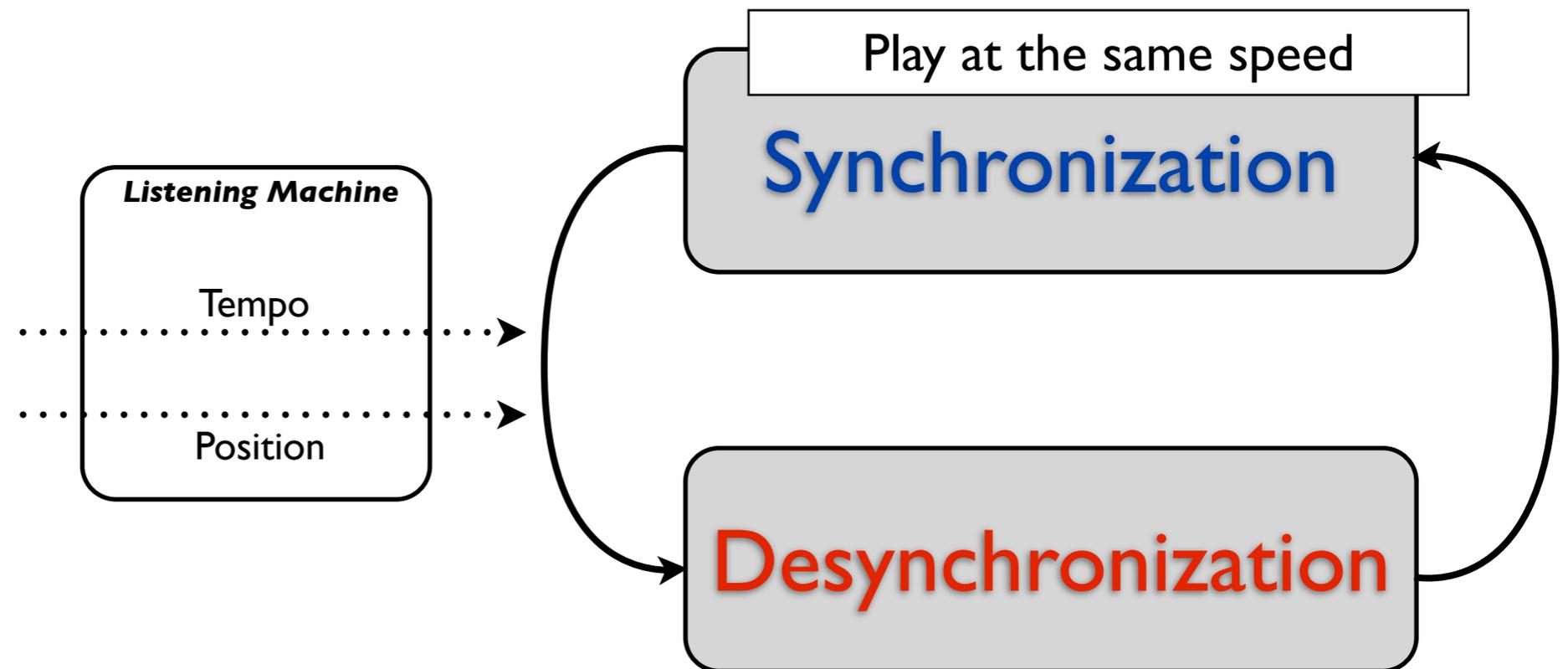
Plays the constant speed part



**Bob**

## Electronic

Handles the desynchronization



**Alice**

# ... in Mixed Music

## Live musician

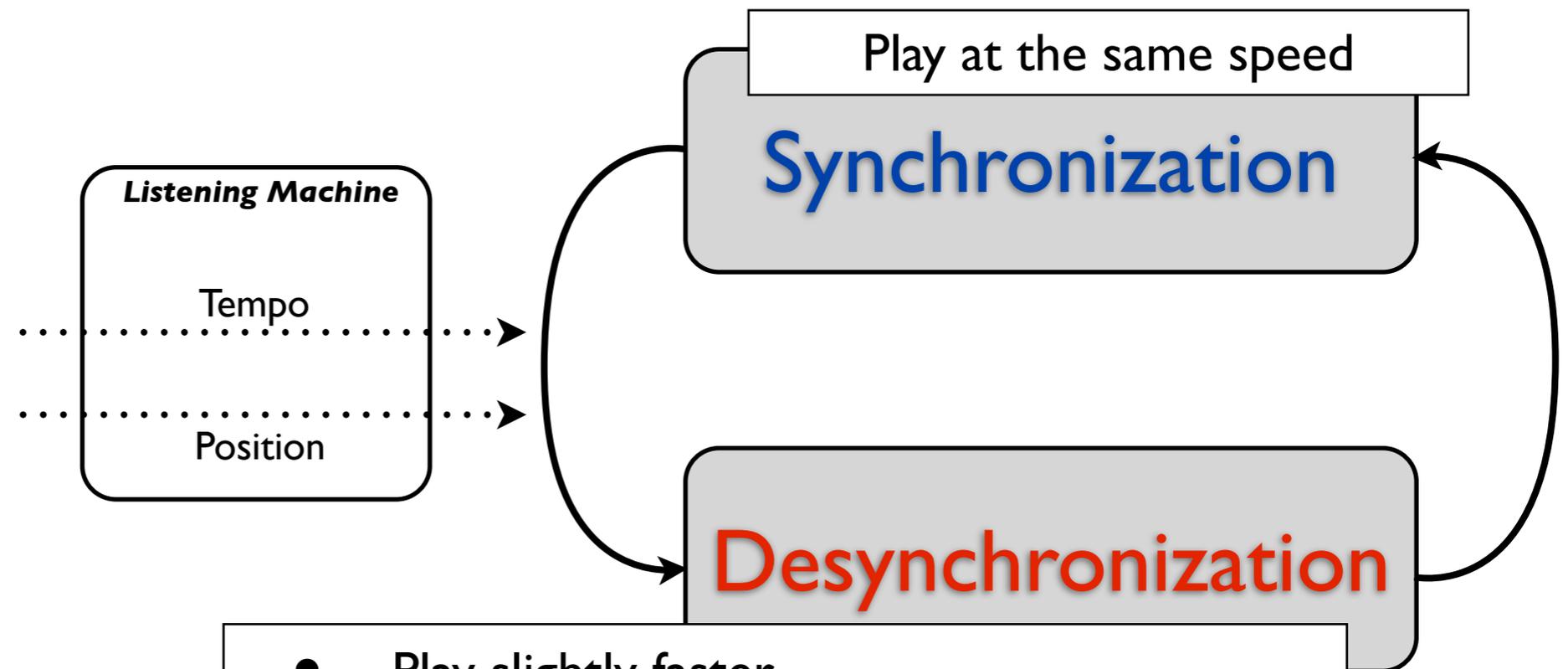
Plays the constant speed part



**Bob**

## Electronic

Handles the desynchronization



- Play slightly faster
- Track the first note of Bob
- Resynchronize when the k-th note of Alice is close enough of the first note of Bob

# Implementation

**Two phases:**  
**Synchronization**  
**Desynchronization**

```
let piano_phase sync desync first_note kth_note =
  let rec process piano_phase k =
    let ev = last_event asco in
    run (melody ev 4 0.25 first_note);
    emit desync;
  do
    let ev = last_event asco in
    run (melody (ev+1) 16 0.2458 first_note) ||
    run (track asco k kth_note) ||
    run (compare asco first_note kth_note sync 0.05)
  until sync done;
  run (piano_phase ((k + 1) mod 12))
in
piano_phase 1
in
```

# Implementation

## **Synchronization**

*Play the melody four times  
and follow the tempo*

*Emit the signal `desync` after  
four iterations of the melody*

```
let piano_phase sync desync first_note kth_note =  
  let rec process piano_phase k =  
    let ev = last_event asco in  
    run (melody ev 4 0.25 first_note);  
    emit desync;  
  do  
    let ev = last_event asco in  
    run (melody (ev+1) 16 0.2458 first_note) ||  
    run (track asco k kth_note) ||  
    run (compare asco first_note kth_note sync 0.05)  
  until sync done;  
  run (piano_phase ((k + 1) mod 12))  
in  
piano_phase 1  
in
```

# Implementation

## Desynchronization

*Play slightly faster  
and emit the signal `first_note`  
whenever the first note is played*

*Track the `k`-th note of the musician*

*Compare the emission of signals  
`kth_note` and `first_note` and emit  
`sync` when they are close enough*

```
let piano_phase sync desync first_note kth_note =  
  let rec process piano_phase k =  
    let ev = last_event asco in  
    run (melody ev 4 0.25 first_note);  
    emit desync;  
  do  
    let ev = last_event asco in  
    run (melody (ev+1) 16 0.2458 first_note) ||  
    run (track asco k kth_note) ||  
    run (compare asco first_note kth_note sync 0.05)  
  until sync done;  
  run (piano_phase ((k + 1) mod 12))  
in  
piano_phase 1  
in
```

# Retour sur les choix

- Choix du langage hôte
- Choix du modèle de concurrence
- Choix de faire un langage

encore une démo

<http://reactiveml.org>