Programming Mixed Music in ReactiveML

Guillaume Baudart, ENS
Louis Mandel, ENS
Marc Pouzet, ENS

FARM'13

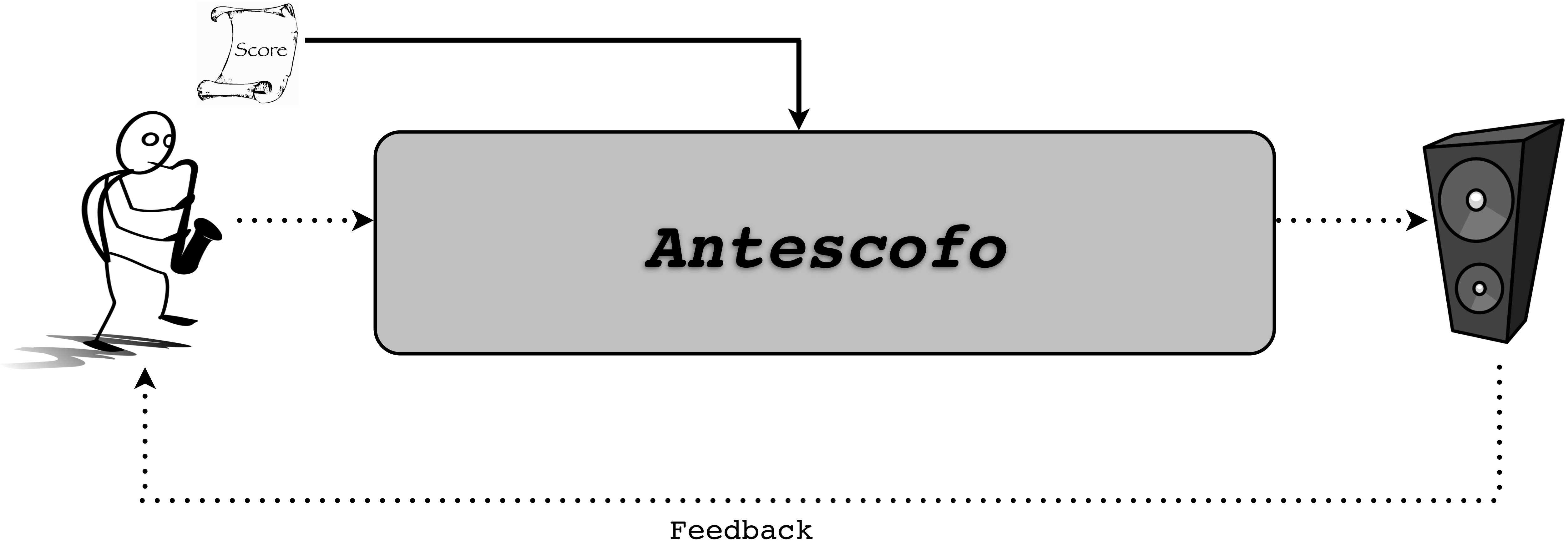
Boston, USA 09.28.2013

Mixed Music and Antescofo

[Cont 2008]

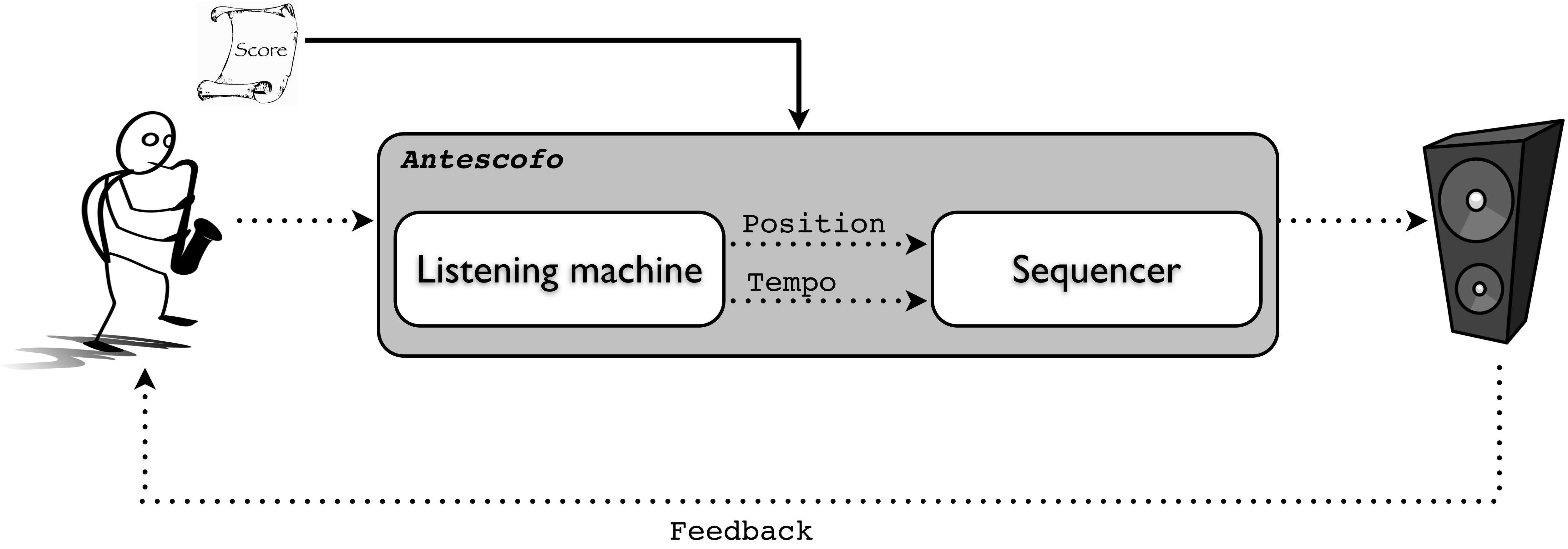
Mixed Music and Antescofo

[Cont 2008]



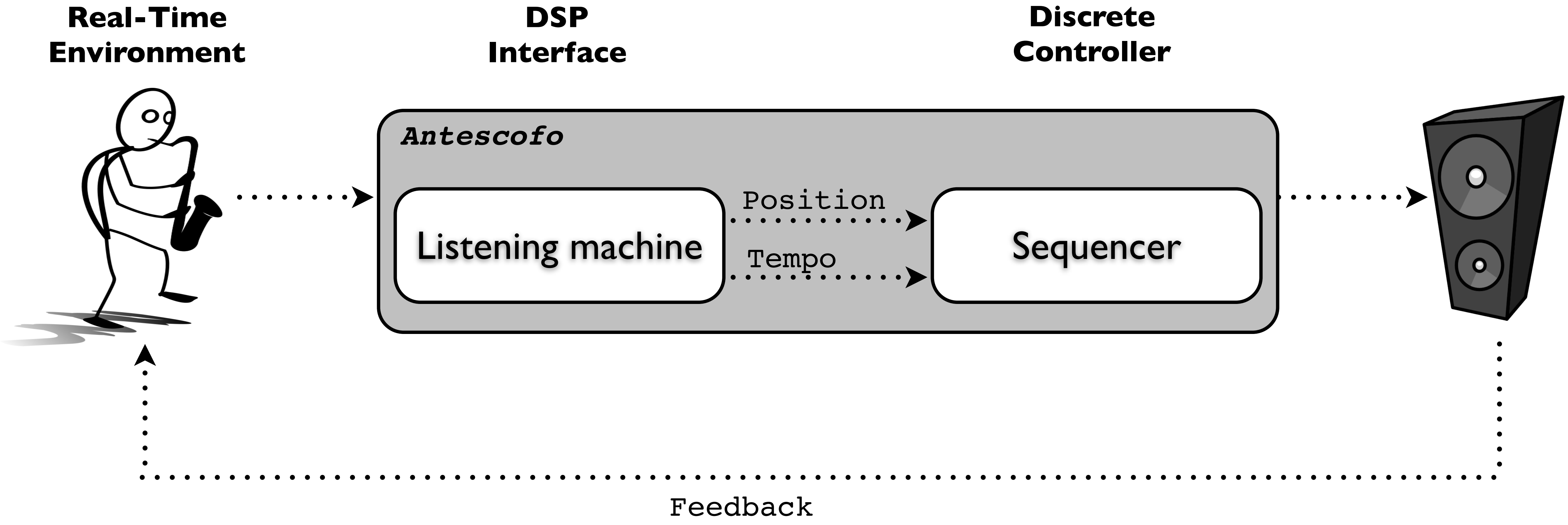
Antescofo Architecture

[Cont 2008]



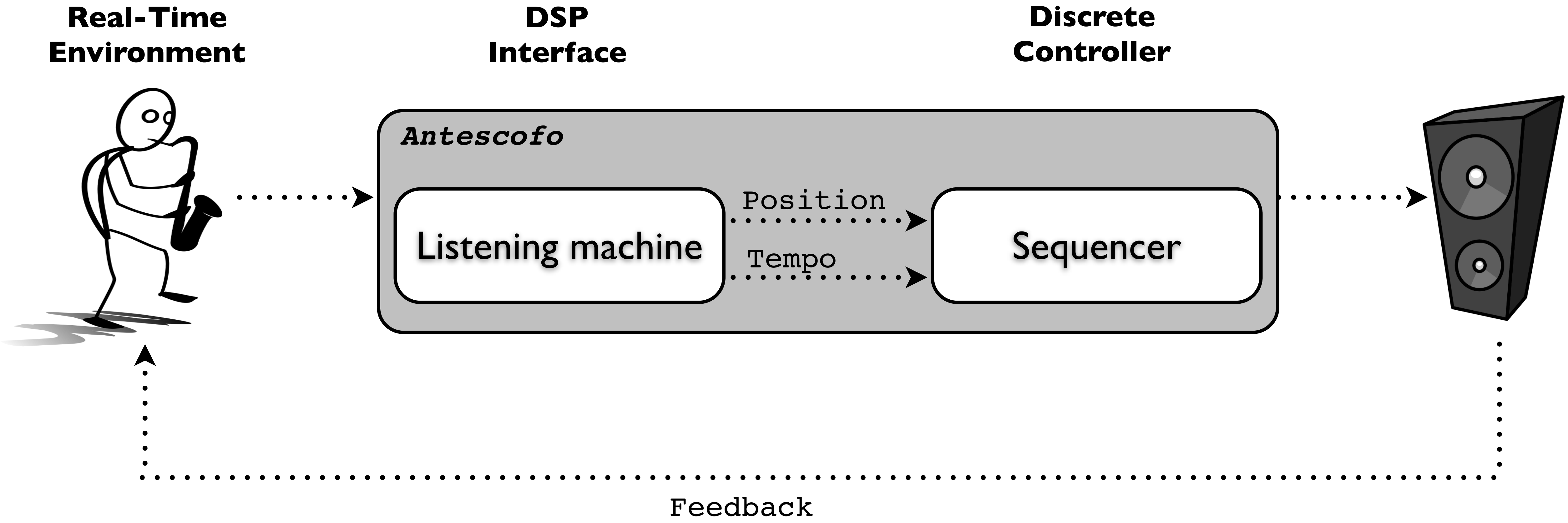
Antescofo Architecture

[Cont 2008]



Antescofo Architecture

[Cont 2008]

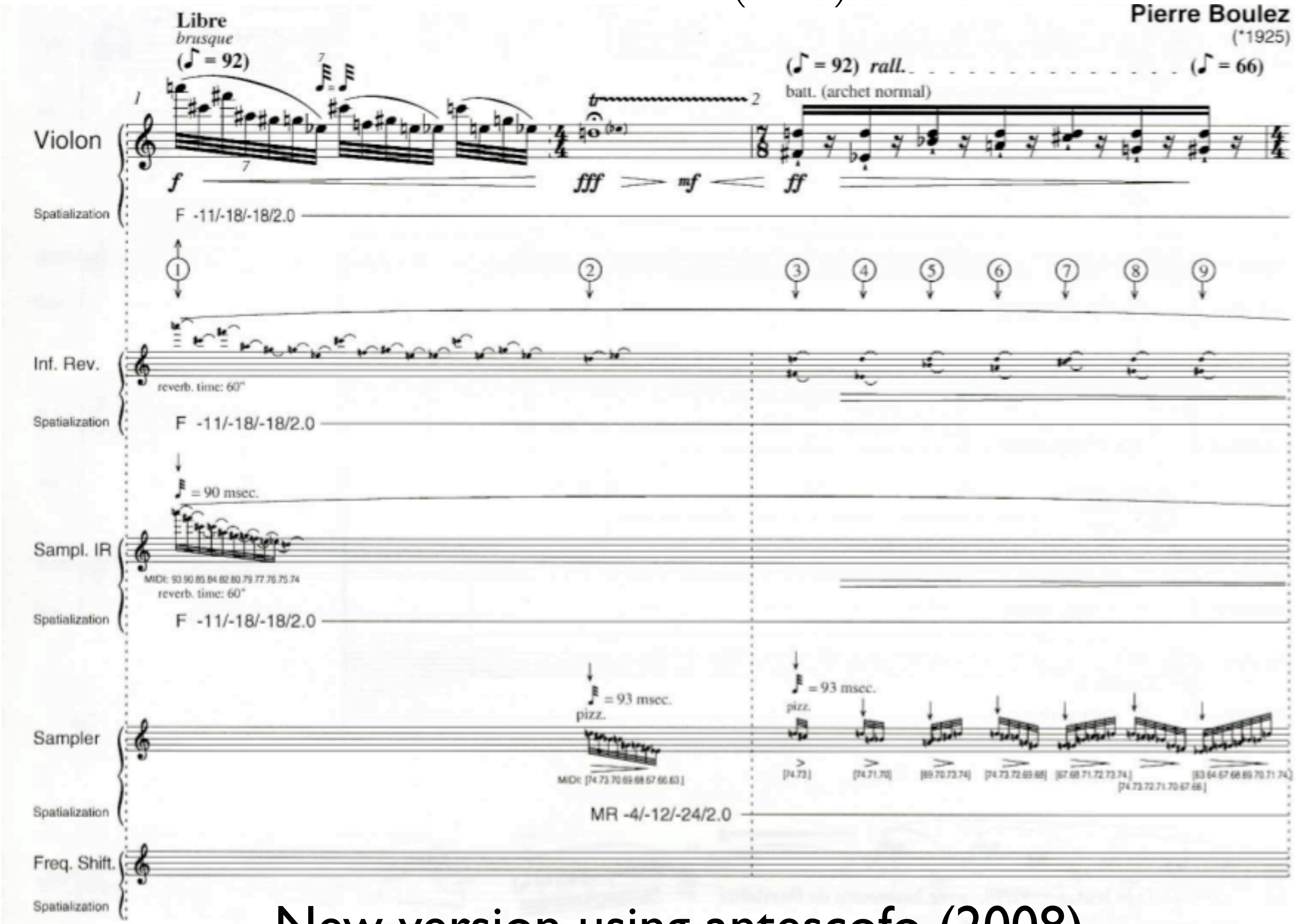


The score is a specification of a musical reactive system

The Antescofo Language

Goal: Jointly specify electronic and instrumental parts

Anthèmes II (1994)

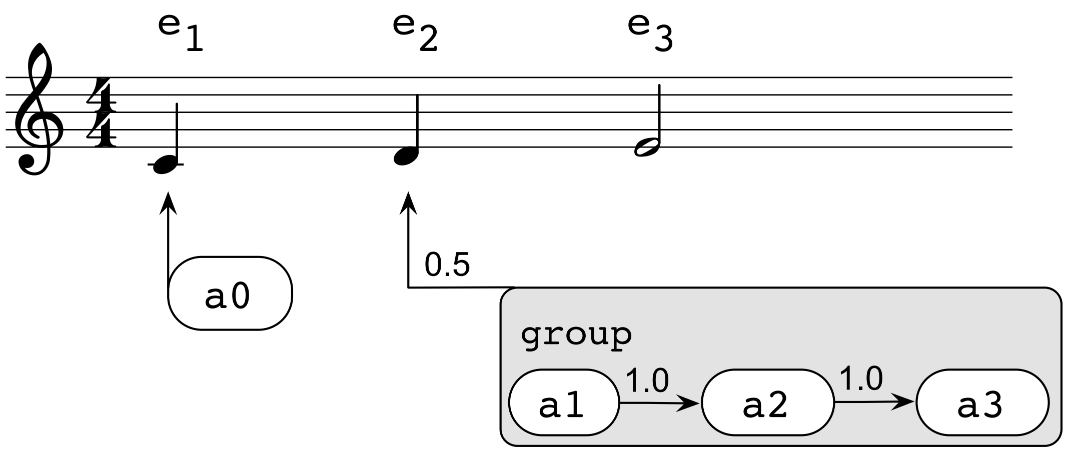


New version using antescofo (2008)

The Antescofo Language

Goal: Jointly specify electronic and instrumental parts

[Echeveste et al. 2012]



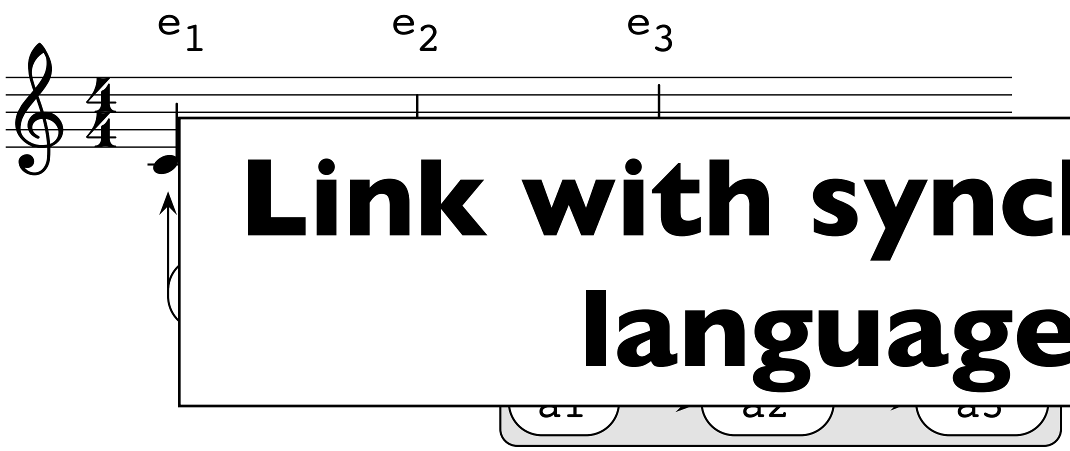
```
NOTE  60  1.0
0.0  'a_0'

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*

The Antescofo Language

Goal: Jointly specify electronic and instrumental parts

[Echeveste et al. 2012]



```
NOTE   60   1.0
0.0   'a_0'
                  )
                 oose  causal
                 1'
                 2'
                 3' }

NOTE   64   2.0
```

Link with synchronous languages?

- 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*

ReactiveML

The temporal expressiveness of synchronous languages with the power of functional programming

ReactiveML

[Mandel-Pouzet 2005]

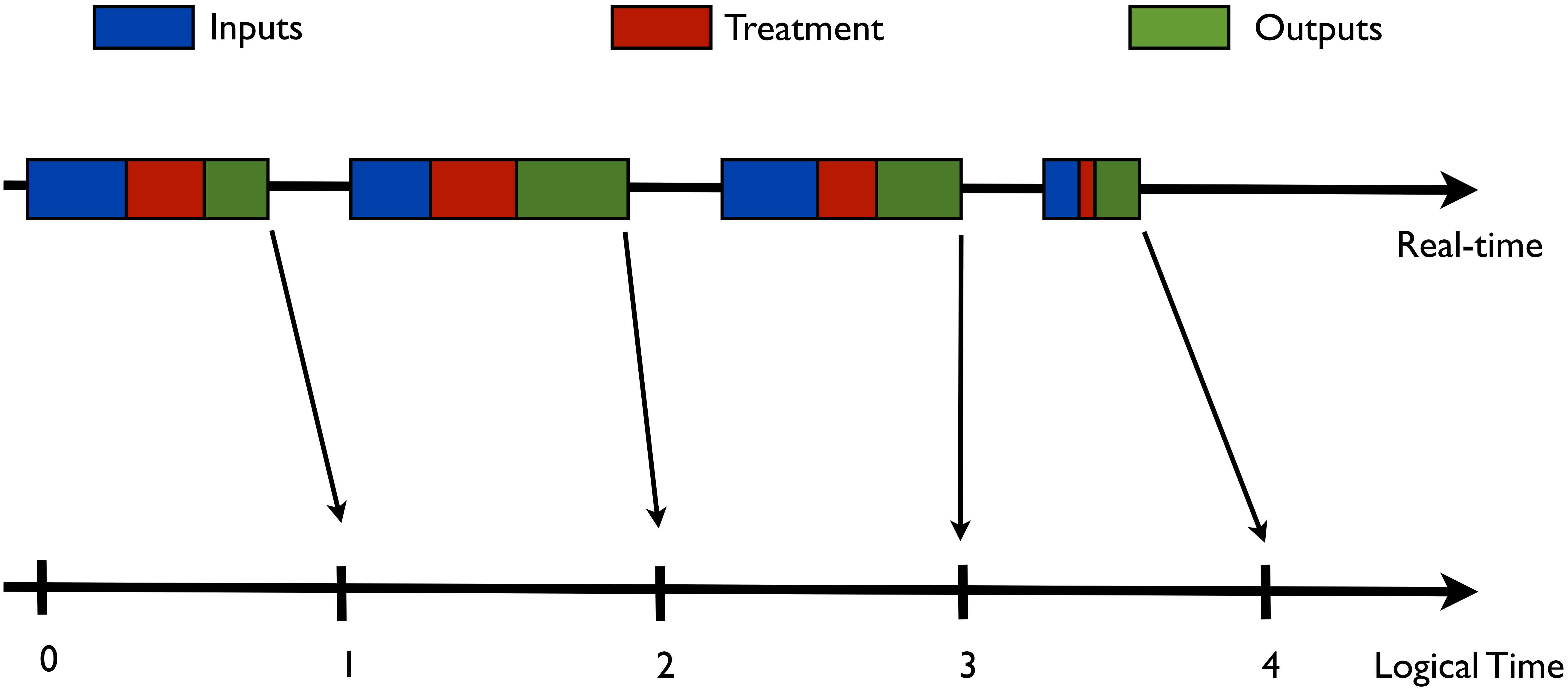
OCaml

- Data structures
- Control structures

Synchronous model of concurrency

- A global logical time
- Parallel composition
- Communication between processes

The Synchronous Hypothesis



The Language

Process

```
let process <id> {<pattern>} = <expr>
```

State machines, executed through several instants.
Simple OCaml functions are considered to be instantaneous.

Basics

Synchronization: `pause`
Execution: `run <expr>`

Composition

Sequence: `<expr> ; <expr>`
Parallelism: `<expr> || <expr>`

Signals

Definition: `signal <id>`
Emission: `emit <id>`
Waiting: `await <id>`

Broadcast communication
between processes

First Example

Wait in parallel for the emission of two signals

```
let process simple a b =
  (await a; print "a")
  ||
  (await b; print "b")
val simple:
  (unit, unit) event -> (unit, unit) event ->
    unit process
```

First Example

Wait in parallel for the emission of two signals

signals

```
let process simple a b =
  (await a; print "a")
  ||
  (await b; print "b")
val simple:
  (unit, unit) event -> (unit, unit) event ->
    unit process
```

First Example

Wait in parallel for the emission of two signals

signals

```
let process simple a b =
  (await a; print "a")
  ||
  (await b; print "b")
val simple:
  (unit, unit) event -> (unit, unit) event ->
    unit process
```

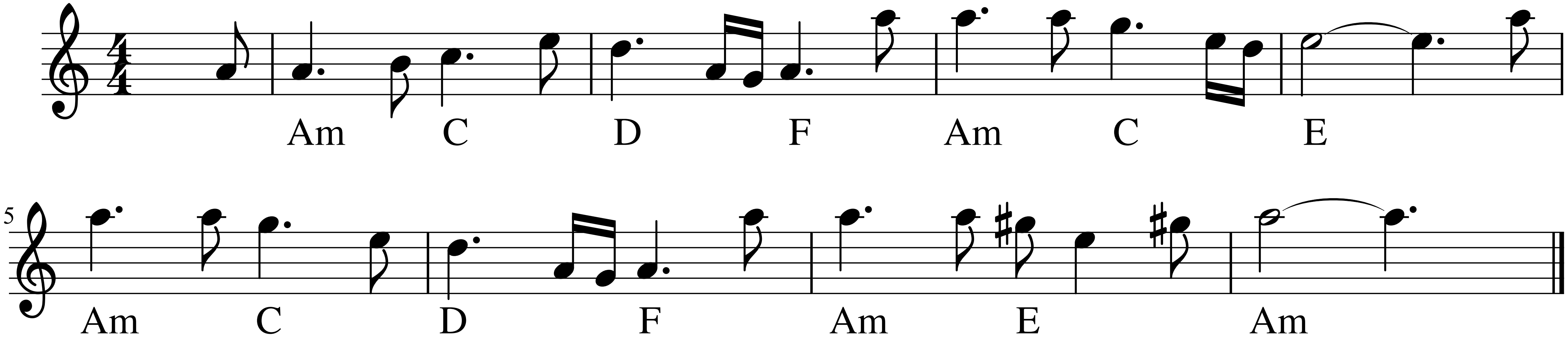
Parallel composition

Live Coding

Modify, correct and interact with the score
during the performance

Automatic Accompaniment

The house of the rising sun



- **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
```

Interactions

- **Kill a process when a signal is emitted**
  allow to modify the accompaniment
- **Suspend a the execution of a process**
  pause and resume a process with a signal
- **Dynamically change the behavior of a process**
  switch between different kinds of accompaniment

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

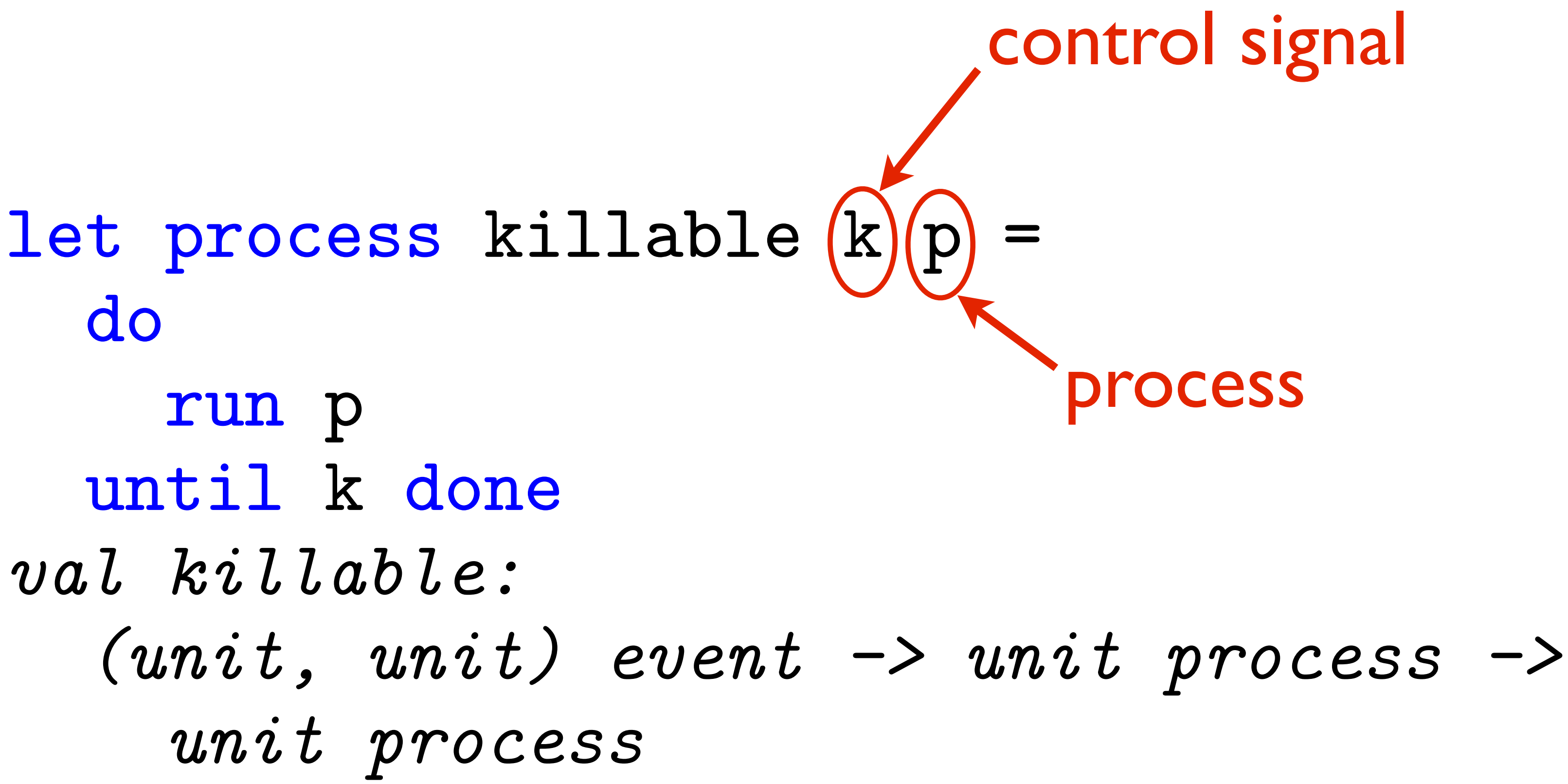
Example of a higher-order process

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

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
```

control signal

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

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

signal

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

signal

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
```

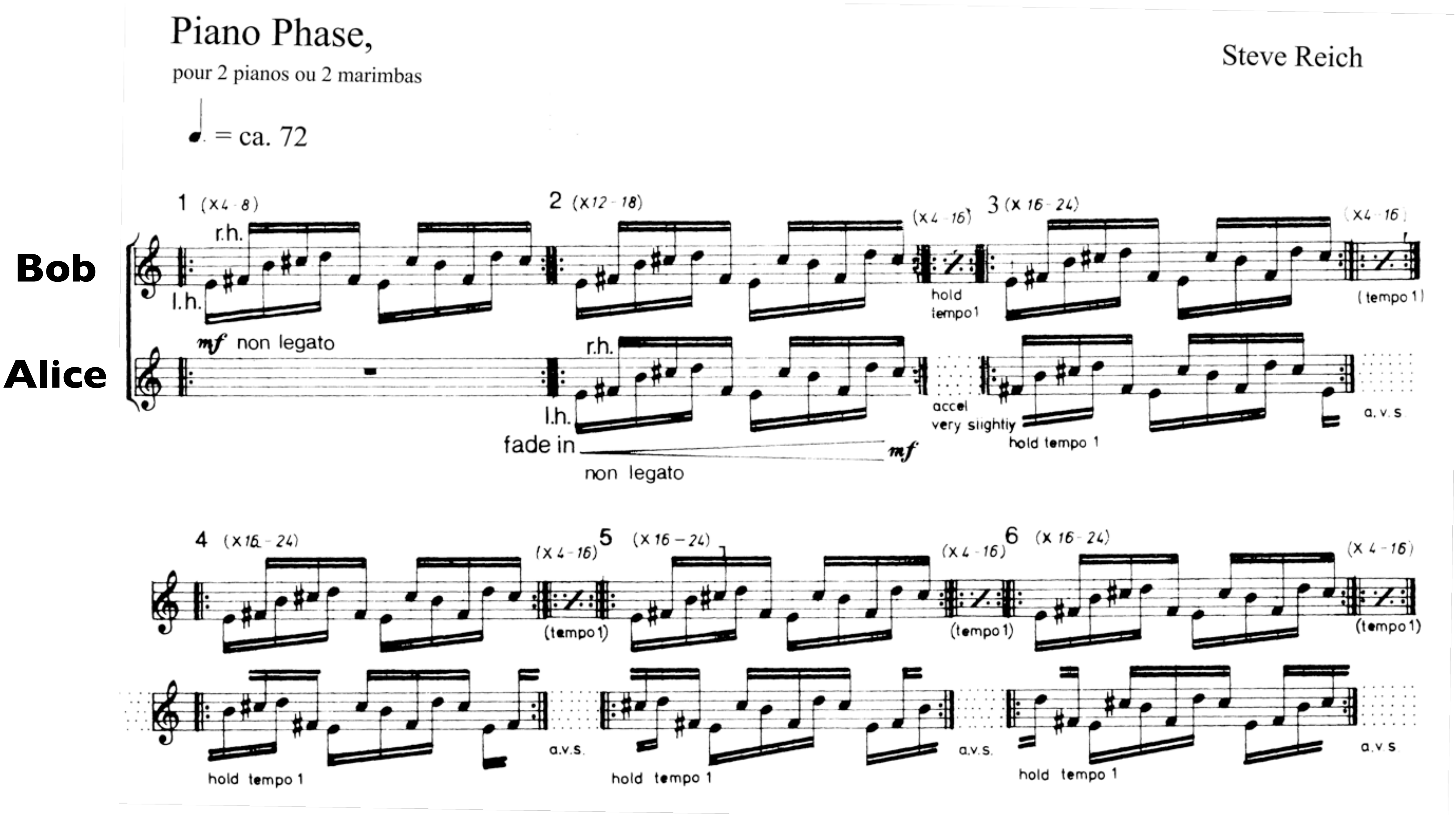
new behavior

signal can carry processes

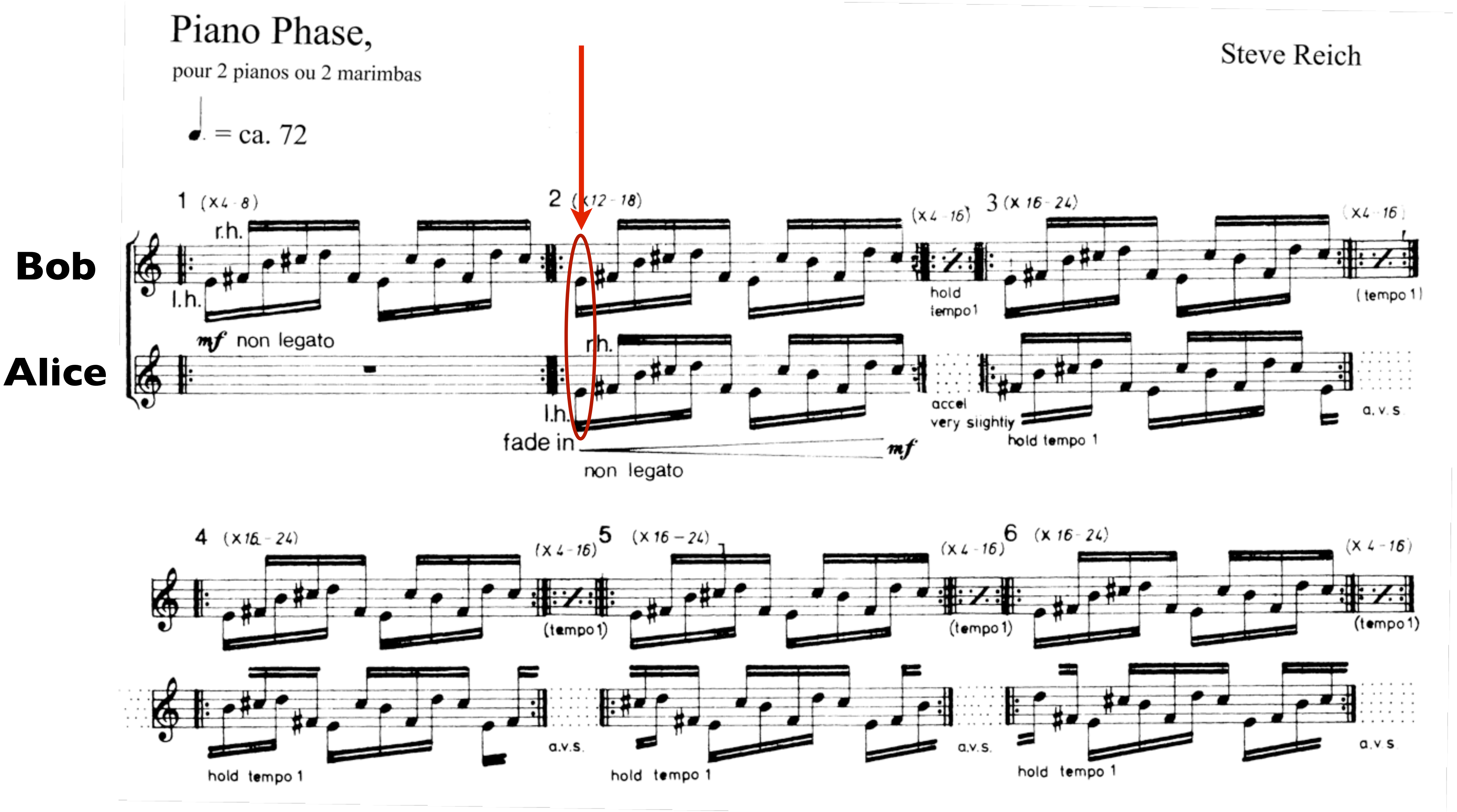
New Reactive Behaviors

Example: Steve Reich's Piano Phase

Piano Phase ...

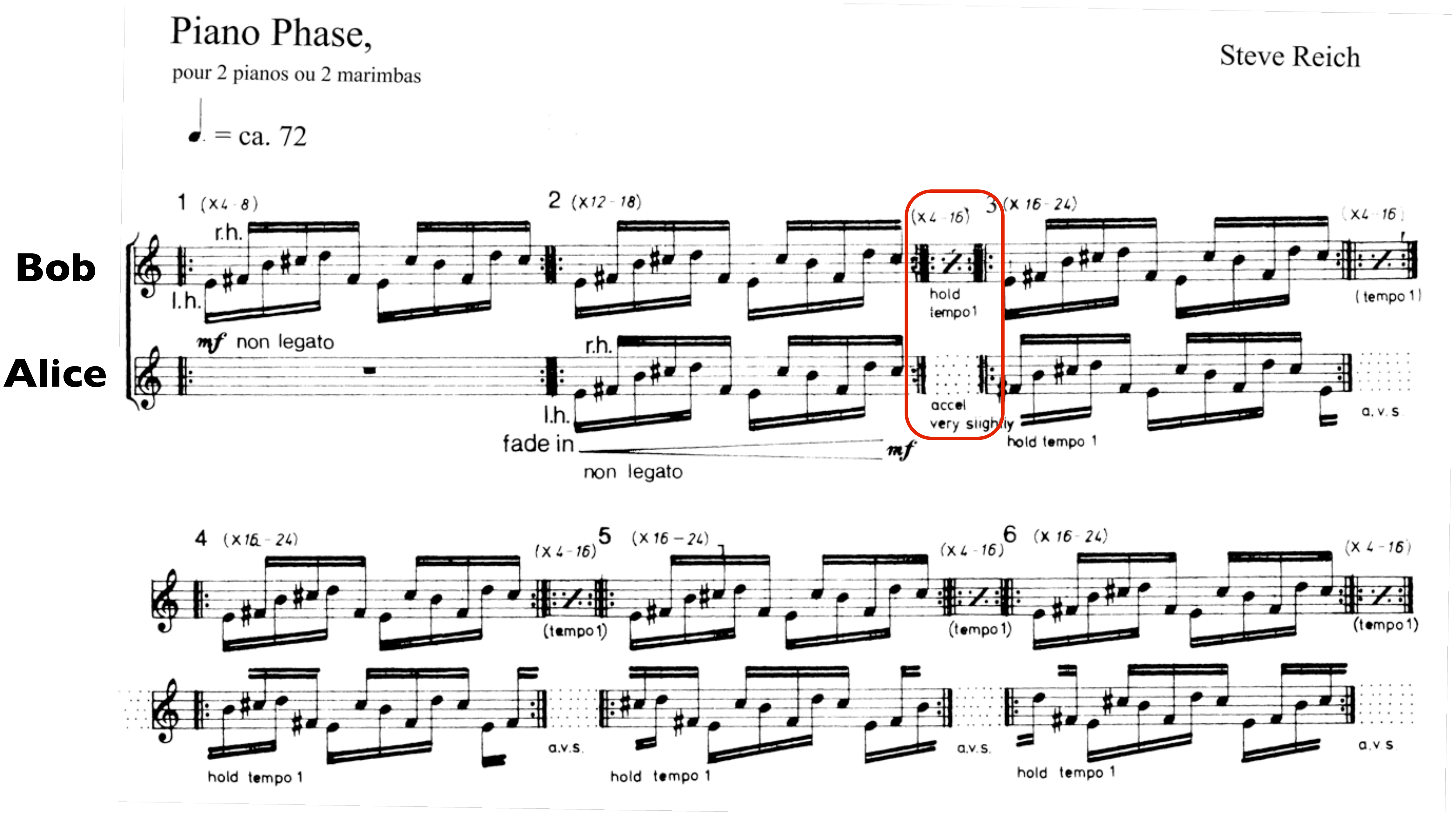

Piano Phase,
pour 2 pianos ou 2 marimbas
Steve Reich
♩. = ca. 72
Bob
Alice
1 (x4-8)
2 (x12-18)
(x4-16)
3 (x16-24)
(x4-16)
r.h.
l.h.
mf non legato
hold tempo 1
(tempo 1)
accel very slightly
a.v.s.
fade in
mf
non legato
4 (x16-24)
(x4-16)
5 (x16-24)
(x4-16)
6 (x16-24)
(x4-16)
(tempo 1)
hold tempo 1
a.v.s.

Piano Phase ...



Synchronization

Piano Phase ...

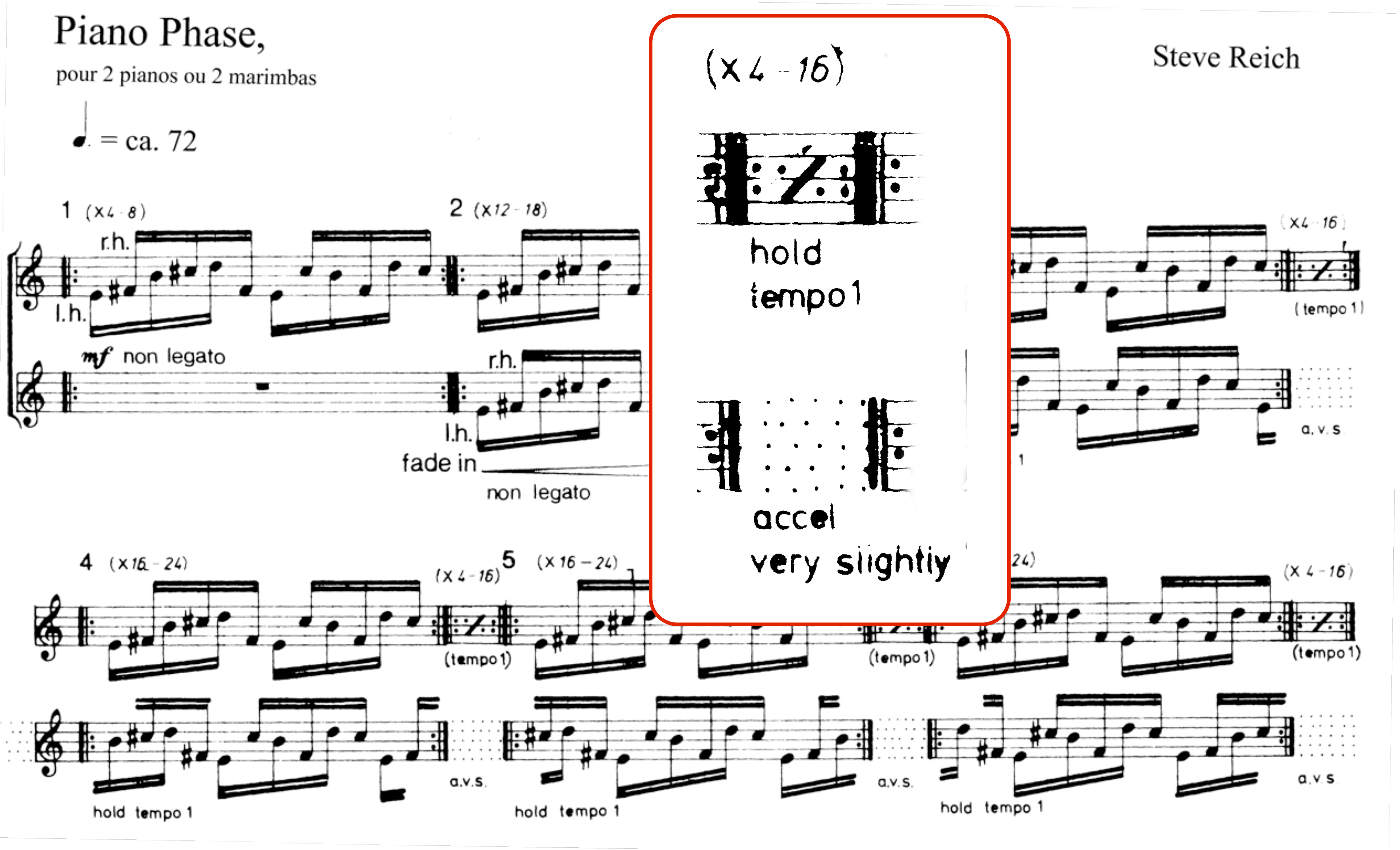


Desynchronization

Piano Phase ...

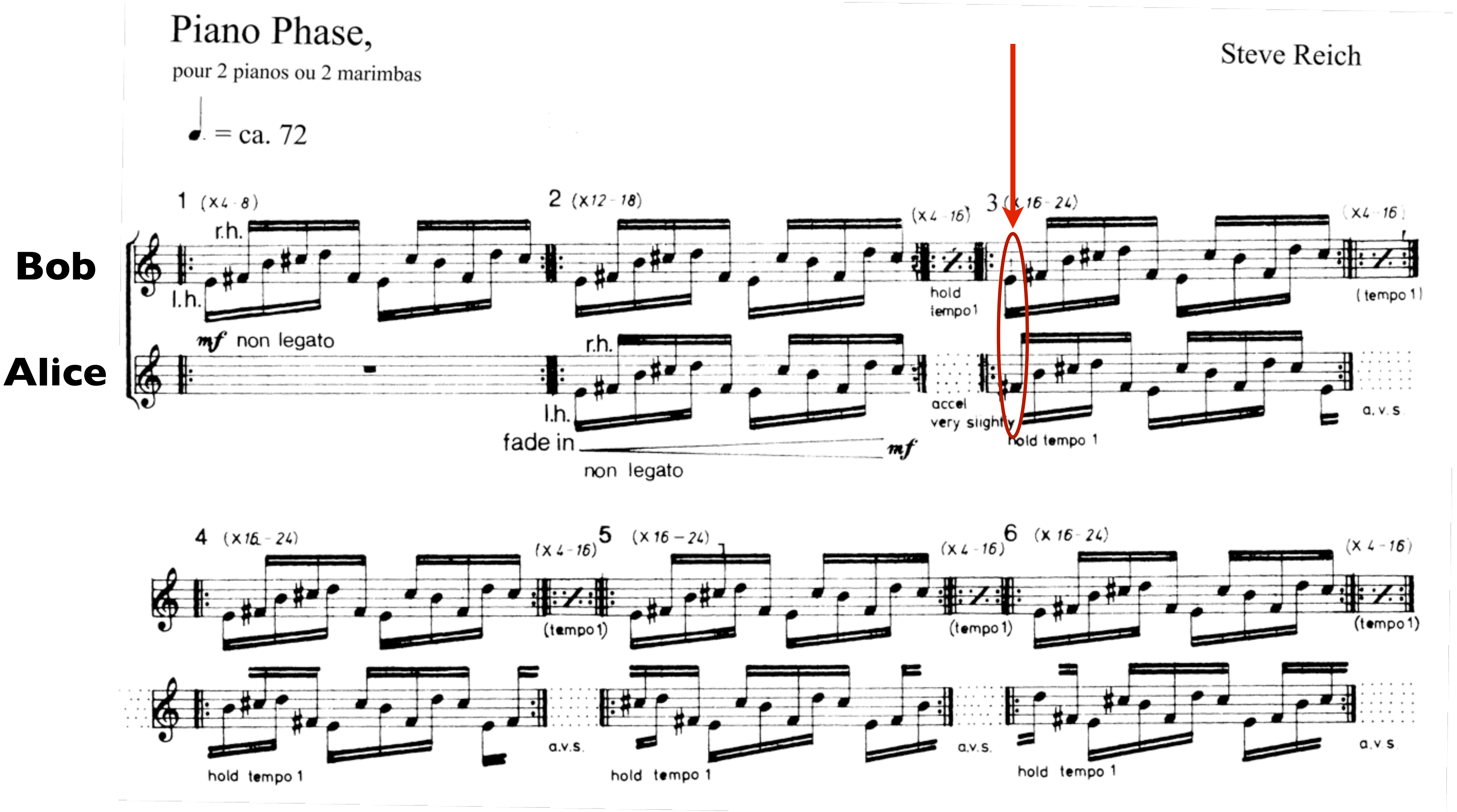
Bob

Alice



Desynchronization

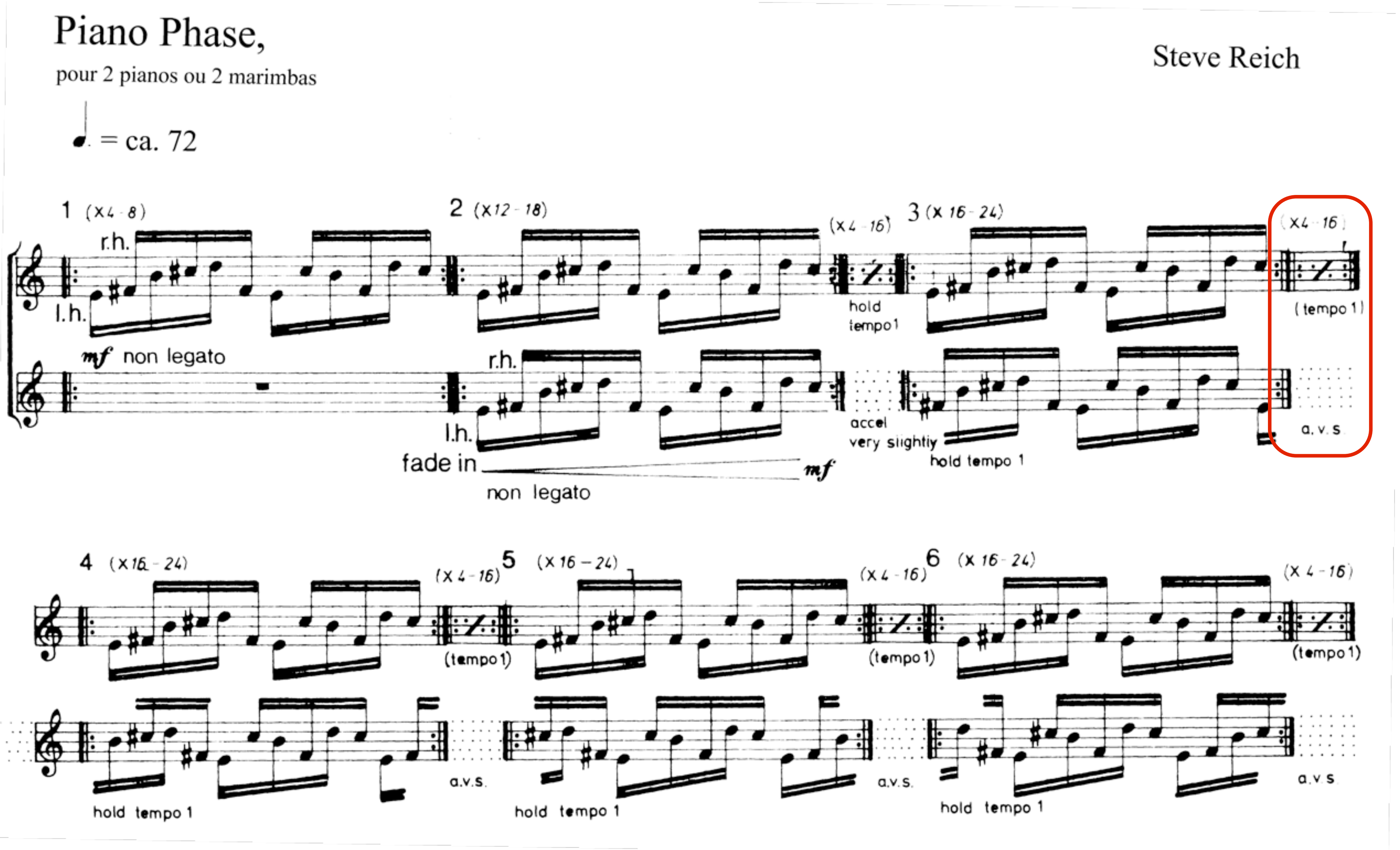
Piano Phase ...


Piano Phase,
pour 2 pianos ou 2 marimbas
Steve Reich
♩. = ca. 72
Bob
Alice
1 (x4-8)
2 (x12-18)
(x4-16)
3 (x16-24)
(x4-16)
r.h.
l.h.
mf non legato
hold tempo 1
(tempo 1)
accel very slightly
hold tempo 1
a.v.s.
fade in
mf
non legato
4 (x16-24)
(x4-16)
5 (x16-24)
(x4-16)
6 (x16-24)
(x4-16)
(tempo 1)
hold tempo 1
a.v.s.

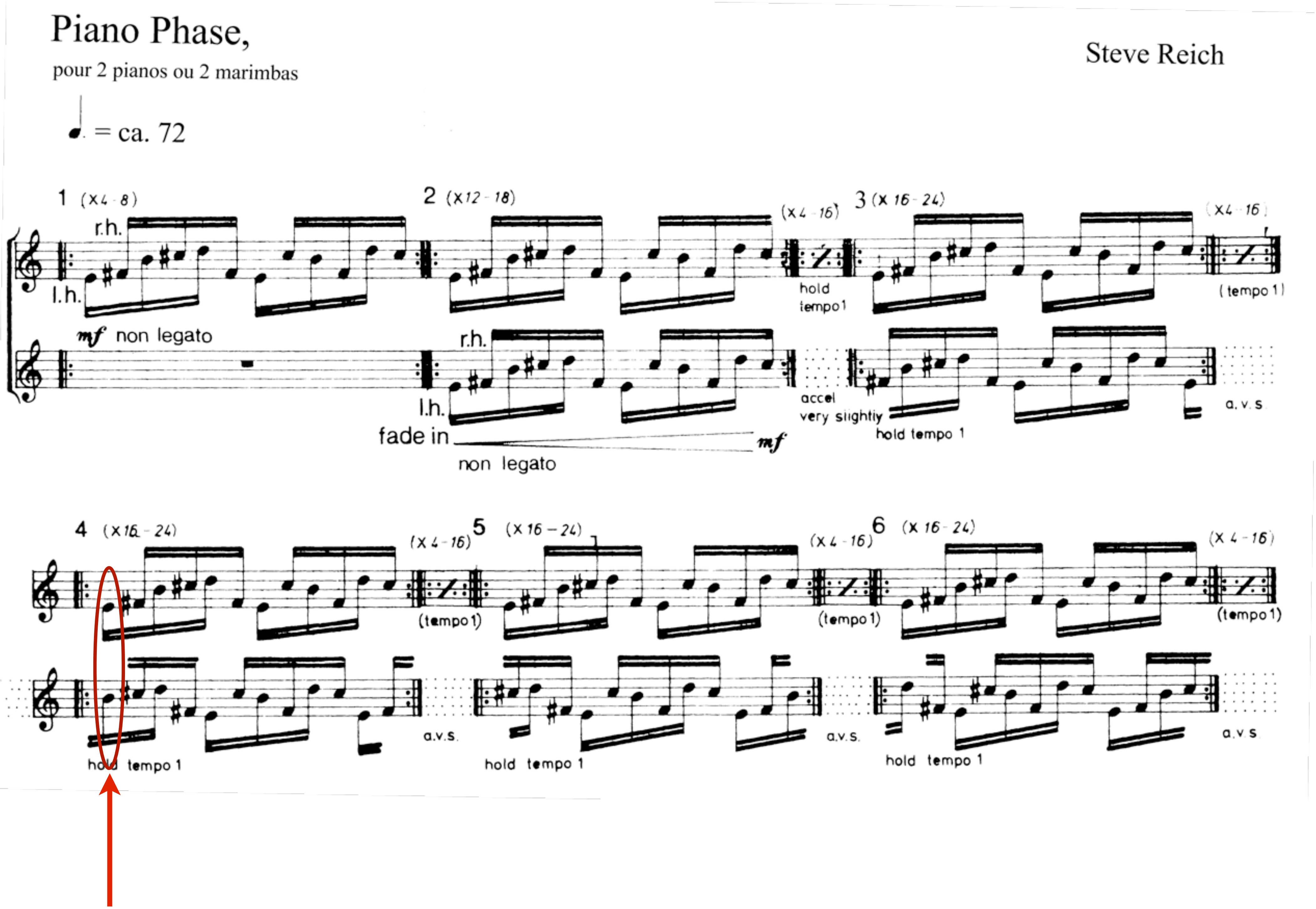
Piano Phase ...

Bob

Alice



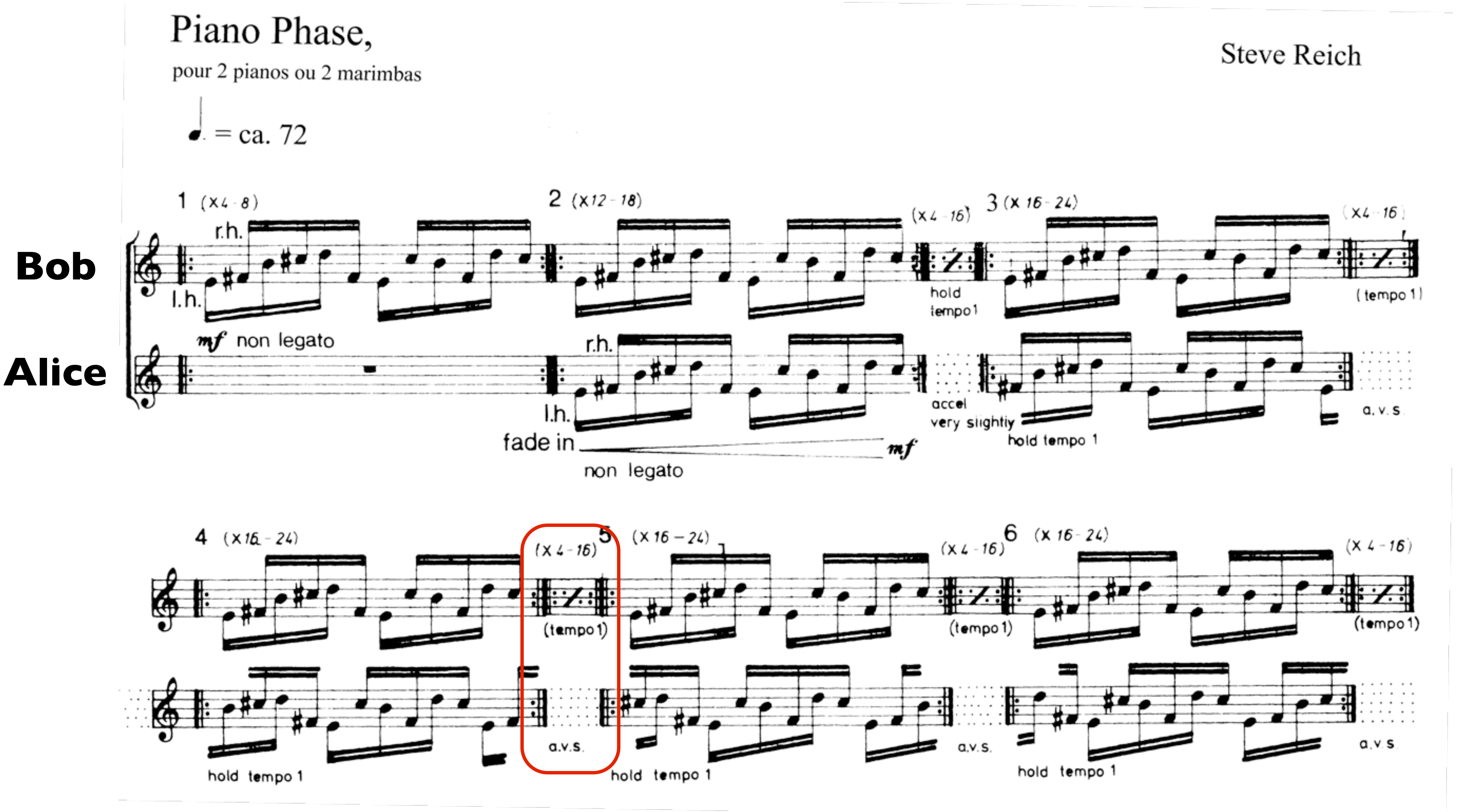
Piano Phase ...



Bob

Alice

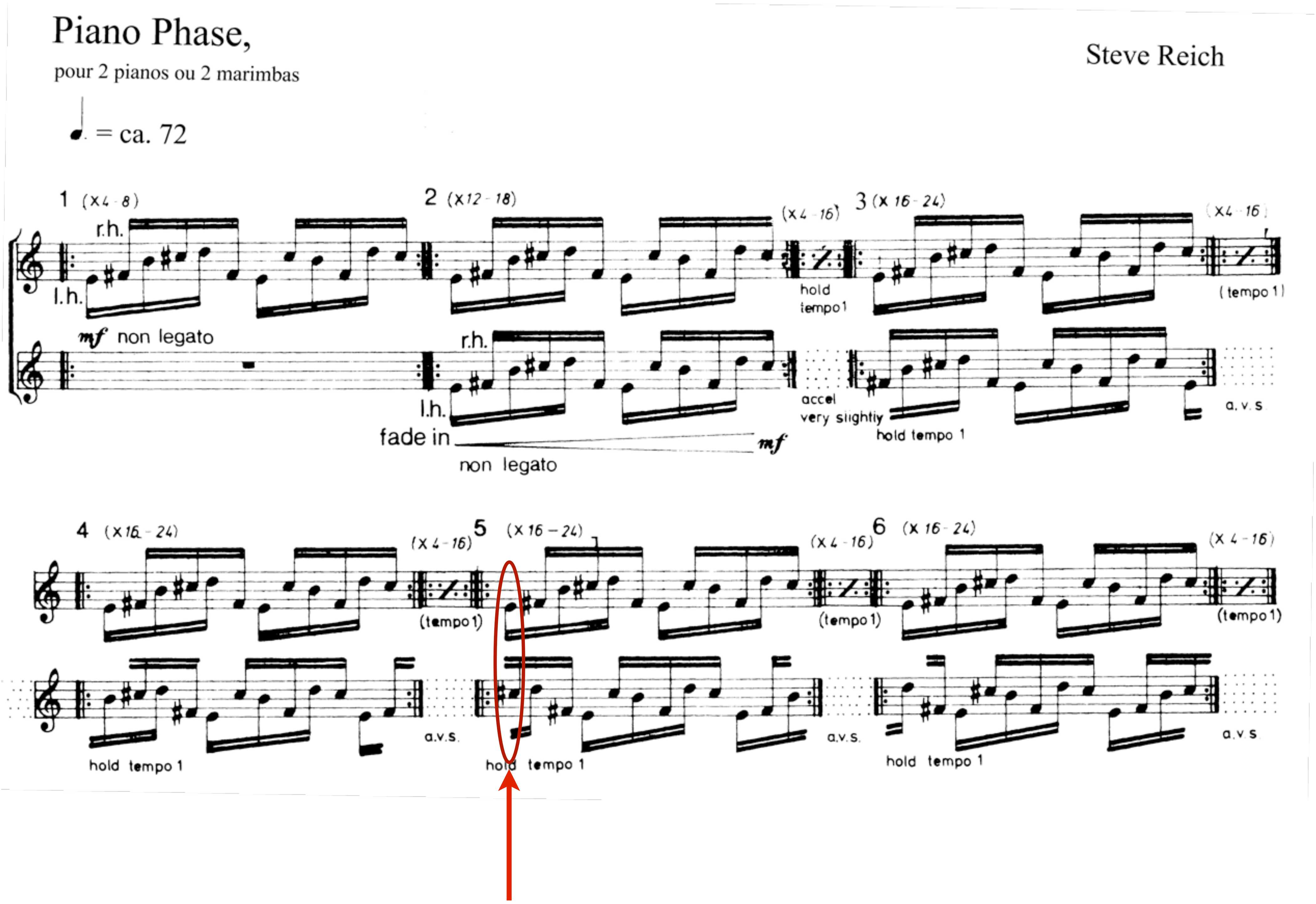
Piano Phase ...


Bob
Alice
Piano Phase,
pour 2 pianos ou 2 marimbas
Steve Reich
♩. = ca. 72
1 (x 4 - 8)
r.h.
l.h.
mf non legato
2 (x 12 - 18)
r.h.
l.h.
fade in
mf
non legato
(x 4 - 16)
hold tempo 1
accel very slightly
3 (x 16 - 24)
hold tempo 1
(x 4 - 16)
(tempo 1)
a.v.s.
4 (x 16 - 24)
hold tempo 1
(x 4 - 16)
(tempo 1)
a.v.s.
5 (x 16 - 24)
hold tempo 1
(x 4 - 16)
(tempo 1)
a.v.s.
6 (x 16 - 24)
hold tempo 1
(x 4 - 16)
(tempo 1)
a.v.s.

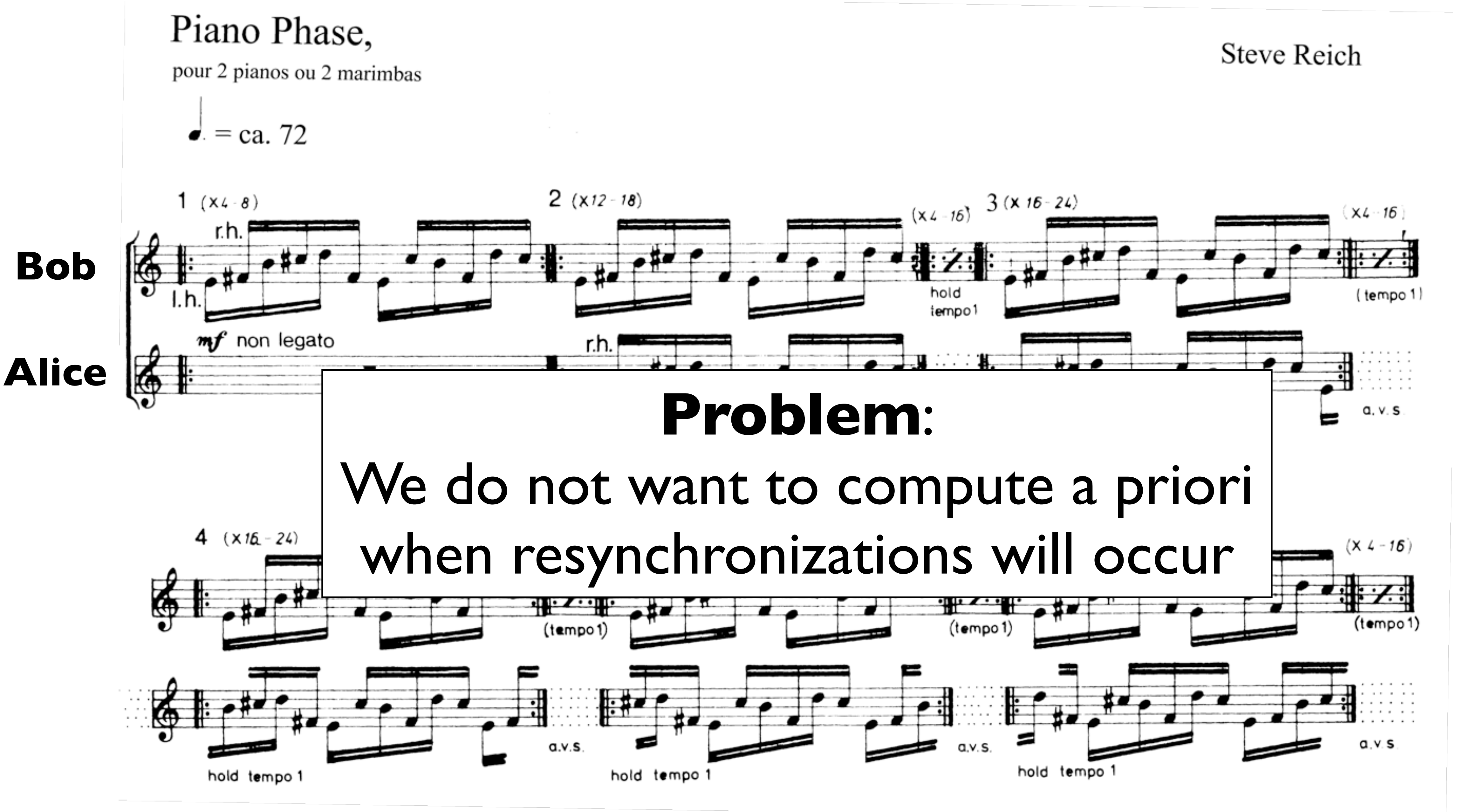
Piano Phase ...

Bob

Alice



Piano Phase ...



Bob

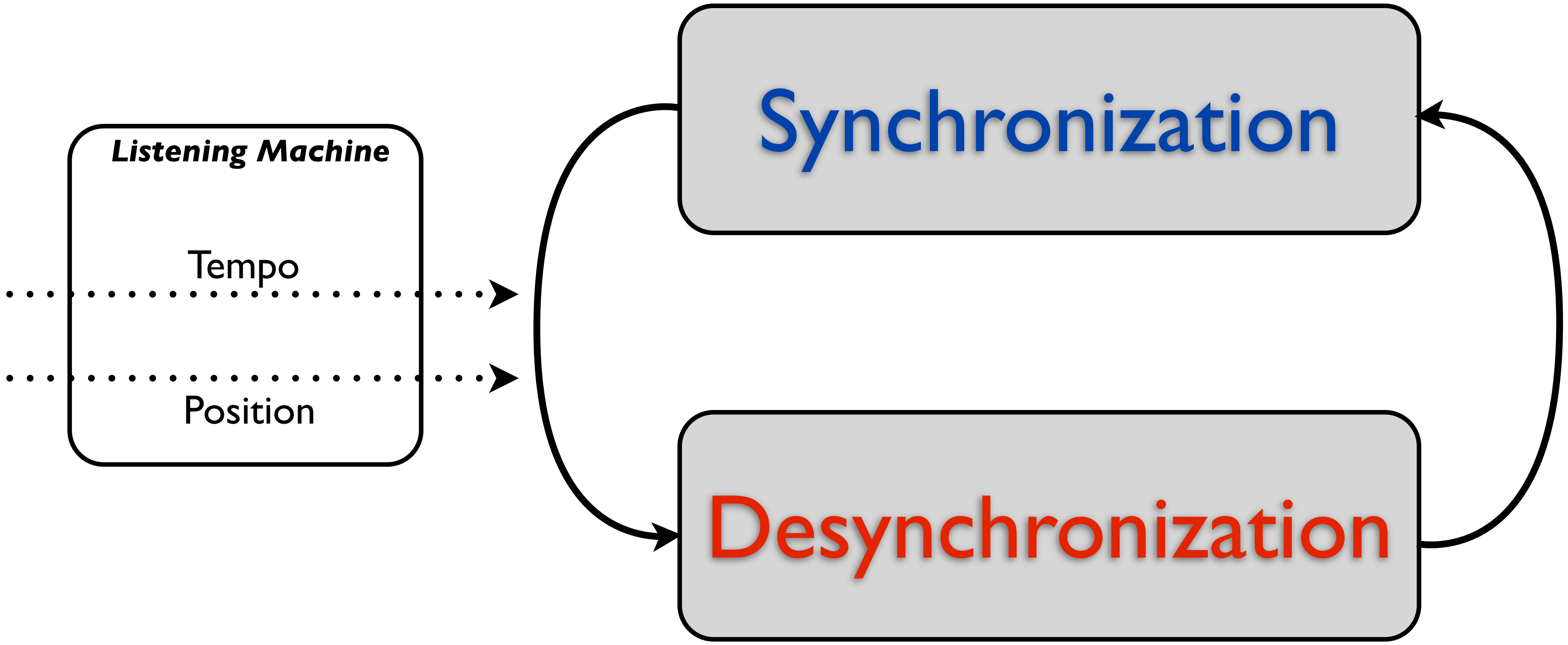
Alice

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

... in Mixed Music

Live musician
Plays the constant speed part

Electronic
Handles the desynchronization



Listening Machine

Tempo

Position

Synchronization

Desynchronization

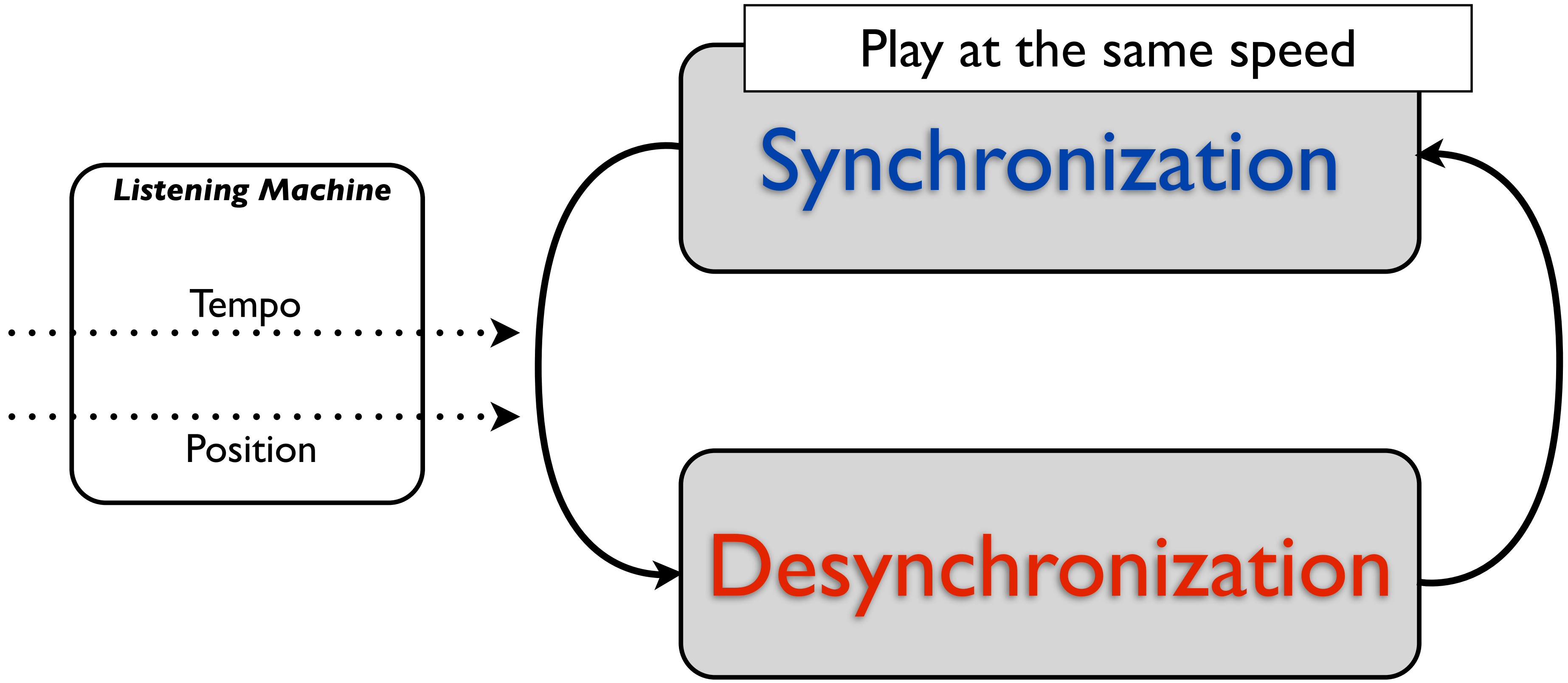
Bob

Alice

… in Mixed Music

Live musician
Plays the constant speed part

Electronic
Handles the desynchronization



Listening Machine

Tempo

Position

Play at the same speed

Synchronization

Desynchronization

Bob

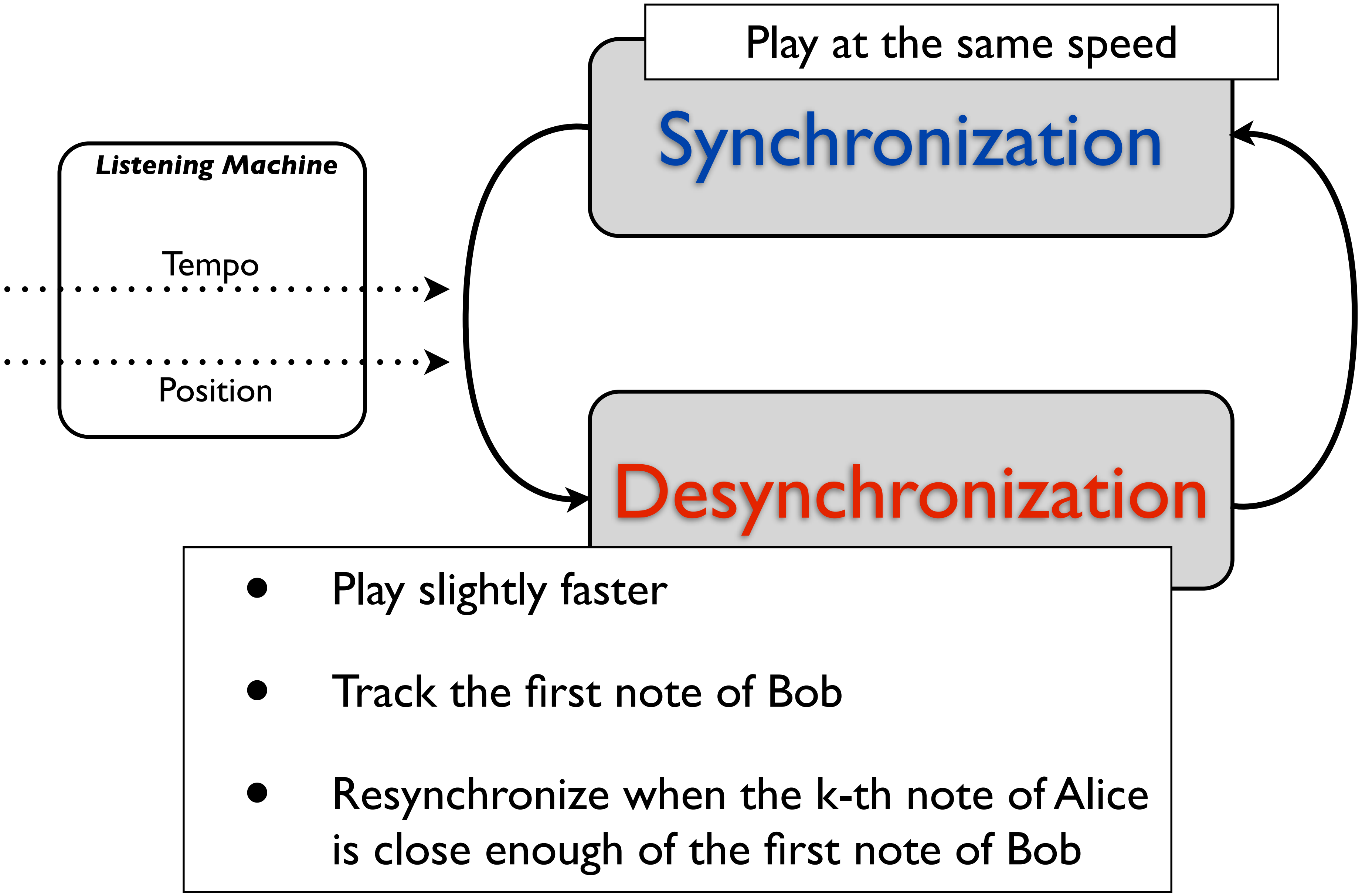
Alice

... in Mixed Music

Live musician
Plays the constant speed part

Electronic
Handles the desynchronization

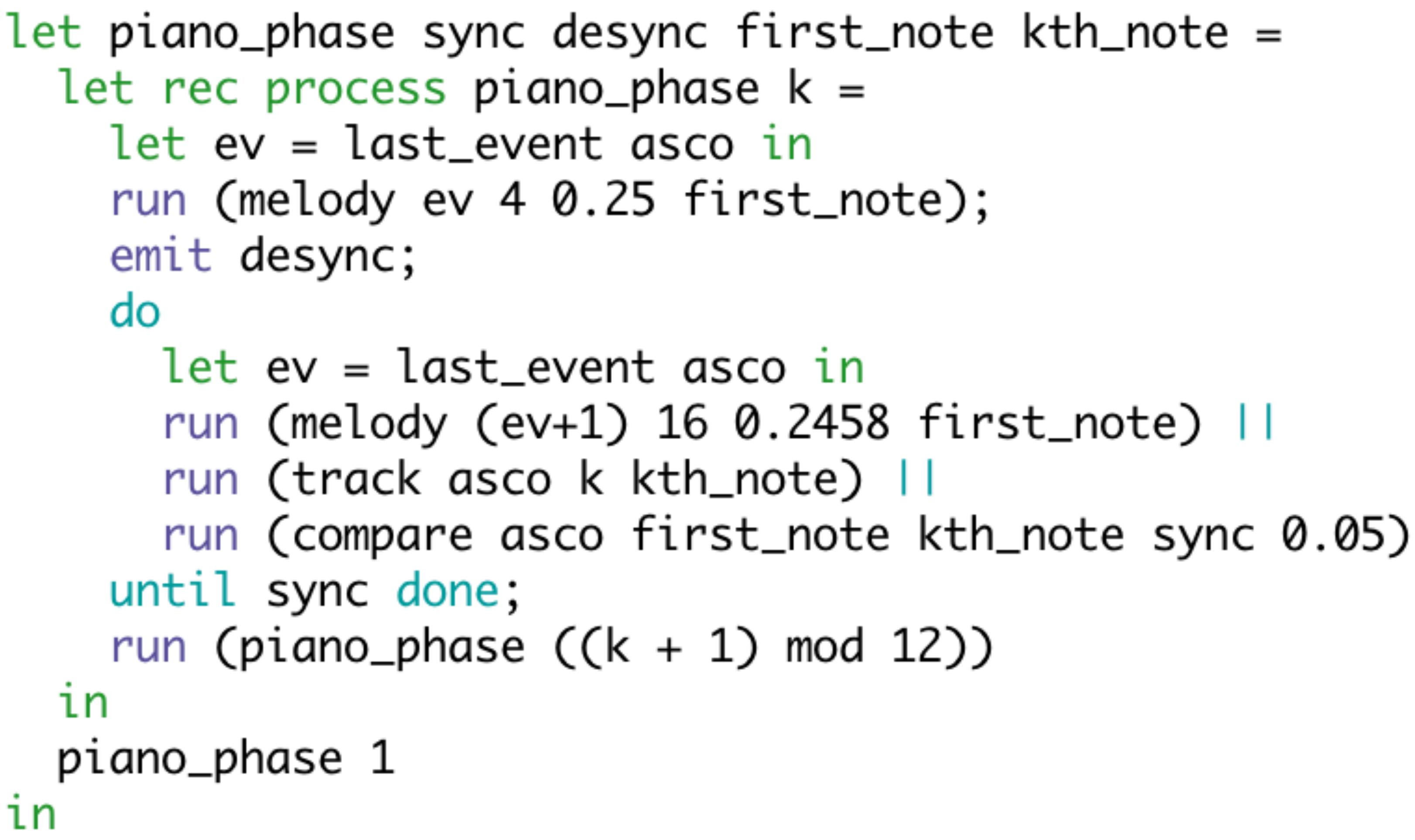
Bob



- 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



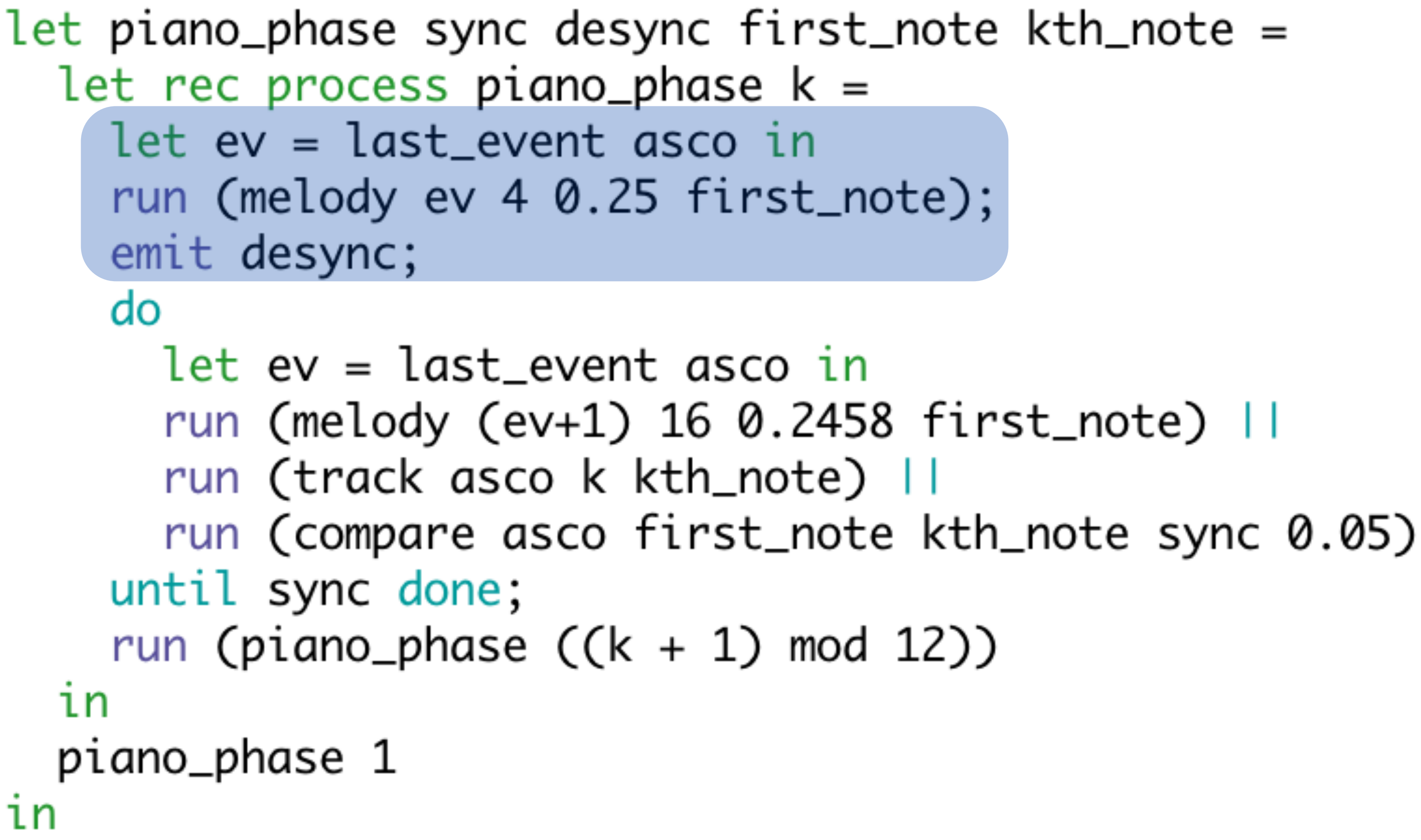
```
let piano_phase sync desync first_note kth_note =
  let rec process piano_phase k =
    let ev = last_event asco in
    run (melody ev 4 0.25 first_note);
    emit desync;
    do
      let ev = last_event asco in
      run (melody (ev+1) 16 0.2458 first_note) ||
      run (track asco k kth_note)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

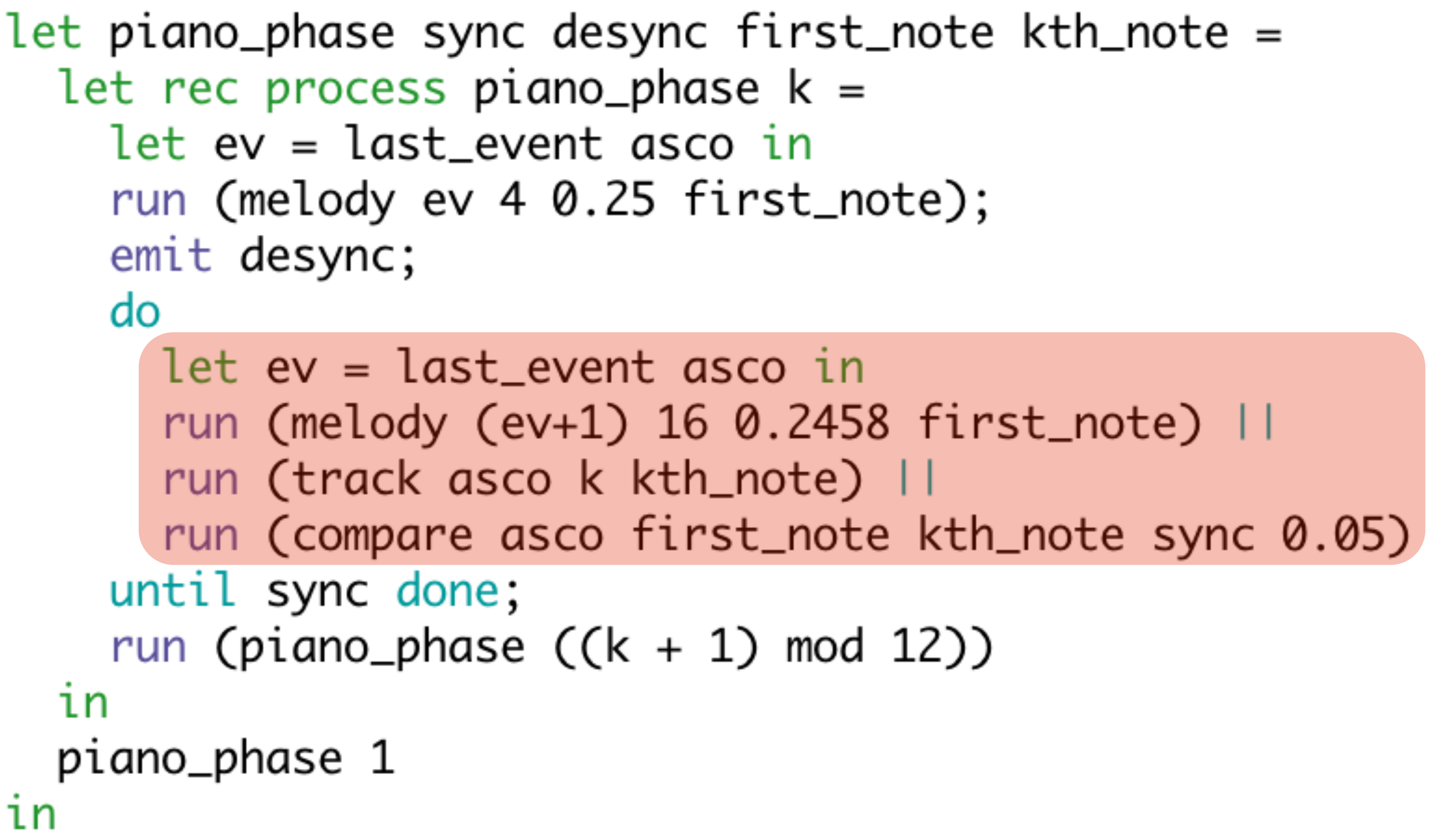
Implementation

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
```

Why ReactiveML?

- **A synchronous language**
  expressiveness for time and events
- **Functional, typed language, on top of OCaml**
  recursion and higher order processes
- **Efficient implementation**
  no busy waiting
- **Dynamical features**
  dynamical creation of processes

In Practice

- **Embedding the Antescofo language**
  new implementation of the sequencer
  using the actual antescofo listening machine
- **Extend the Antescofo language**
  functional and reactive programming
- **A tool for prototyping new features**
  reactive behaviors, live coding, new attributes
- **Link with other media**
  graphical interface, top-level, ...

To Continue...

www.reactiveml.org/farm13

References

[Mandel-Pouzet 2005] L. Mandel and M. Pouzet. *ReactiveML: a reactive extension to ML.* In Proceedings of the International Conference on Principles and Practice of Declarative Programming, 2005.

[Mandel-Plateau 2008] L. Mandel and F. Plateau. *Interactive programming of reactive systems.* In Proceedings of Model-driven High-level Programming of Embedded Systems, 2008.

[Cont 2008] A. Cont. *Antescofo: Anticipatory synchronization and control of interactive parameters in computer music.* In International Computer Music Conference, 2008.

[Echeveste et al 2012] J. Echeveste, A. Cont, J.-L. Giavitto, and F. Jacquemard. *Operational semantics of a domain specific language for real time musician-computer interaction.* Journal of Discrete Event Dynamic Systems, 2013.