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_clock = (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

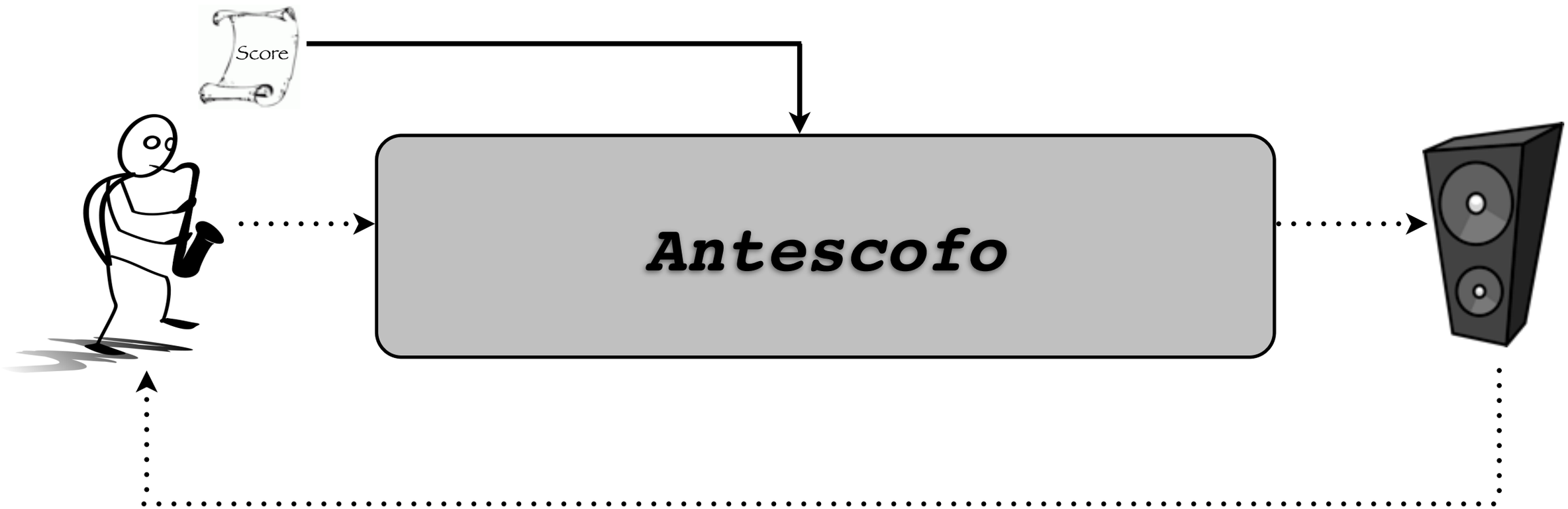
Guillaume Baudart, Florent Jacquemard, Louis Mandel, Marc Pouzet

Mixed Music and Antescofo

[Cont 2008]

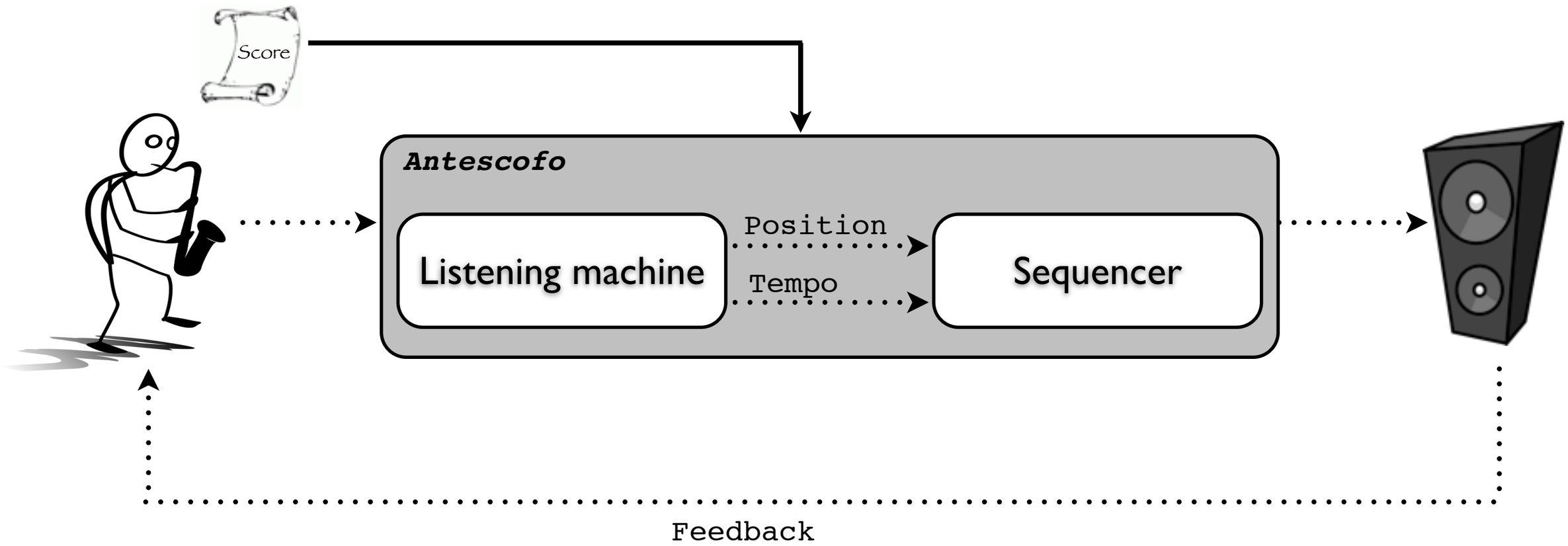
Mixed Music and Antescofo

[Cont 2008]



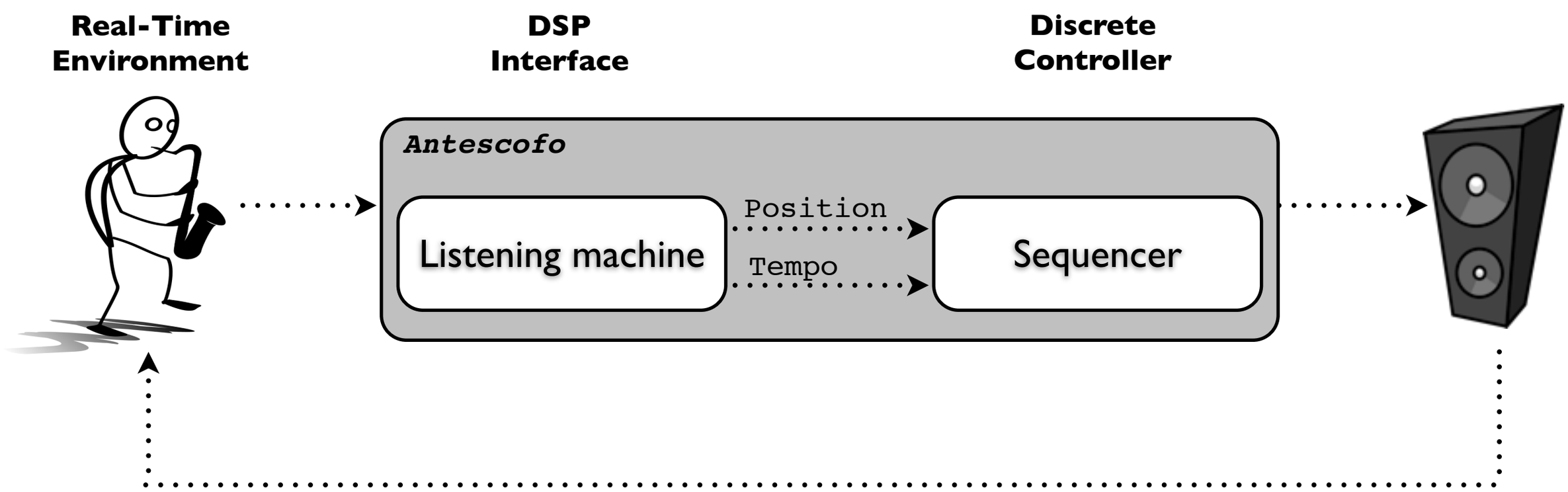
Antescofo Architecture

[Cont 2008]



Antescofo Architecture

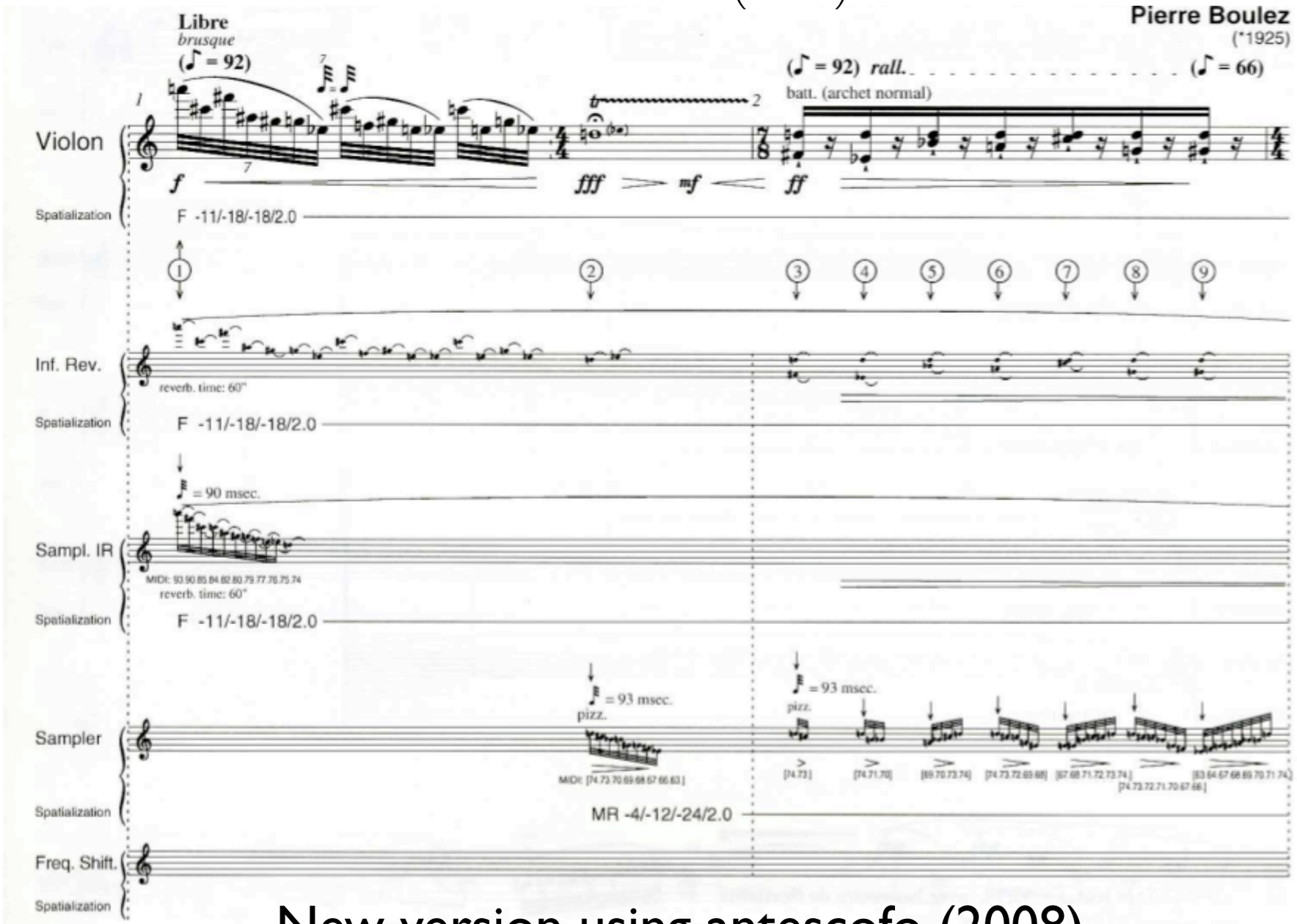
[Cont 2008]



The Antescofo Language

Goal: Jointly specify electronic and instrumental parts

Anthèmes II (1994)

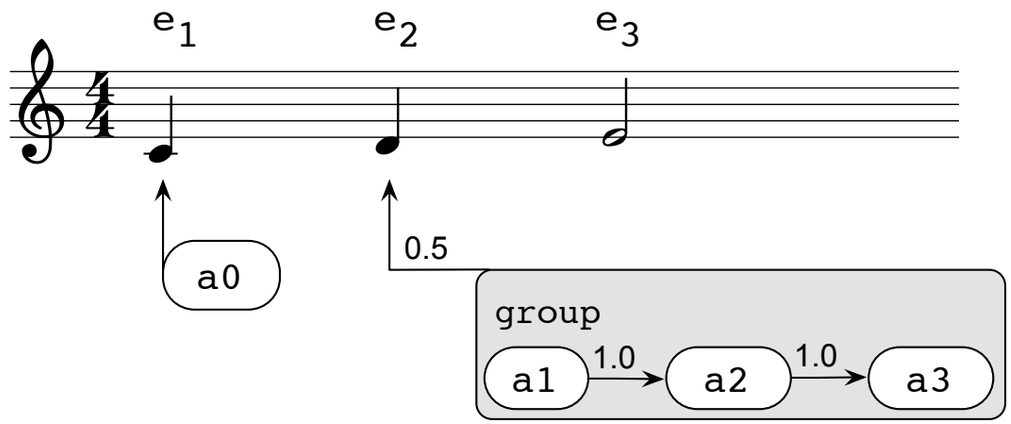


New version using antescofo (2008)

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
```

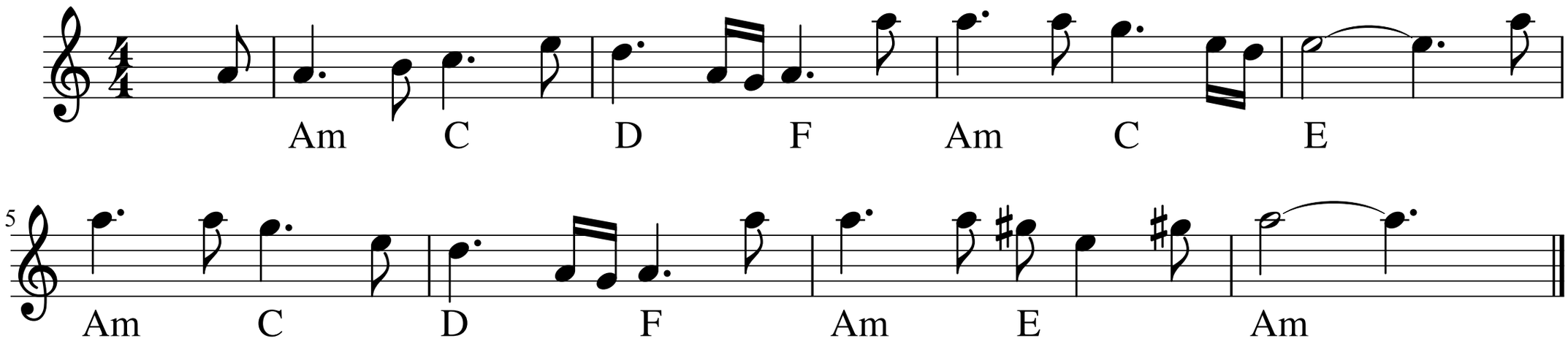
- Time is relative to the tempo
- Electronic actions are characterized by a delay
- Hierarchical structure: *groups and nested groups*
- Synchronization with the musician : *tight, loose*
- Error handling strategies : *partial, causal*

Live Coding

Modify, correct and interact with the score during the performance

Automatic Accompaniment

The house of the rising sun



- **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

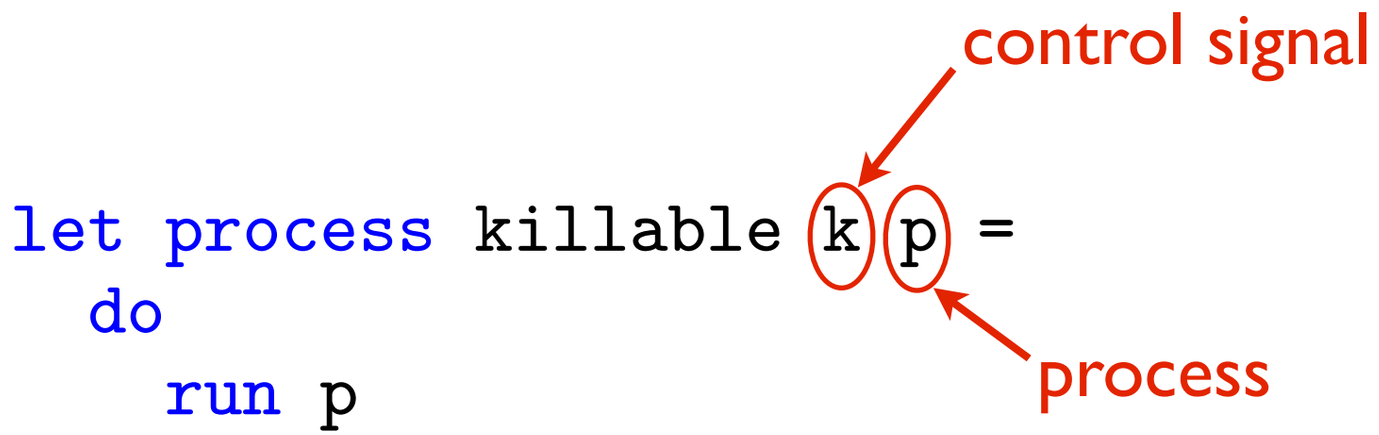
Example of a higher-order process

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

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

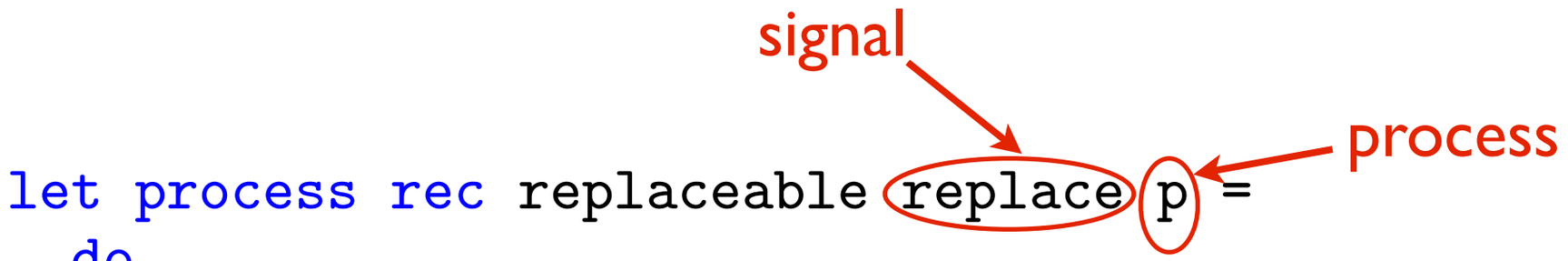
Example of a recursive higher-order process

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 =
  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

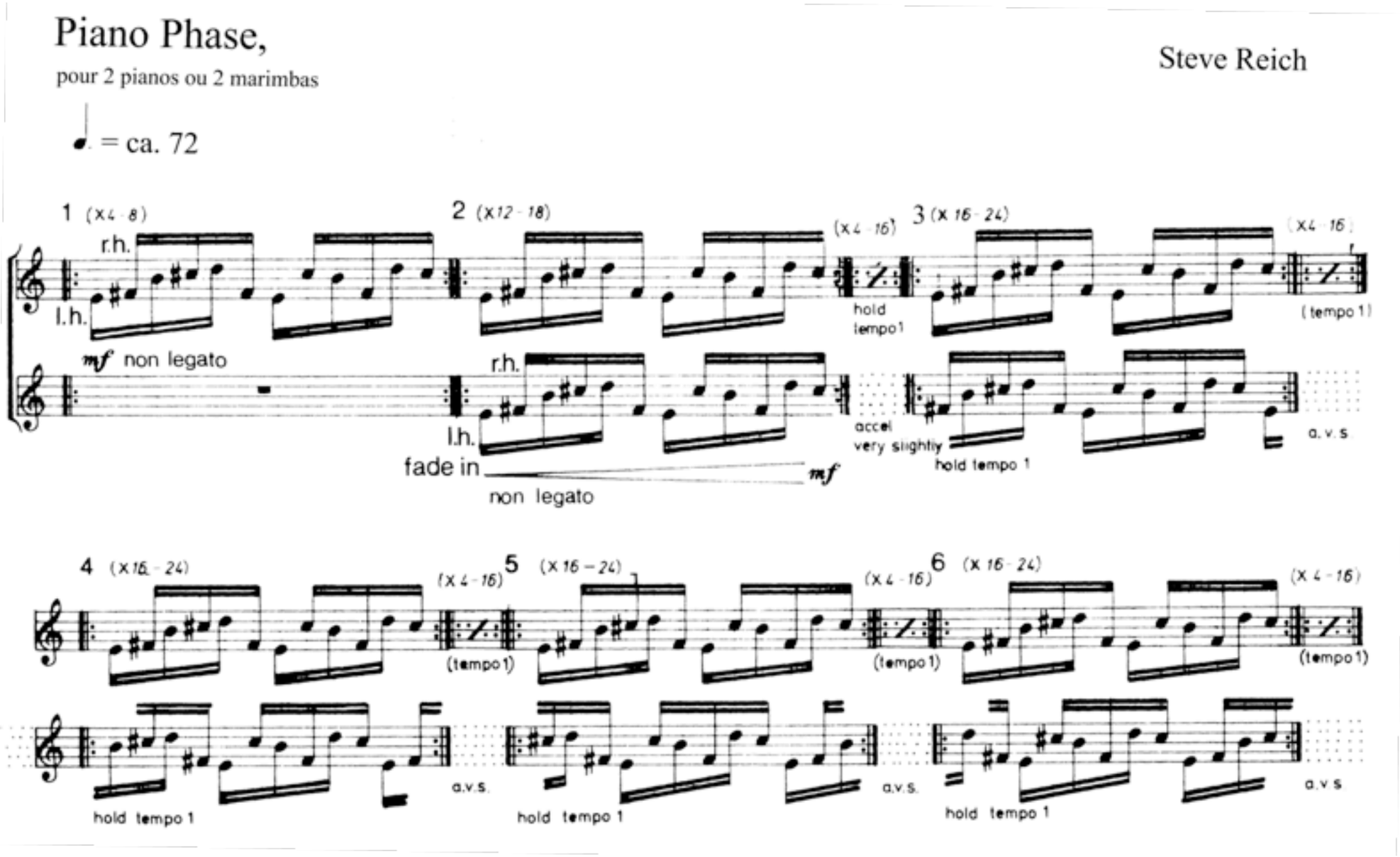
New Reactive Behaviors

Example: Steve Reich's Piano Phase

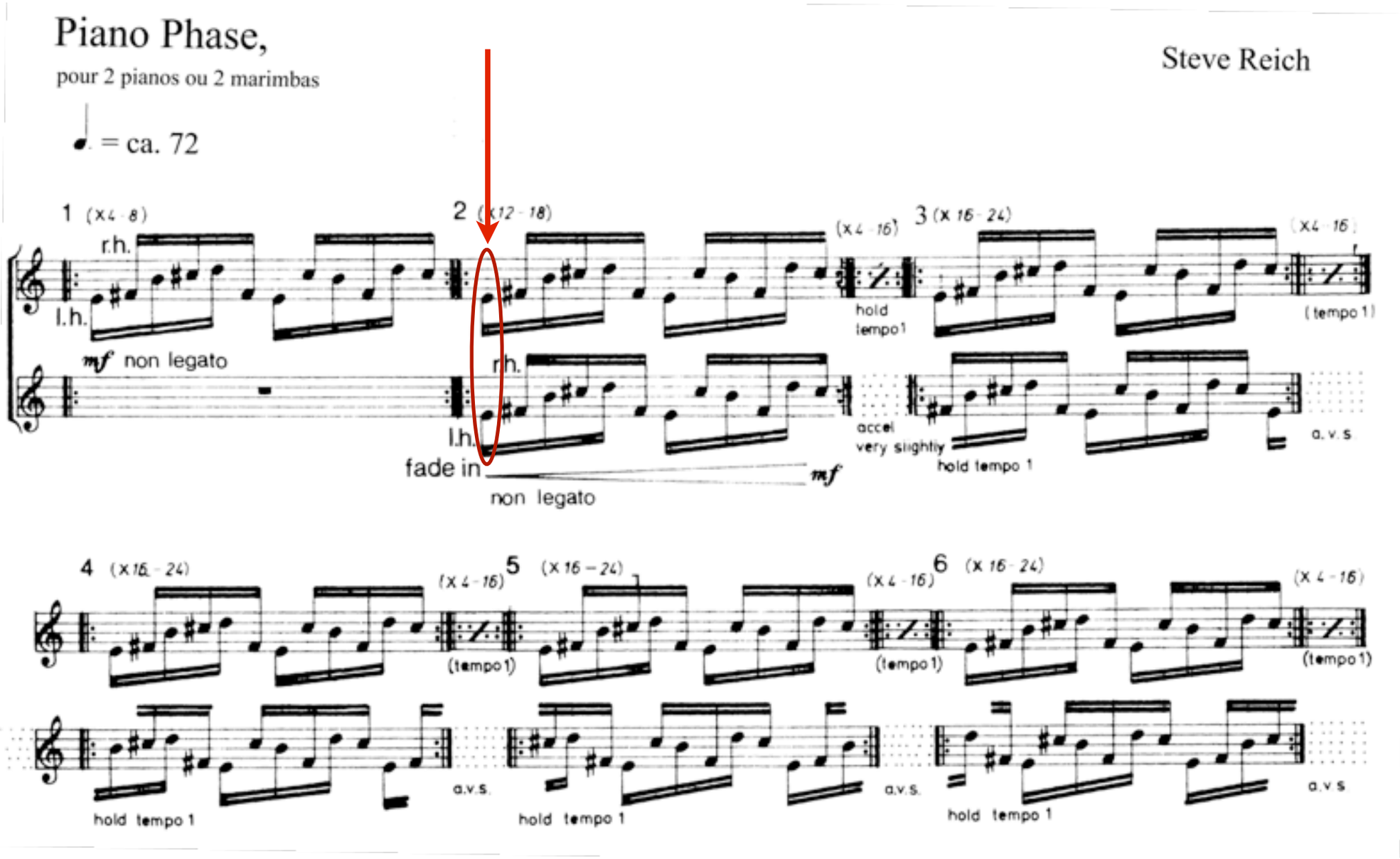
Piano Phase ...

Bob

Alice



Piano Phase ...

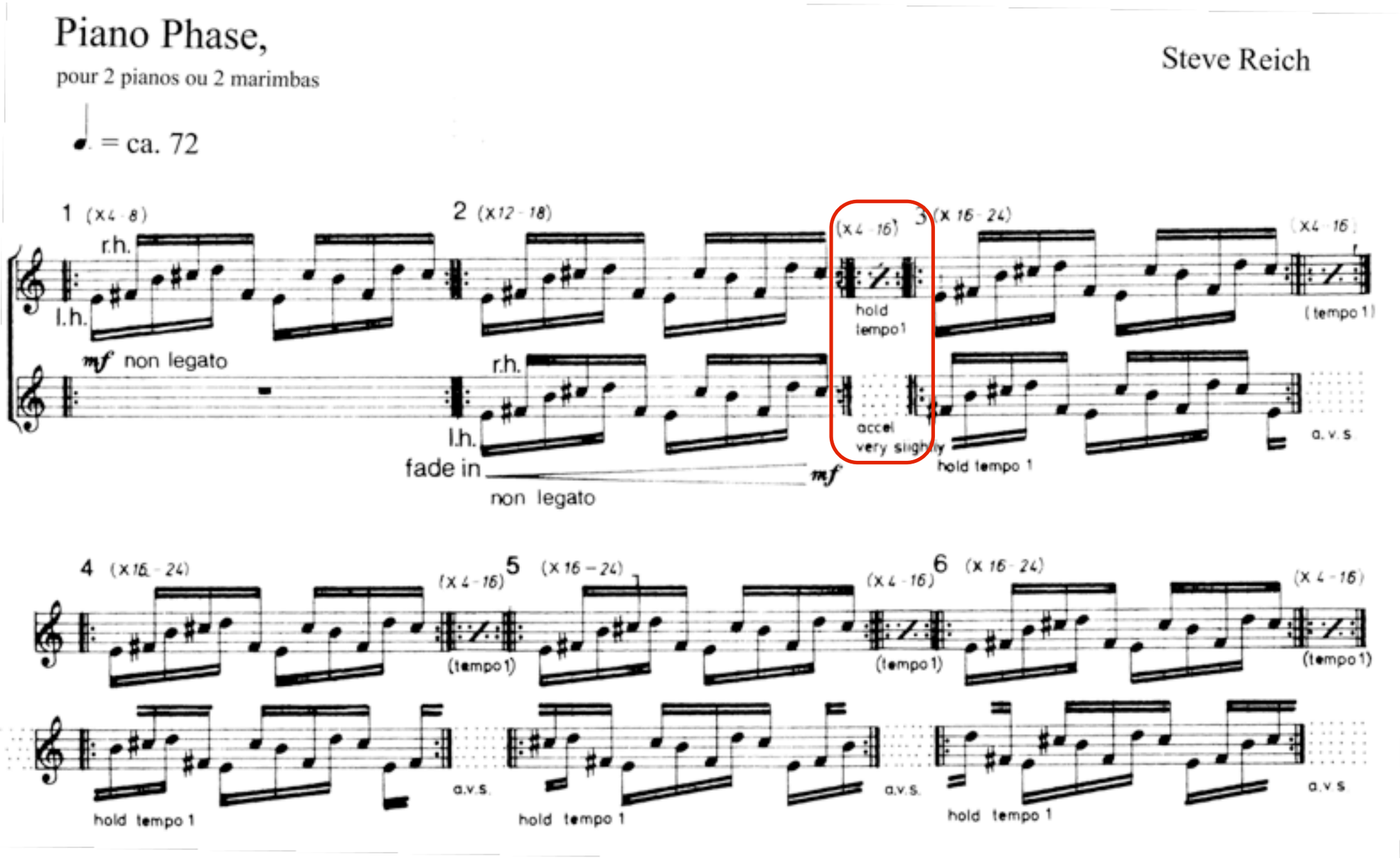


Bob

Alice

Synchronization

Piano Phase ...



Bob

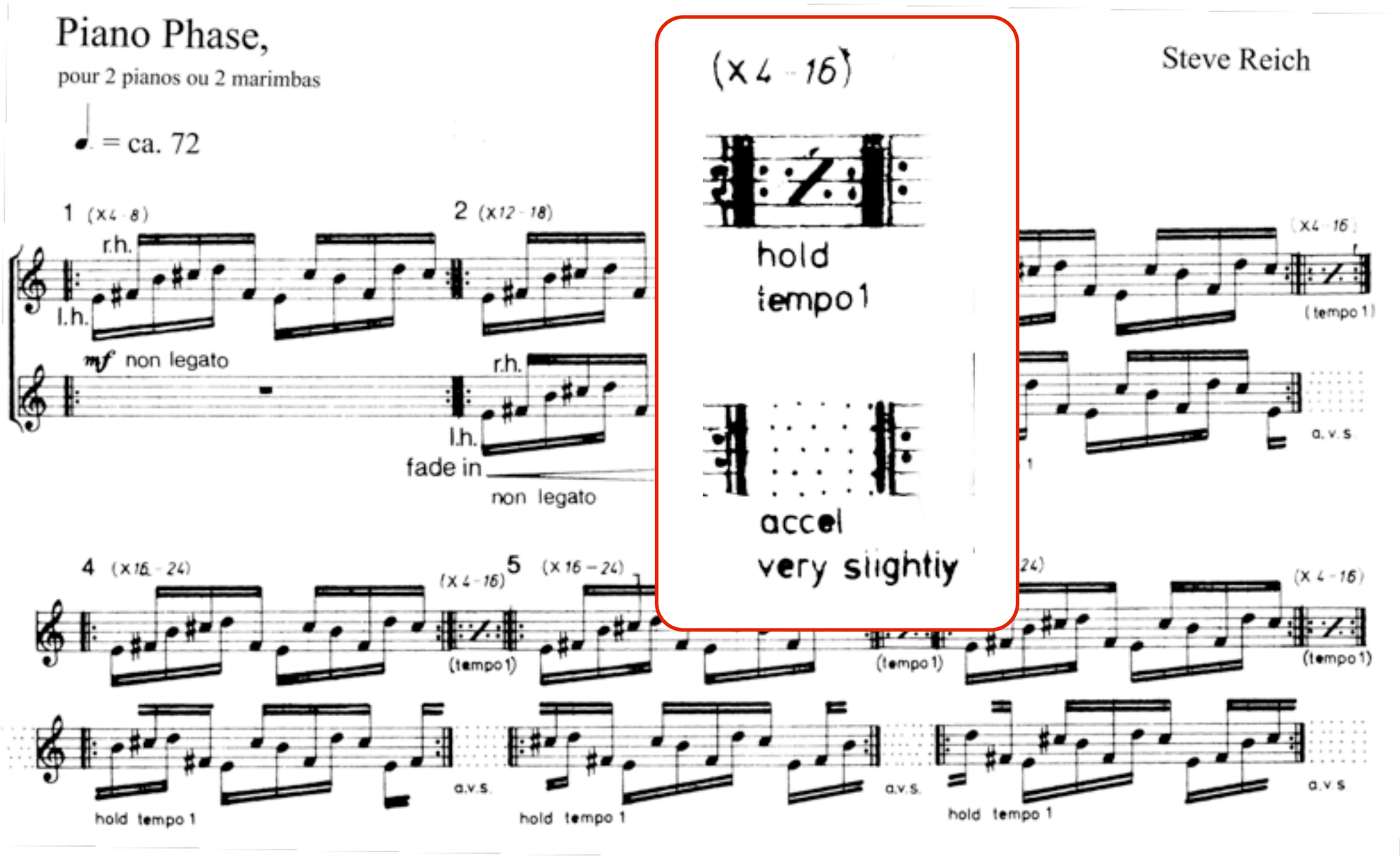
Alice

Desynchronization

Piano Phase ...

Bob

Alice

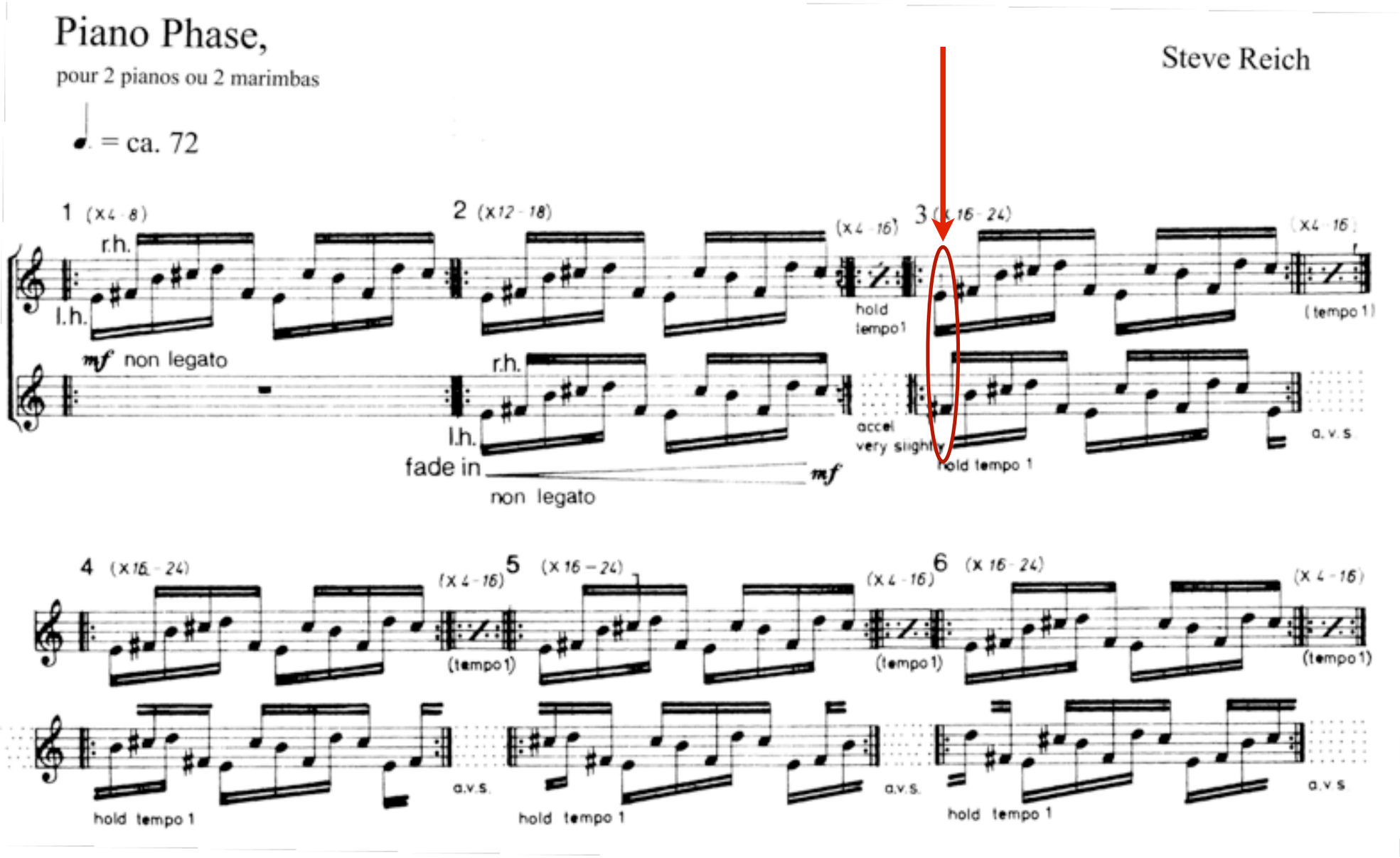


Desynchronization

Piano Phase ...

Bob

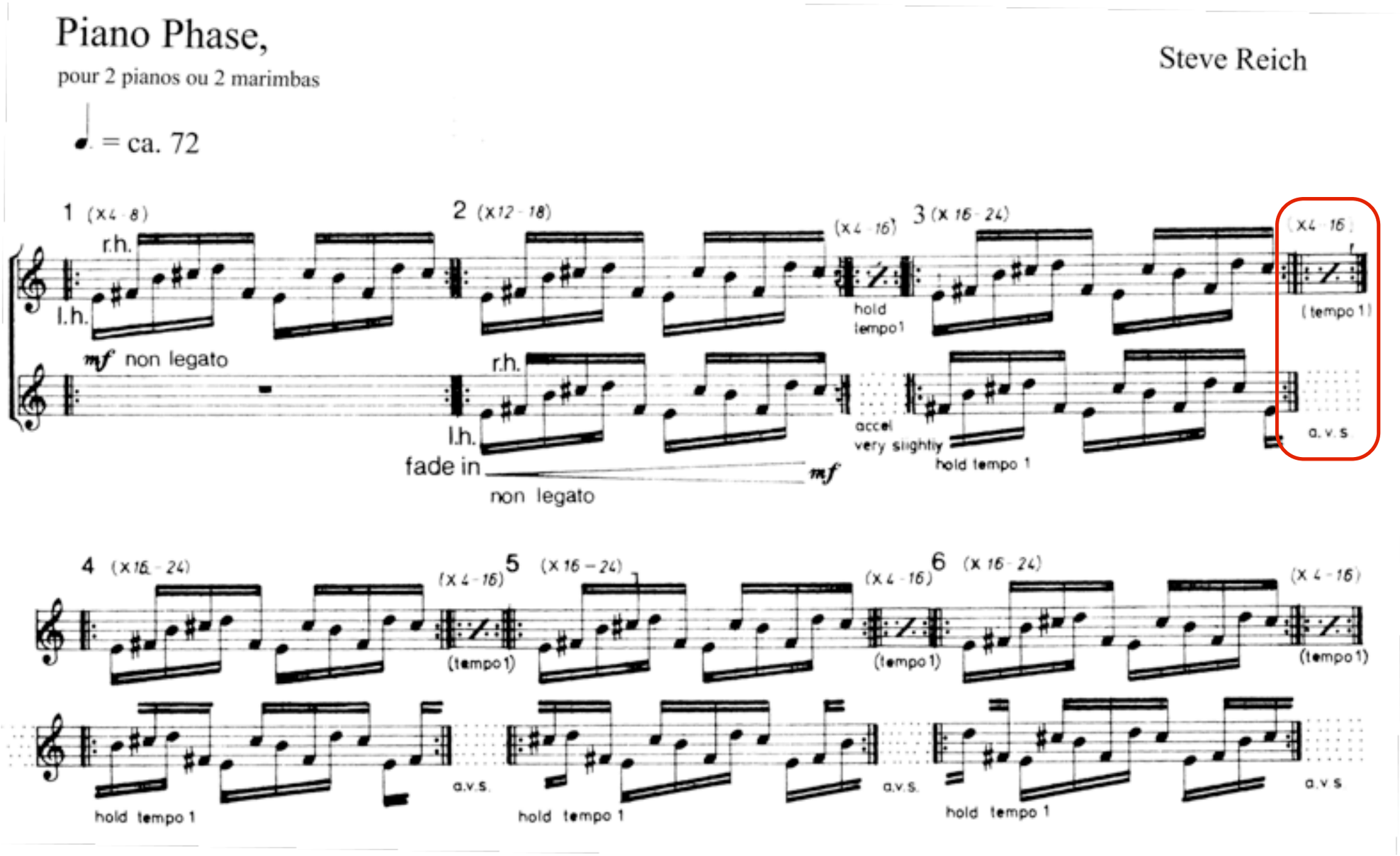
Alice



Piano Phase ...

Bob

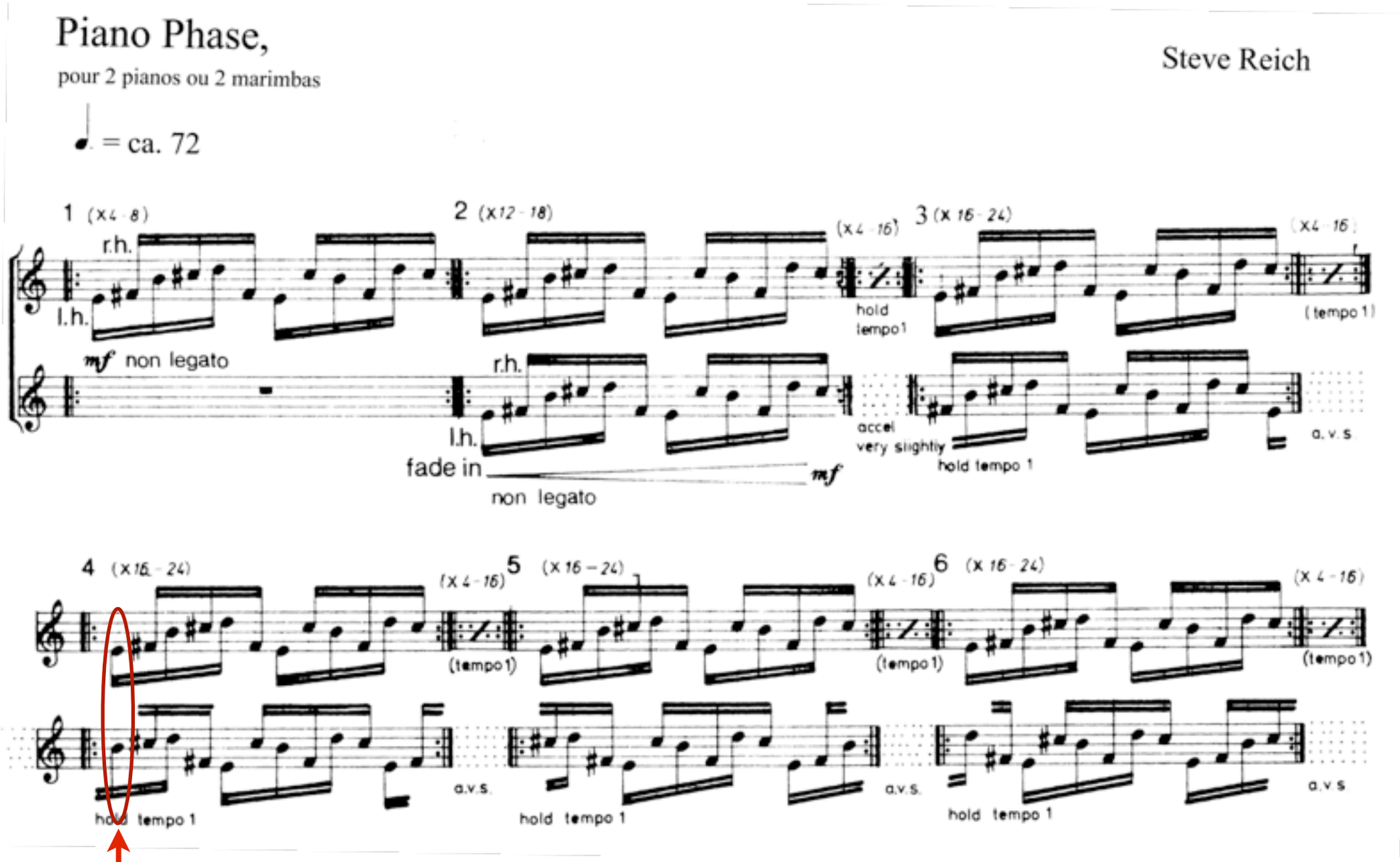
Alice



Piano Phase ...

Bob

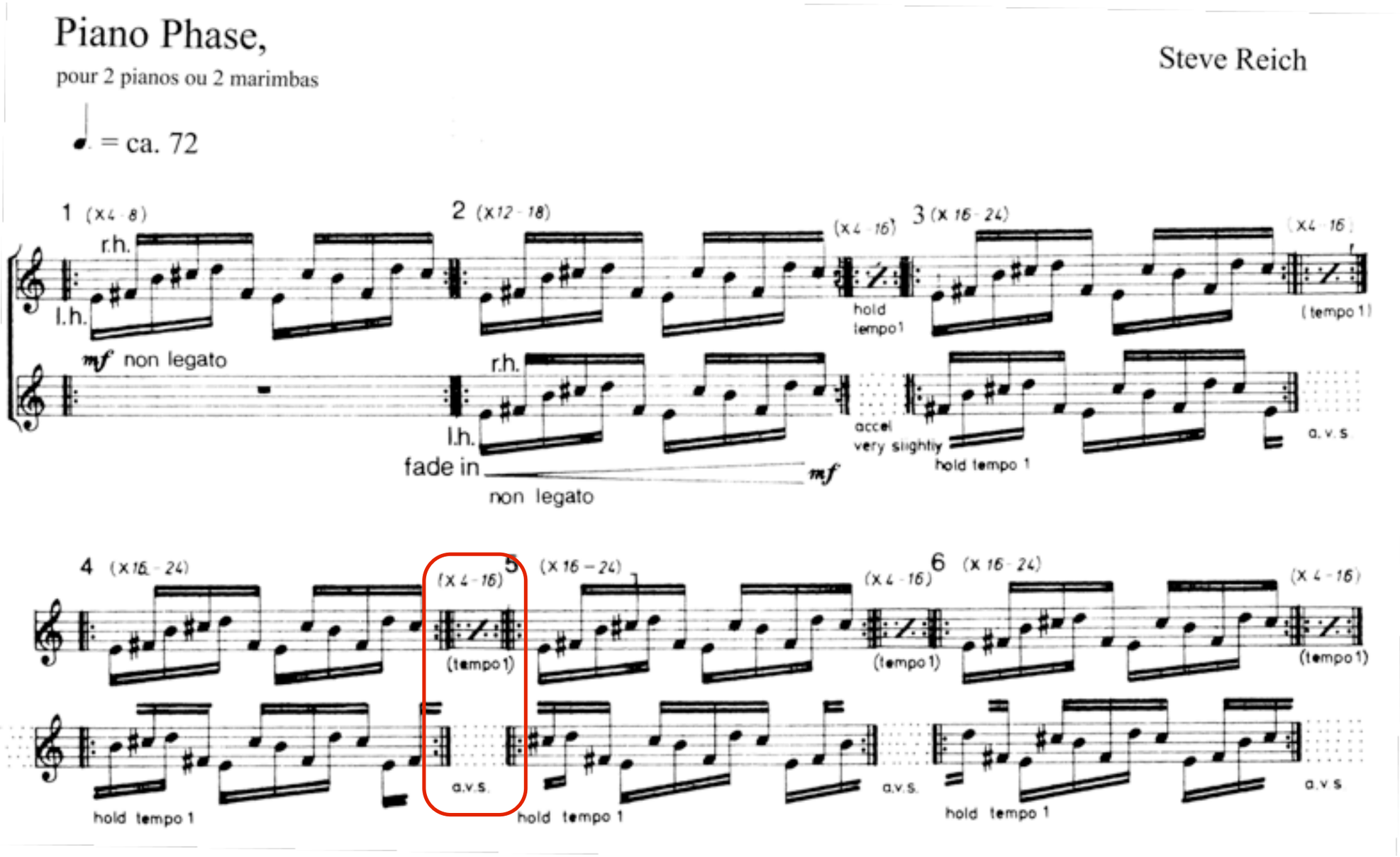
Alice



Piano Phase ...

Bob

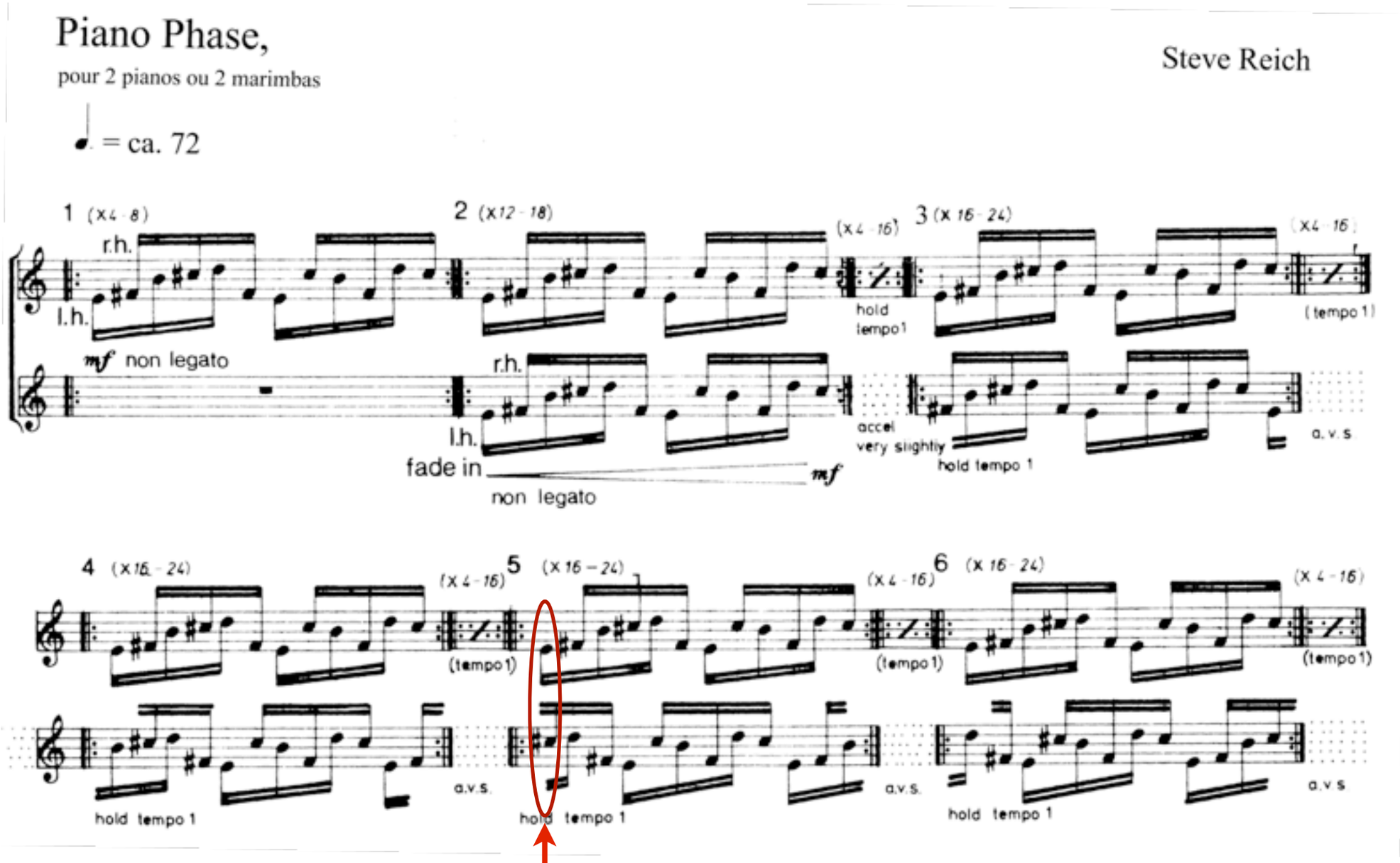
Alice



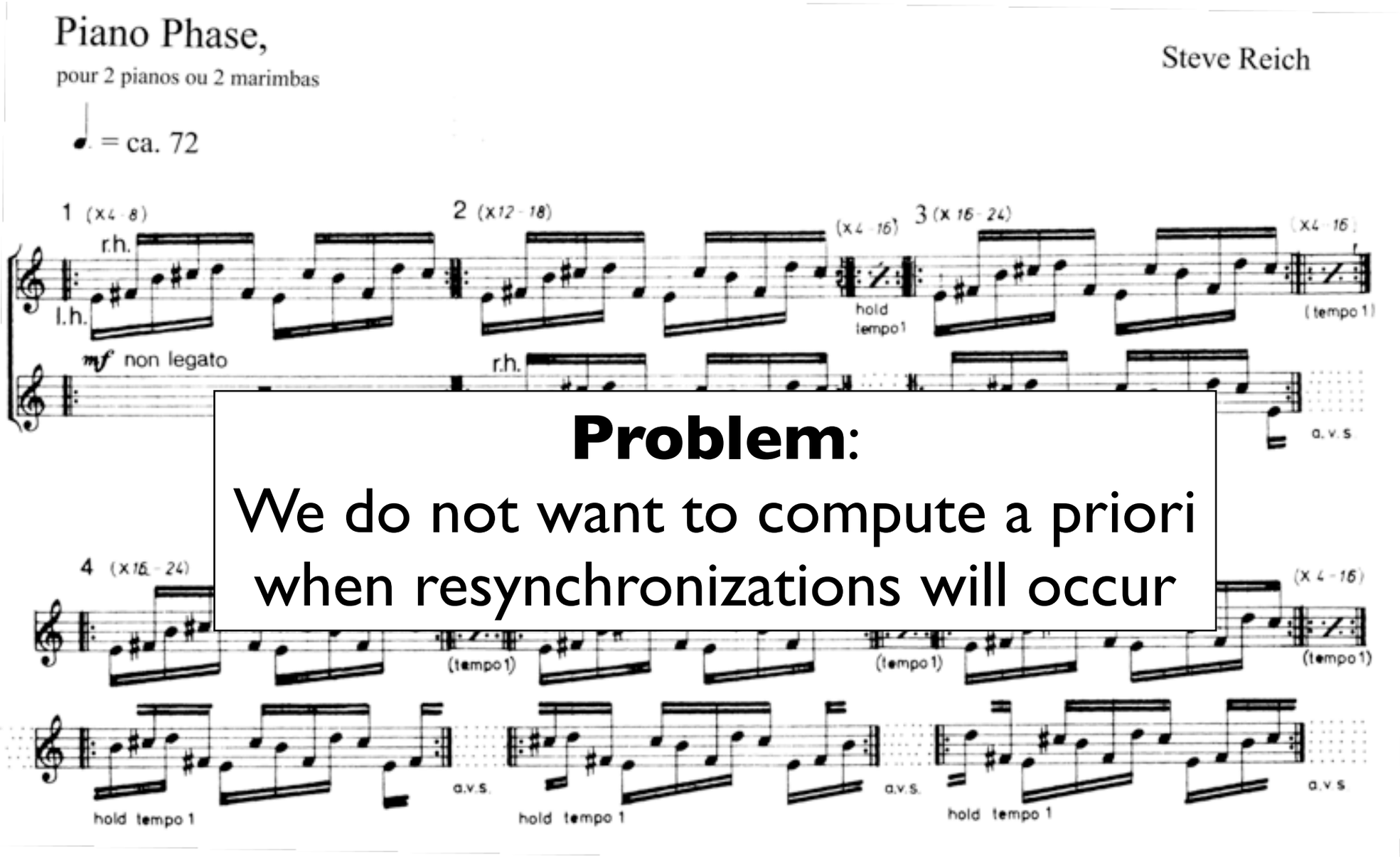
Piano Phase ...

Bob

Alice



Piano Phase ...



Bob

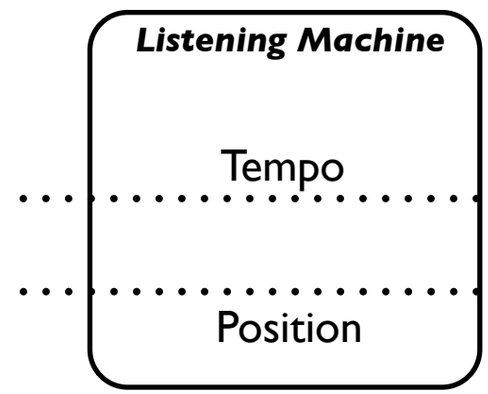
Alice

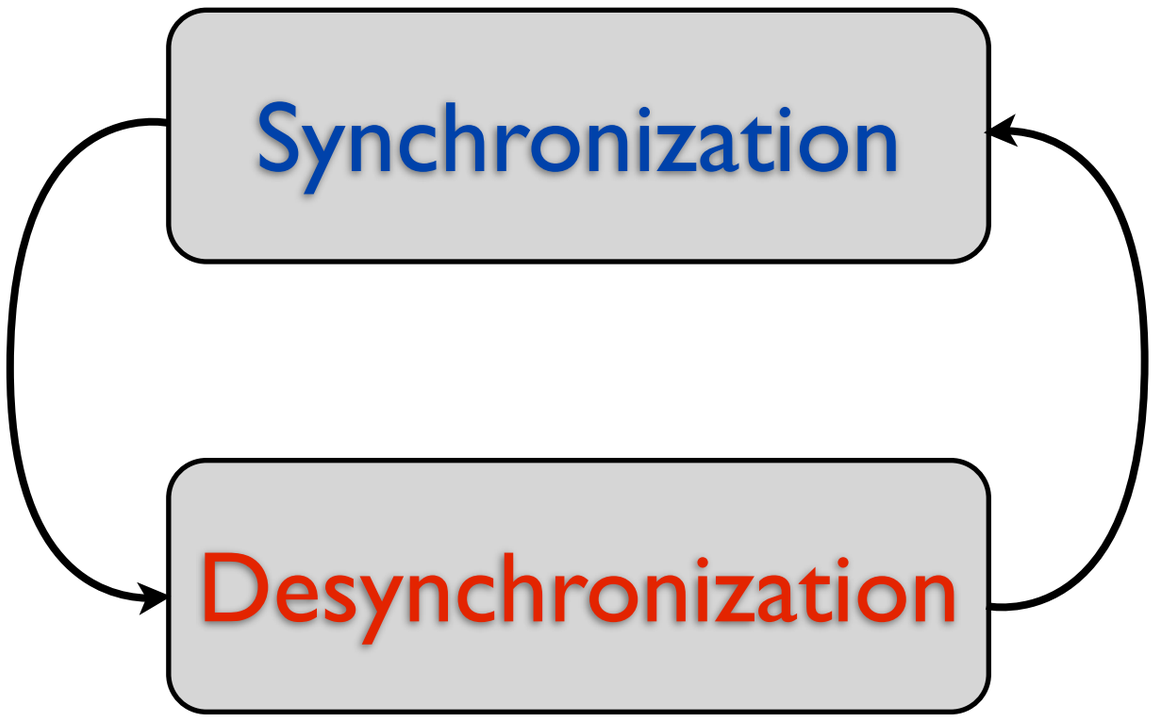
Problem:
We do not want to compute a priori when resynchronizations will occur

... in Mixed Music

Live musician
Plays the constant speed part

Electronic
Handles the desynchronization





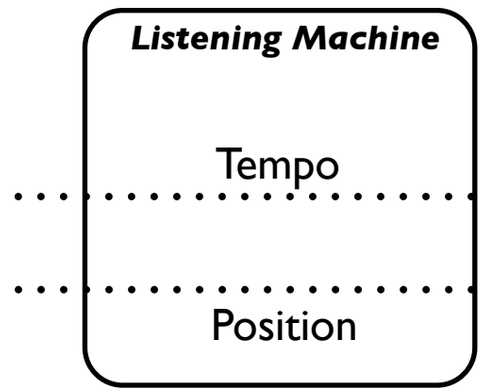
Bob

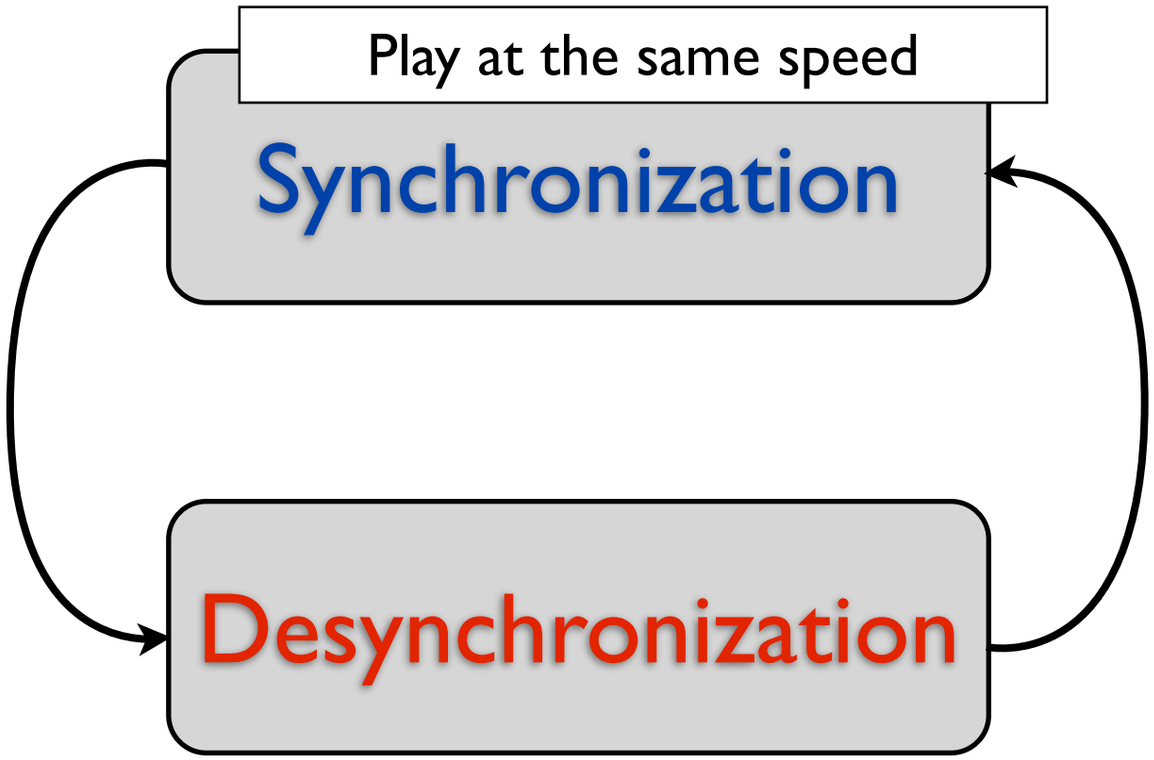
Alice

... in Mixed Music

Live musician
Plays the constant speed part

Electronic
Handles the desynchronization





Bob

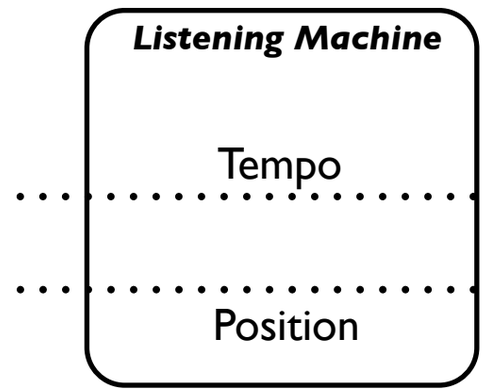
Alice

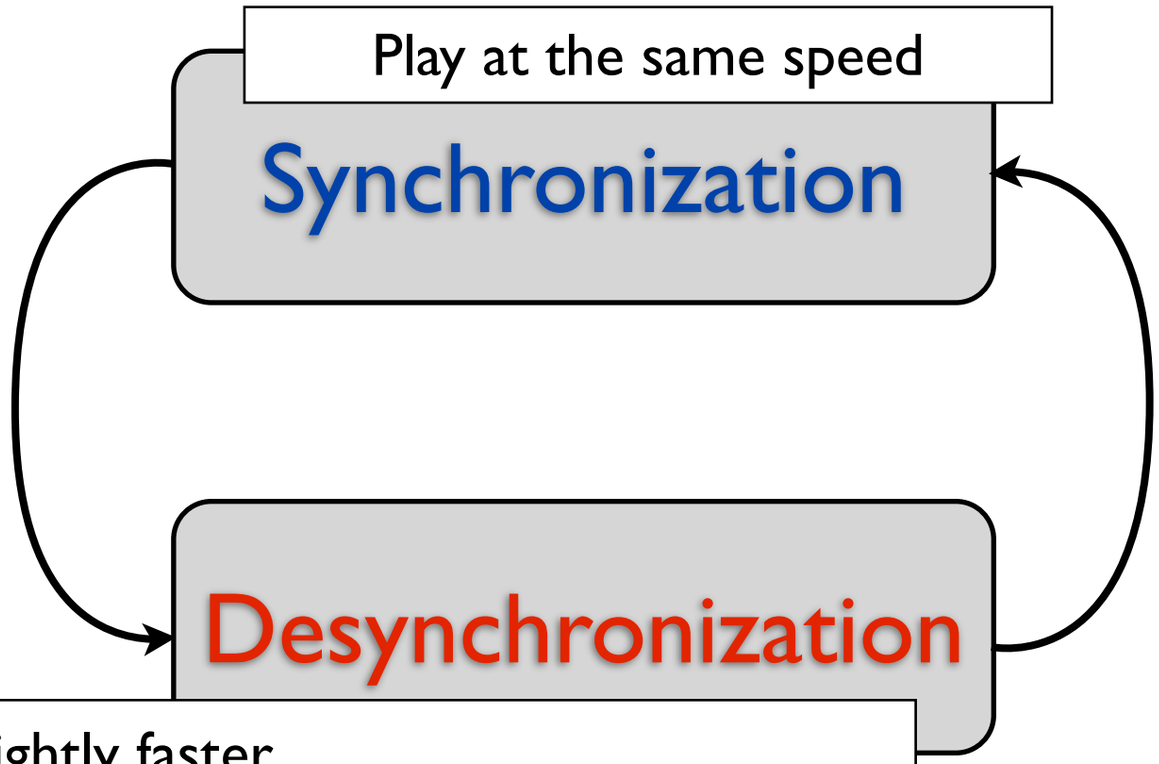
... in Mixed Music

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
```

Retour sur les choix

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

encore une démo

http://reactiveml.org