The Antescofo Language in ReactiveML

Guillaume Baudart, ENS
Florent Jacquemard, IRCAM
Louis Mandel, Collège de France
Marc Pouzet, ENS

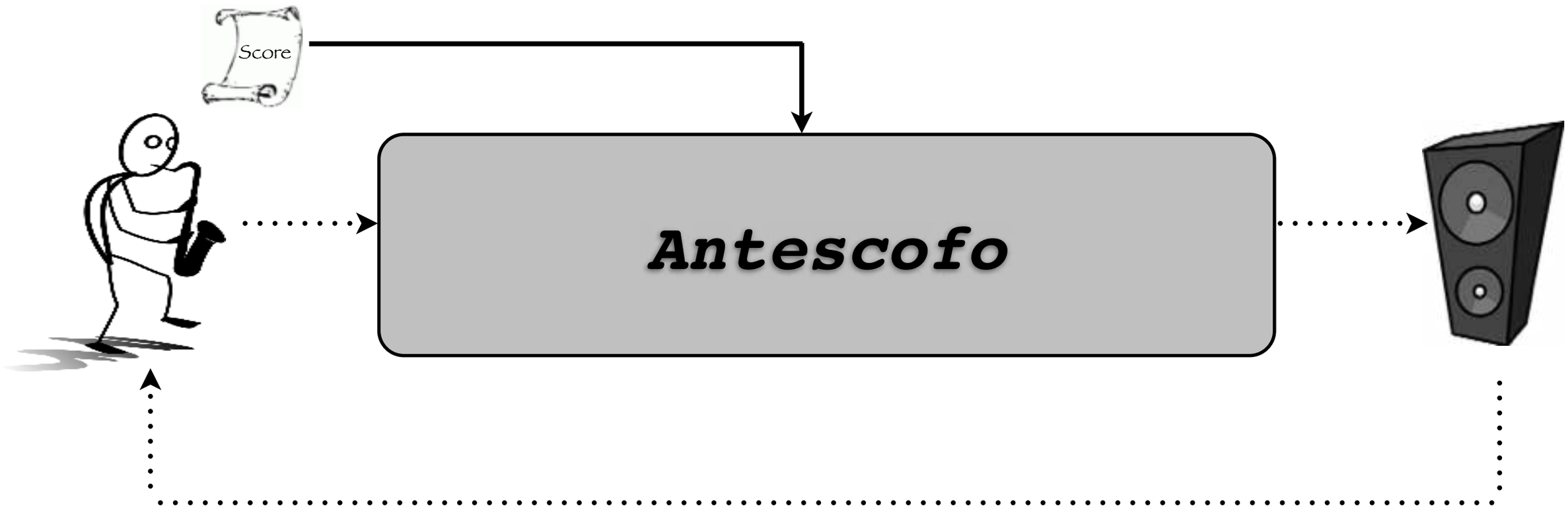
Collège de France **12.18.2013**

Mixed Music and Antescofo

[Cont 2008]

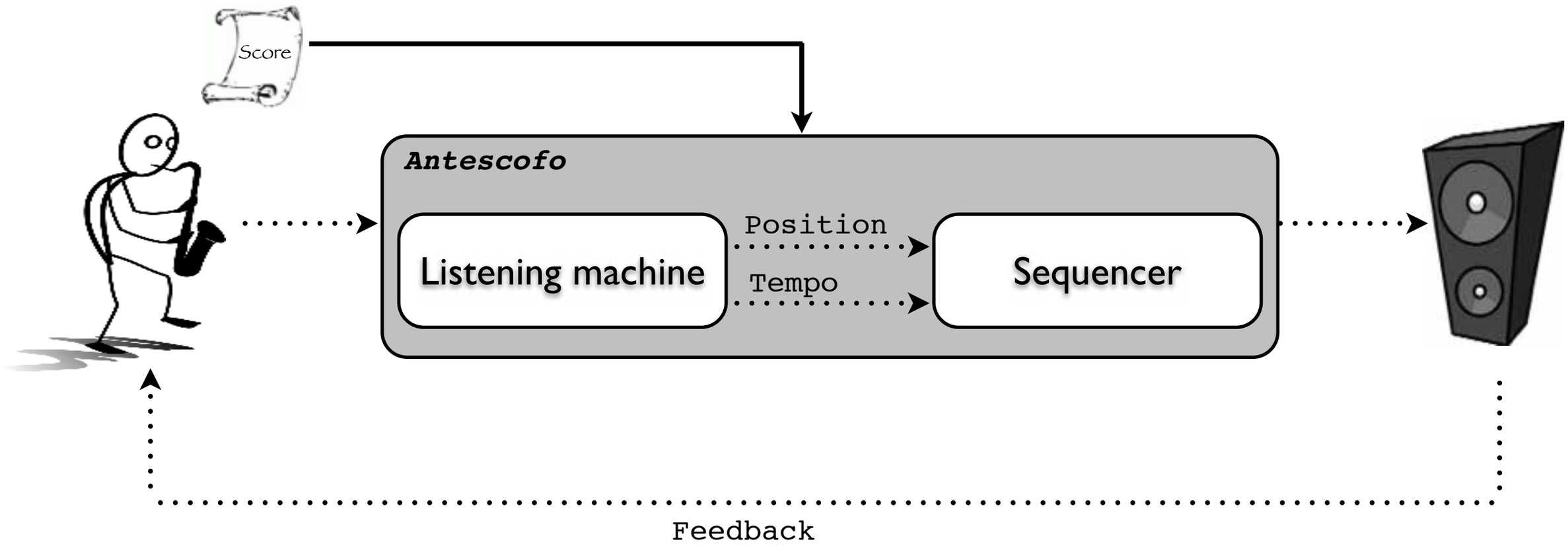
Mixed Music and Antescofo

[Cont 2008]



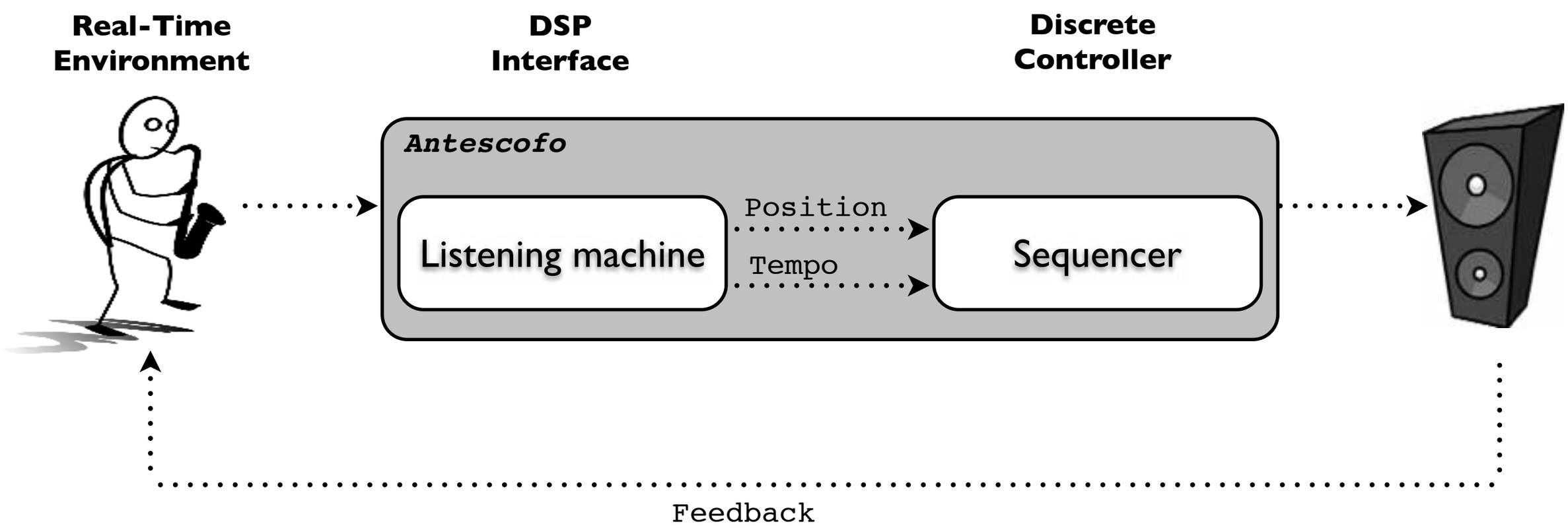
Antescofo Architecture

[Cont 2008]



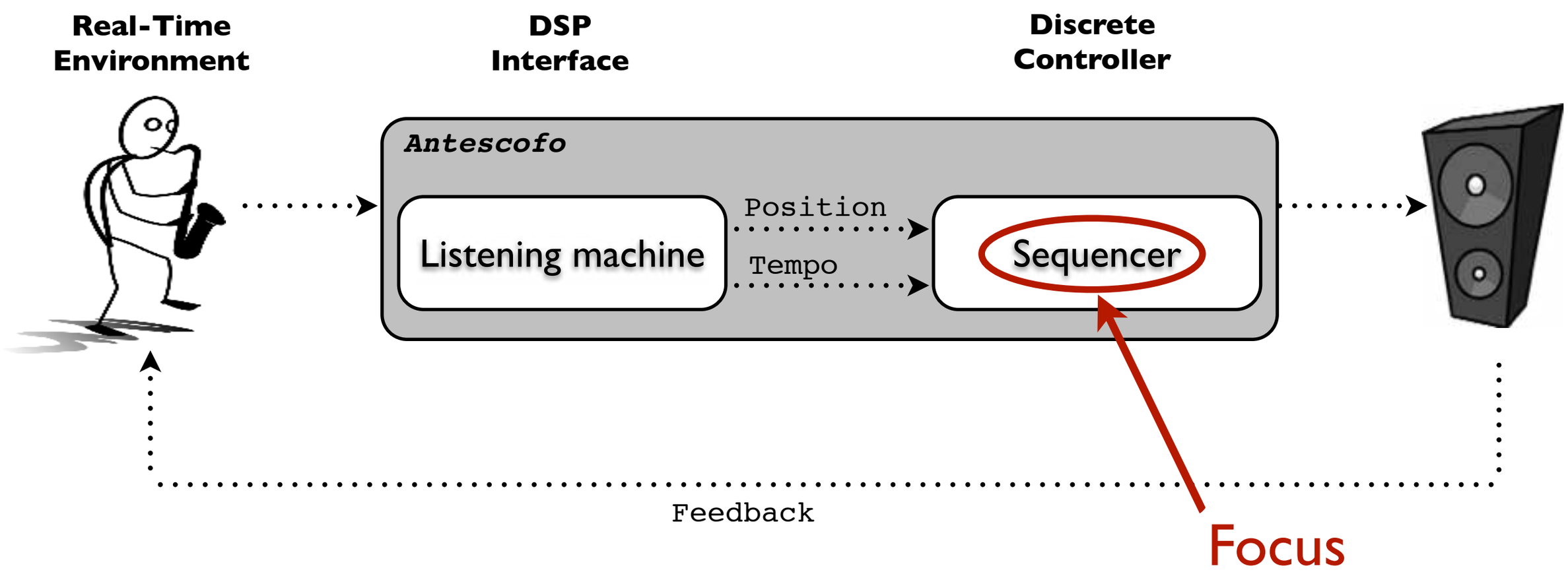
Antescofo Architecture

[Cont 2008]



Antescofo Architecture

[Cont 2008]



Motivations

- **Link with the synchronous model**
  - An executable semantics for Antescofo
  - Embedding in a synchronous reactive language
- **Benefits**
  - Live coding
  - Prototyping new features:
    new attributes, reactive behaviors, ...

I. The Antescofo Language

- Description
- Synchronization and error handling strategies

II. Semantics

- Formalization
- The three predicates

III. Implementation Architecture

- Architecture
- Embedding in ReactiveML

IV. Applications

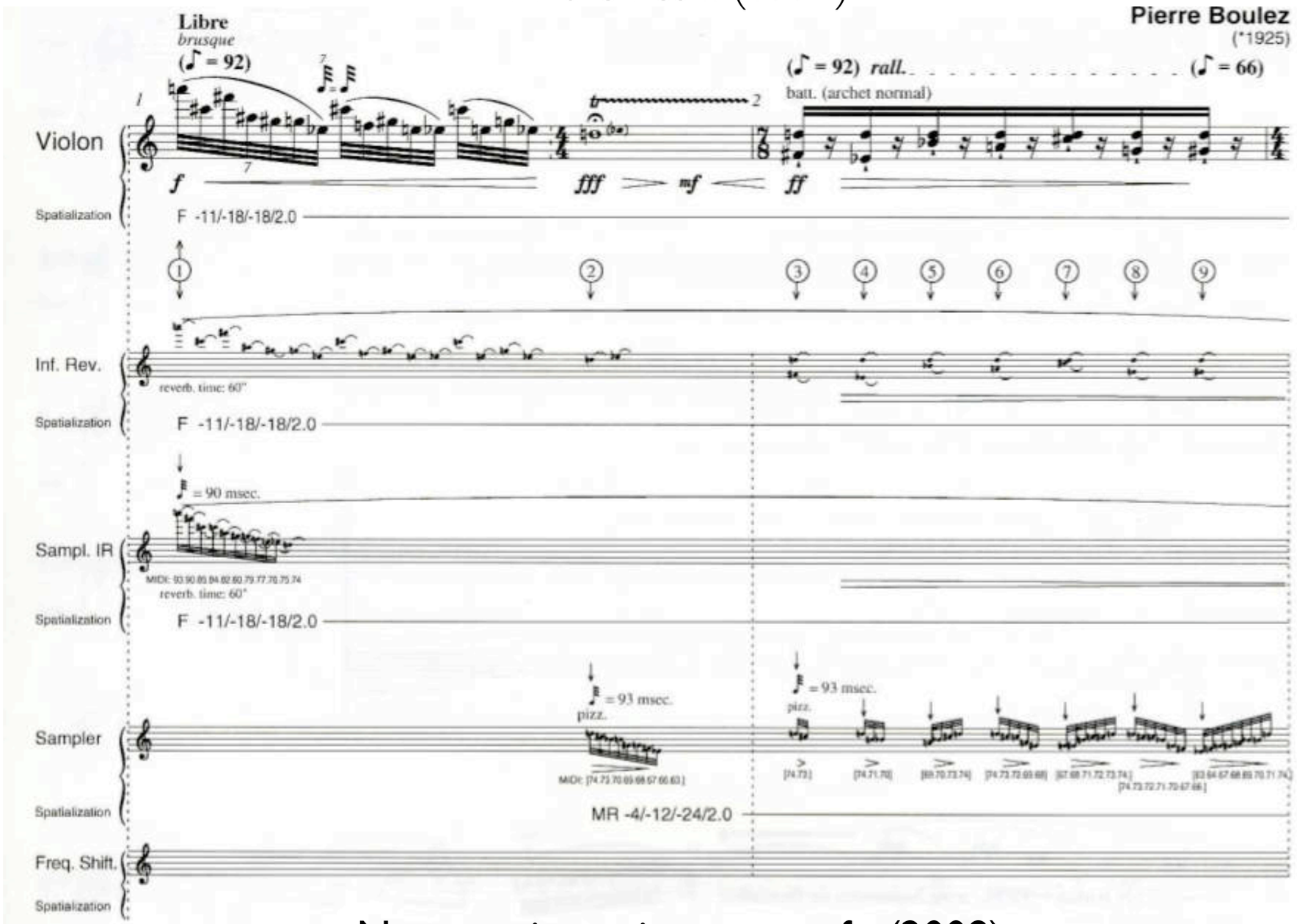
- Live Coding
- New reactive behaviors

The Antescofo Language

The Antescofo Language

Goal: Jointly specify electronic and instrumental parts

Anthèmes II (1994)

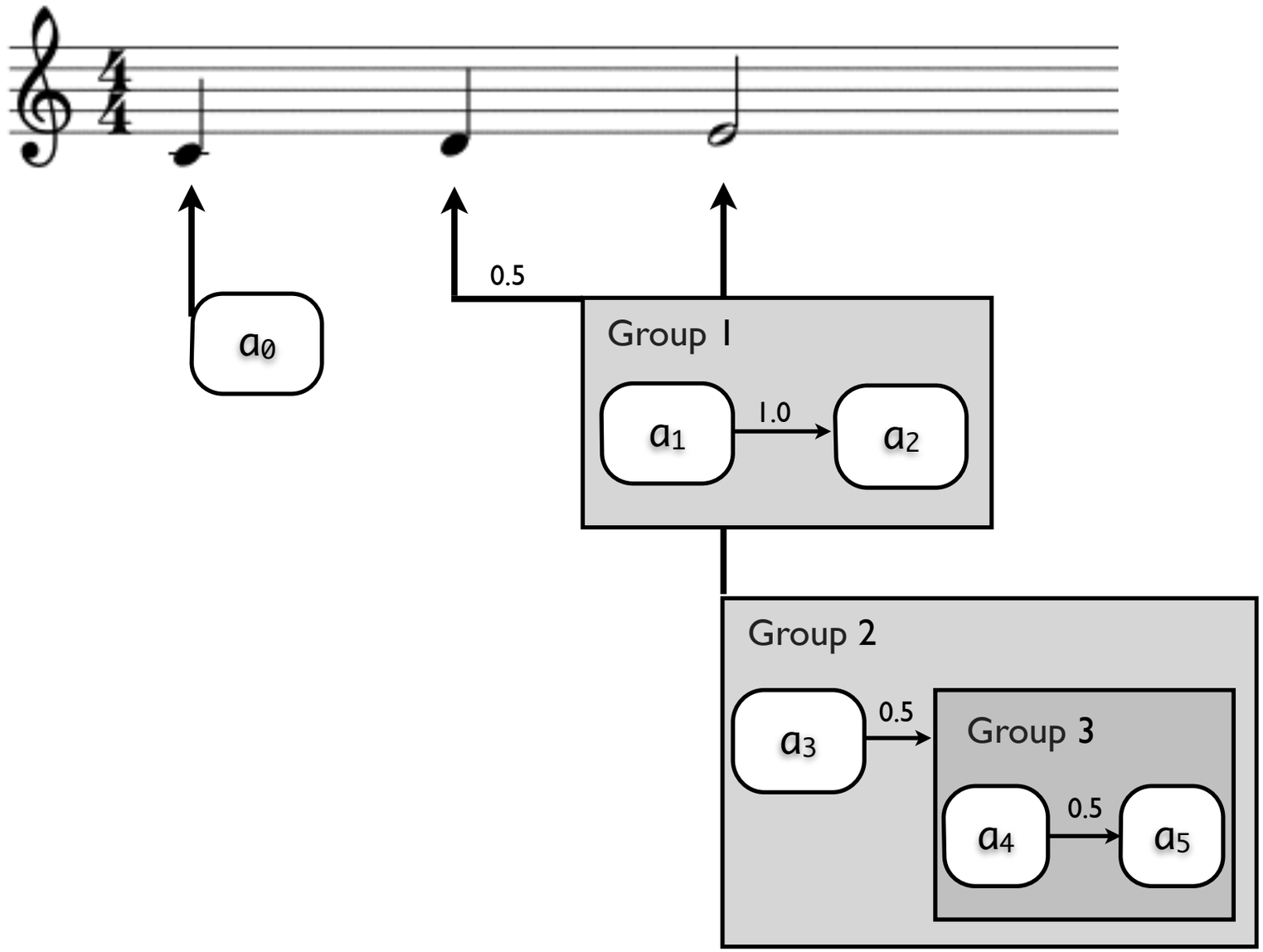


New version using antescofo (2008)

The Antescofo Language

Goal: Jointly specify electronic and instrumental parts

[Echeveste et al. 2012]



The Antescofo Language

Goal: Jointly specify electronic and instrumental parts

[Echeveste et al. 2012]

```
NOTE   65   1.0
0.25   GROUP   tight   partial
          {  1.0   'a_11'
             1.0   'a_12'  }

CHORD   (68 54)   0.5
1.0   'a_21'
0.5   GROUP   loose   causal
         {  1.0   'a_22'
            0.0   GROUP   loose   causal
                     {  0.25   'a_23'
                        0.25   'a_24'  }
            1.0   'a_25'  }

NOTE   52   2.0
0.5   'a_31'
2.5   'a_32'
```

The Antescofo Language

Goal: Jointly specify electronic and instrumental parts

[Echeveste et al. 2012]

Instrumental Score

```
NOTE   65   1.0
0.25   GROUP   tight   partial
          {  1.0   'a_11'
             1.0   'a_12'  }

CHORD   (68 54)   0.5
1.0   'a_21'
0.5   GROUP   loose   causal
          {  1.0   'a_22'
             0.0   GROUP   loose   causal
                      {  0.25   'a_23'
                         0.25   'a_24'  }
             1.0   'a_25'  }

NOTE   52   2.0
0.5   'a_31'
2.5   'a_32'
```

The Antescofo Language

Goal: Jointly specify electronic and instrumental parts

[Echeveste et al. 2012]

```
NOTE   65   1.0
0.25   GROUP   tight   partial
         {  1.0   'a_11'
            1.0   'a_12'  }

CHORD   (68 54)   0.5
1.0   'a_21'
0.5   GROUP   loose   causal
        {  1.0   'a_22'
           0.0   GROUP   loose   causal
                   {  0.25   'a_23'
                      0.25   'a_24'  }
           1.0   'a_25'  }

NOTE   52   2.0
0.5   'a_31'
2.5   'a_32'
```

Electronic Score

The Antescofo Language

Goal: Jointly specify electronic and instrumental parts

[Echeveste et al. 2012]

```
NOTE   65   1.0
0.25   GROUP   tight   partial
          {  1.0   'a_11'
             1.0   'a_12'  }

CHORD   (68 54)   0.5
1.0   'a_21'
0.5   GROUP   loose   causal
          {  1.0   'a_22'
             0.0   GROUP   loose   causal
                      {  0.25   'a_23'
                         0.25   'a_24'  }
                1.0   'a_25'  }

NOTE   52   2.0
0.5   'a_31'
2.5   'a_32'
```

Delay relative to the tempo

The Antescofo Language

Goal: Jointly specify electronic and instrumental parts

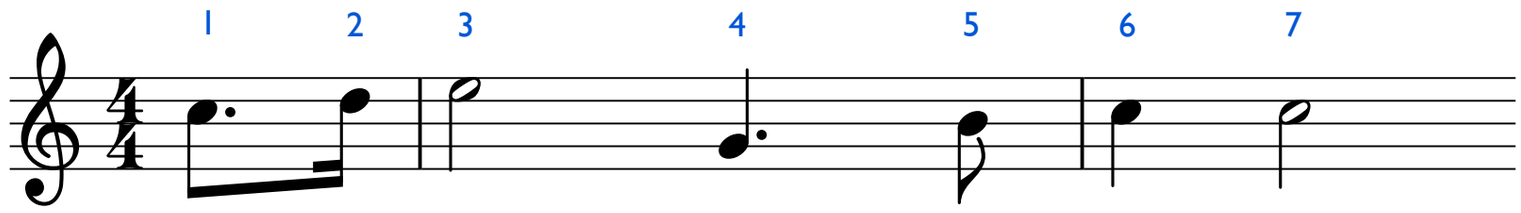
[Echeveste et al. 2012]

```
NOTE   65   1.0
0.25   GROUP   tight   partial
          {  1.0   'a_11'
             1.0   'a_12'  }

CHORD   (68 54)   0.5
1.0   'a_21'
0.5   GROUP   loose   causal
          {  1.0   'a_22'
             0.0   GROUP   loose   causal
                      {  0.25   'a_23'
                         0.25   'a_24'  }
             1.0   'a_25'  }

NOTE   52   2.0
0.5   'a_31'
2.5   'a_32'
```

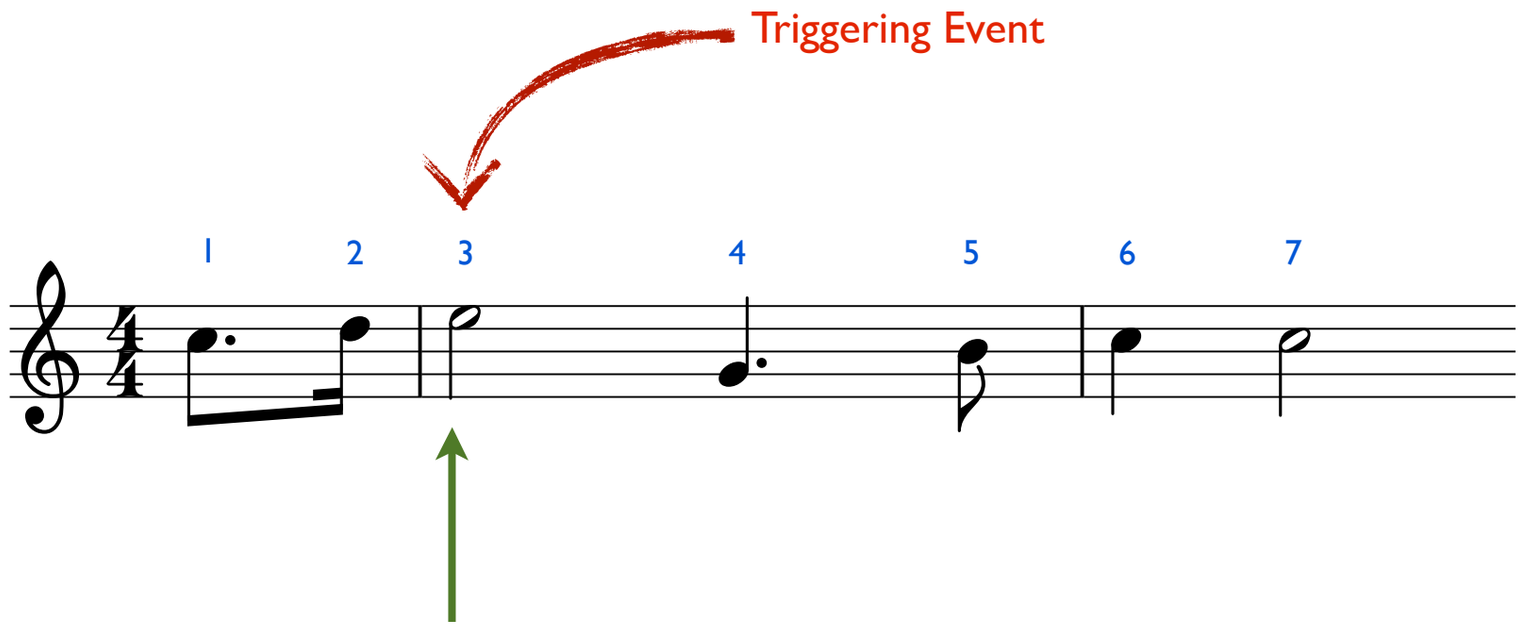
Group Attributes

Synchronization

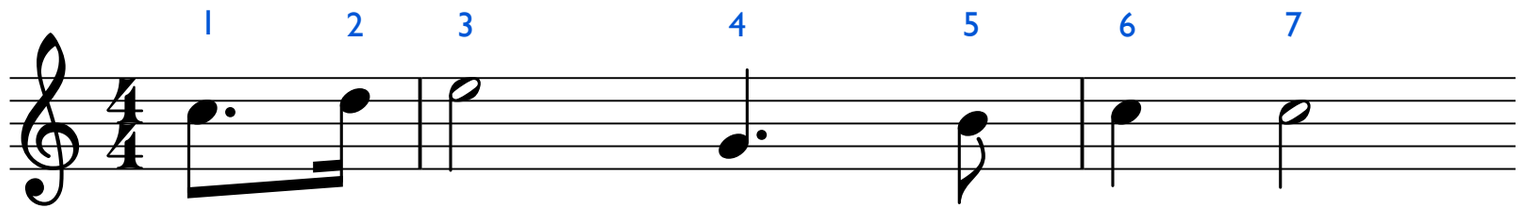
Strategies



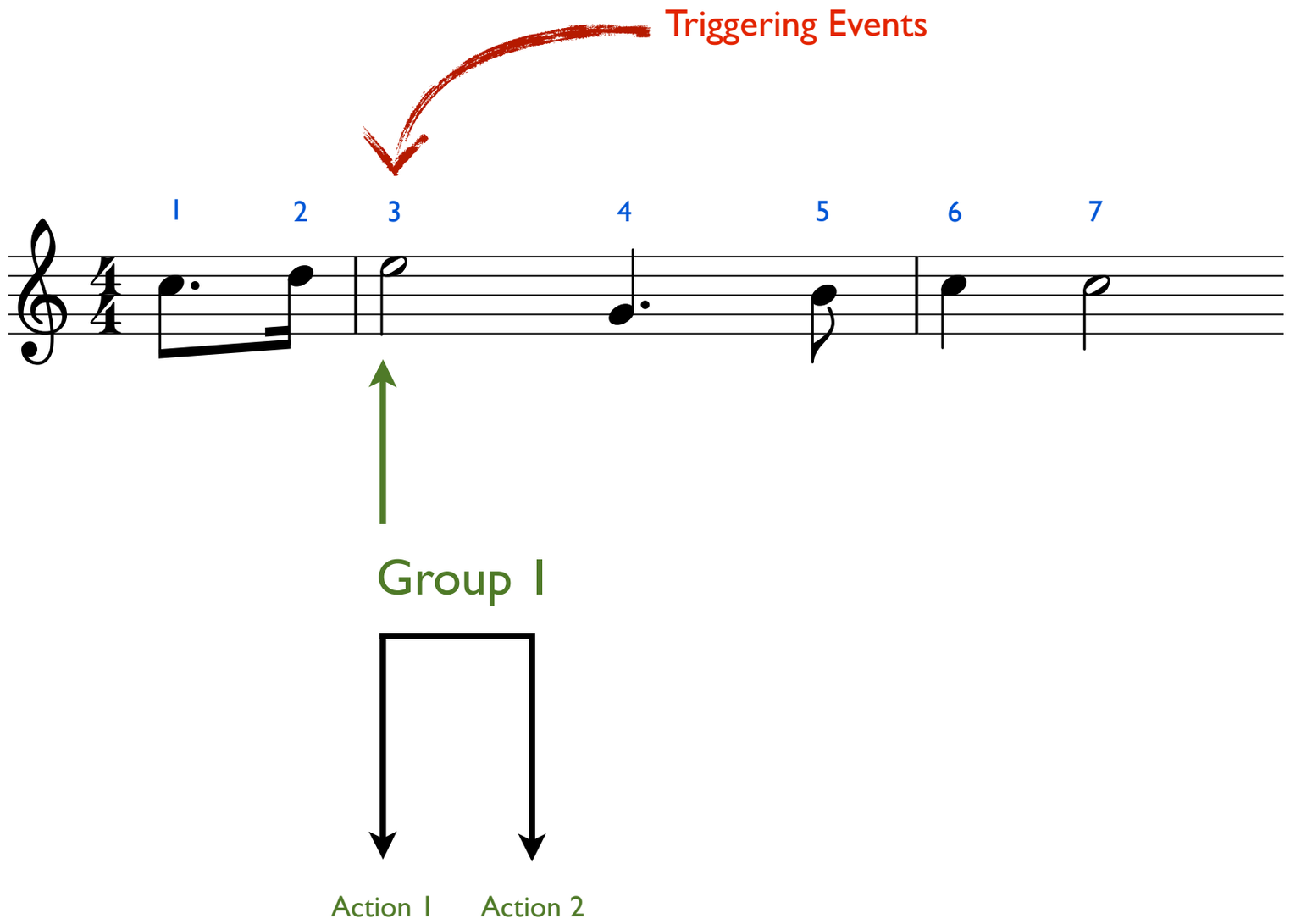
Loose: Synchronization
with the tempo stream.



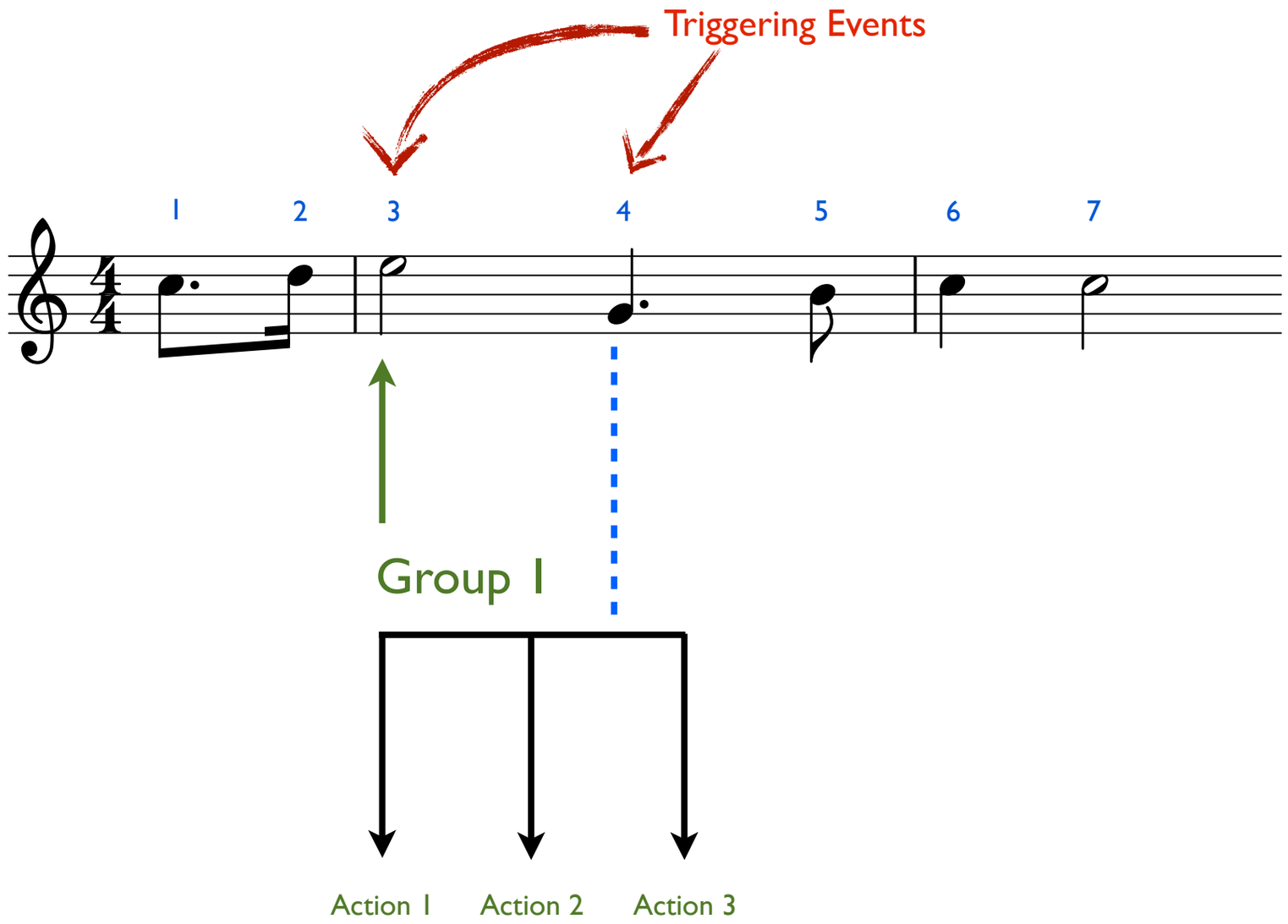
Loose: Synchronization with the tempo stream.



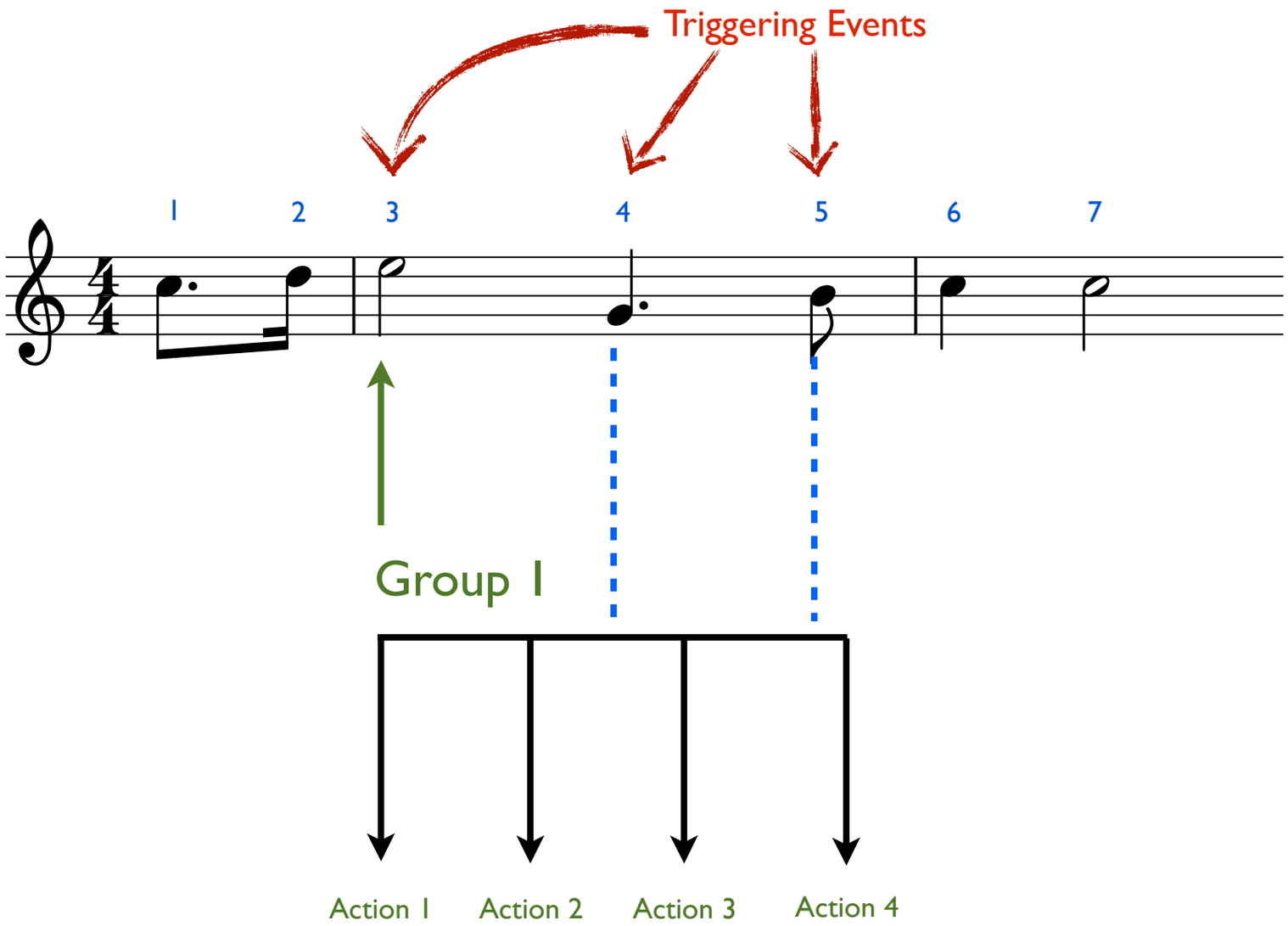
Tight: Synchronization
with tempo and events stream.



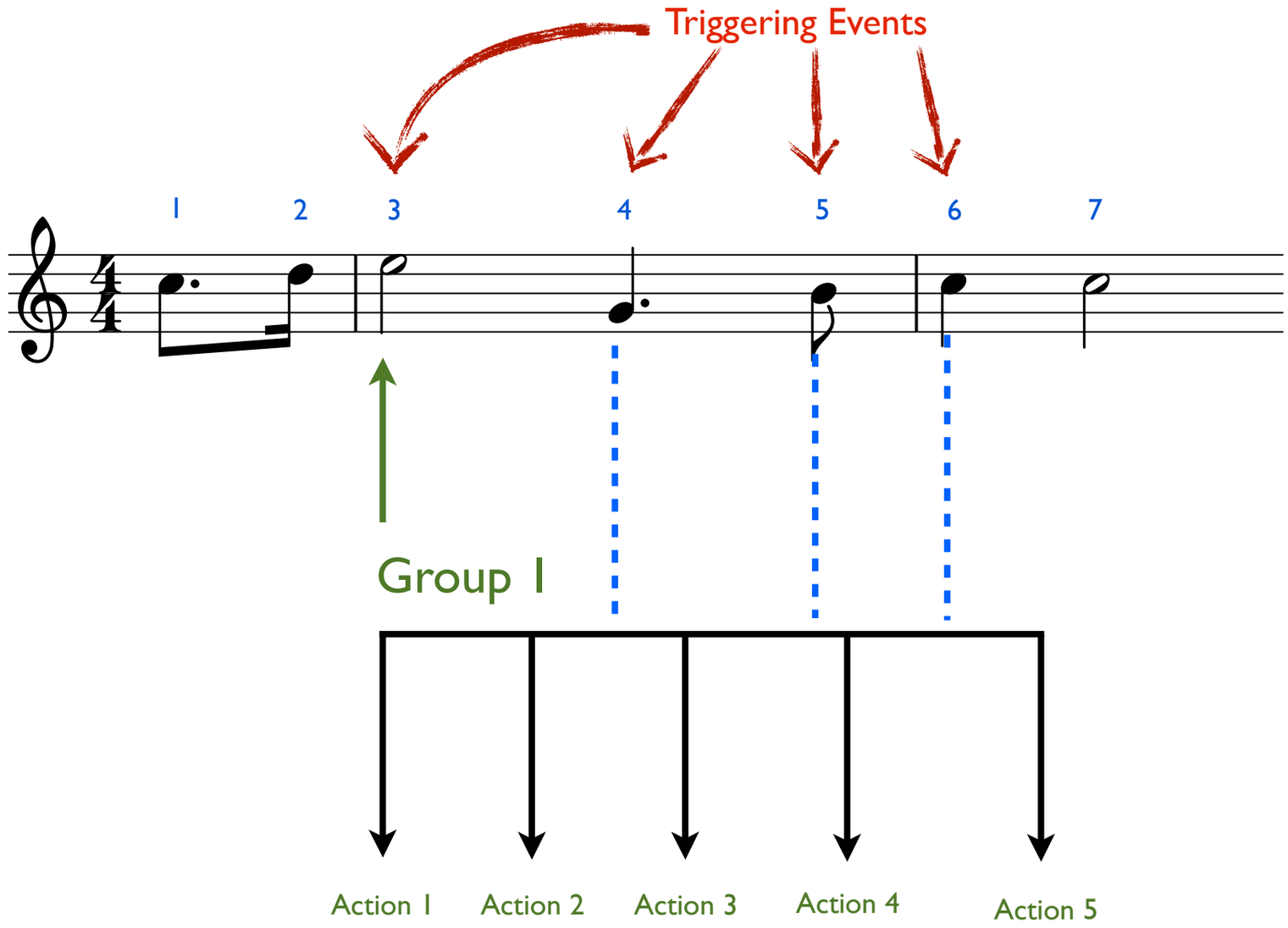
Tight: Synchronization with tempo and events stream.



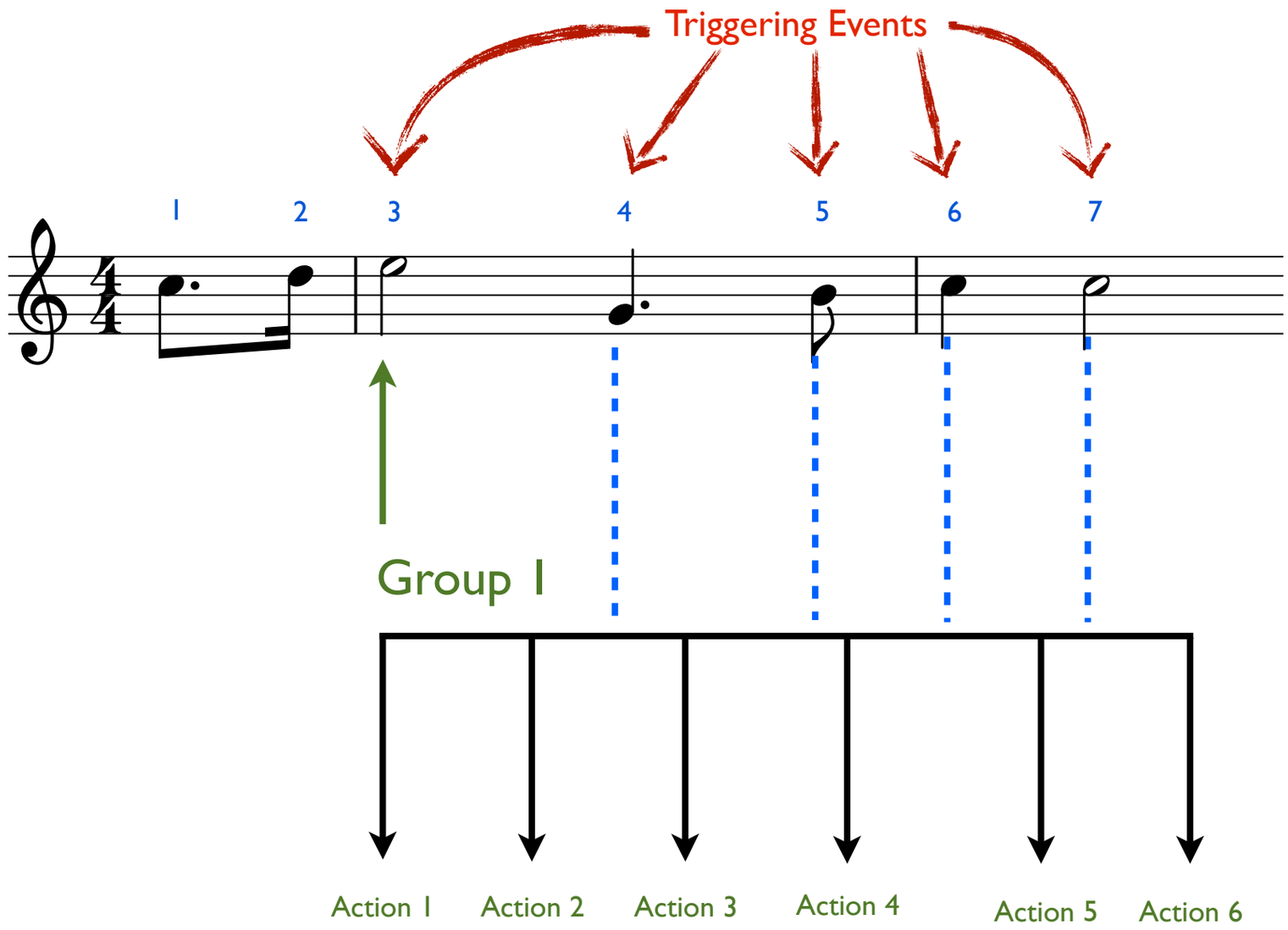
Tight: Synchronization with tempo and events stream.



Tight: Synchronization
with tempo and events stream.



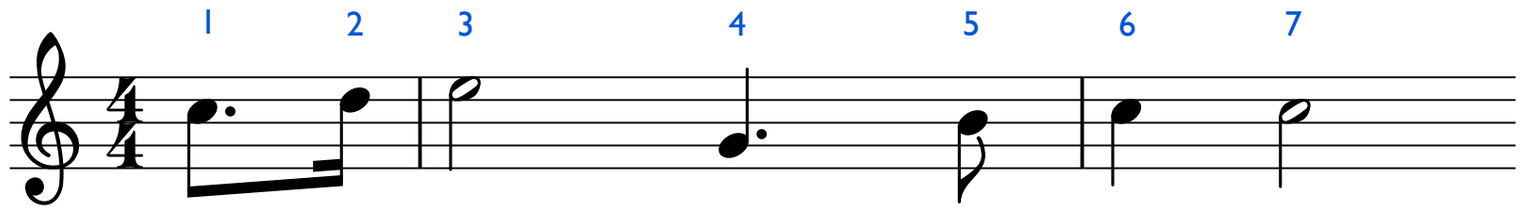
Tight: Synchronization with tempo and events stream.



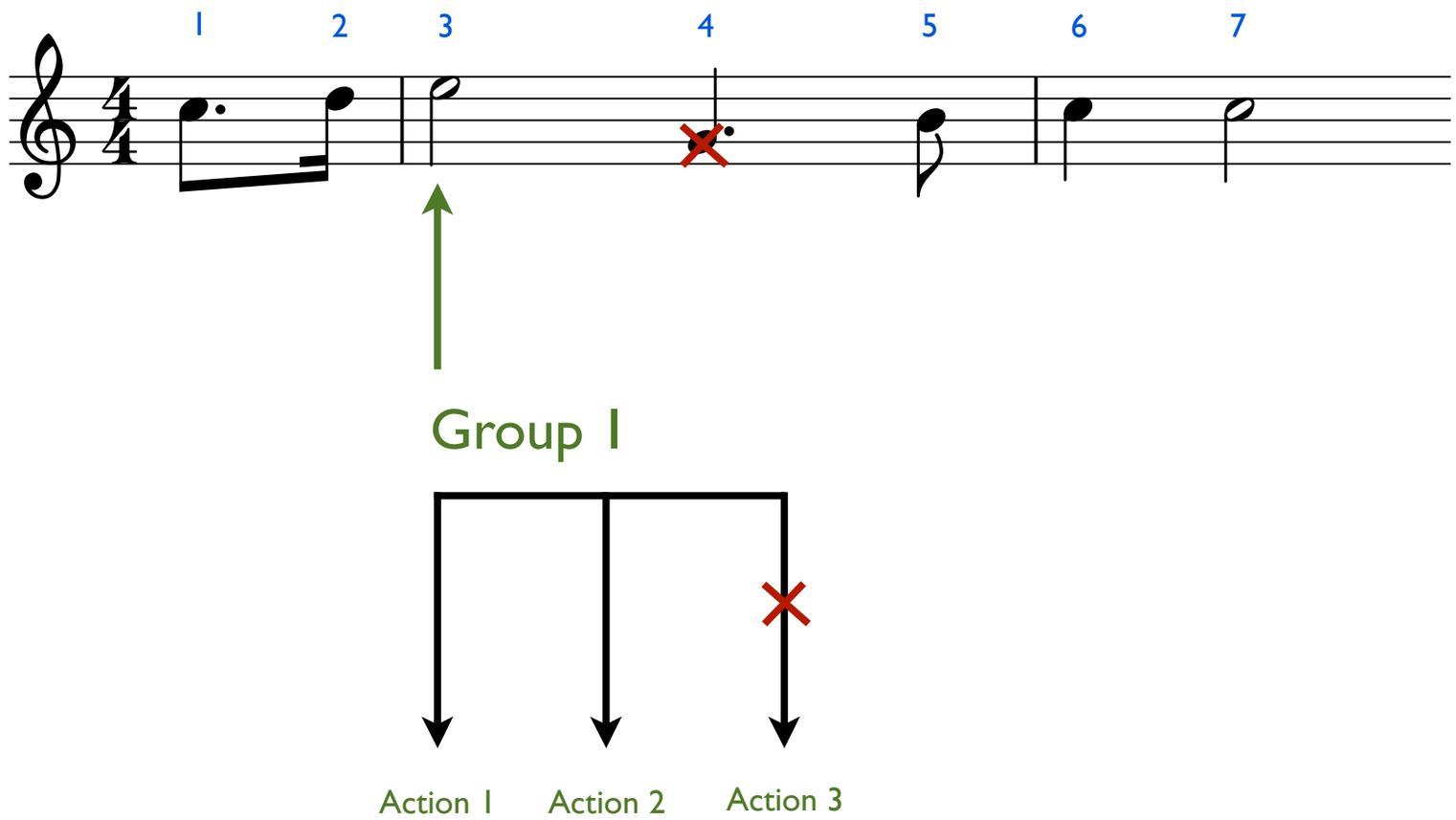
Tight: Synchronization with tempo and events stream.

Error Handling

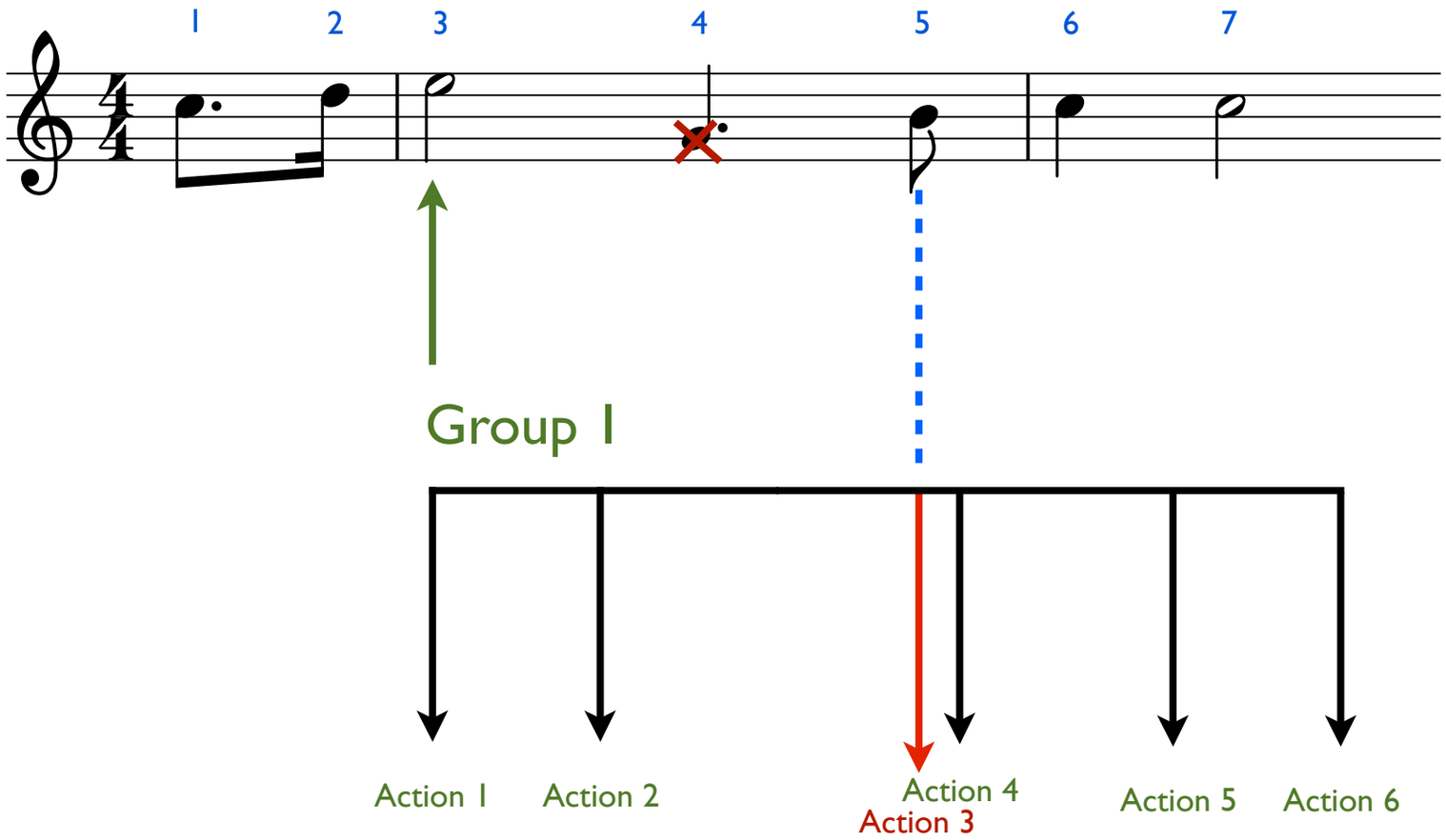
Strategies



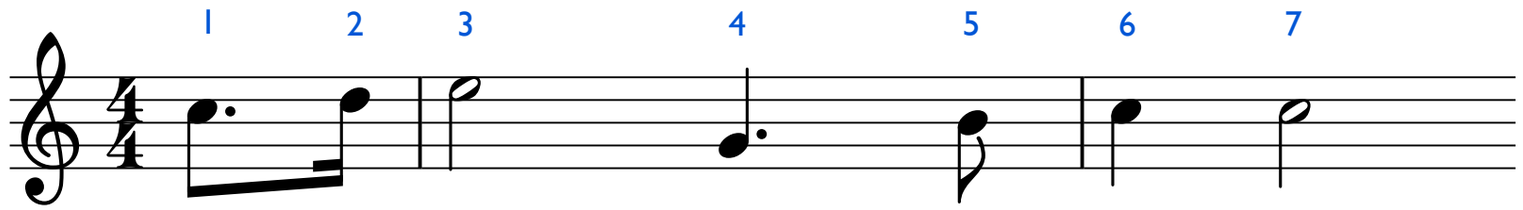
Causal: Actions should be launched immediately when the system recognizes the absence of the triggering event



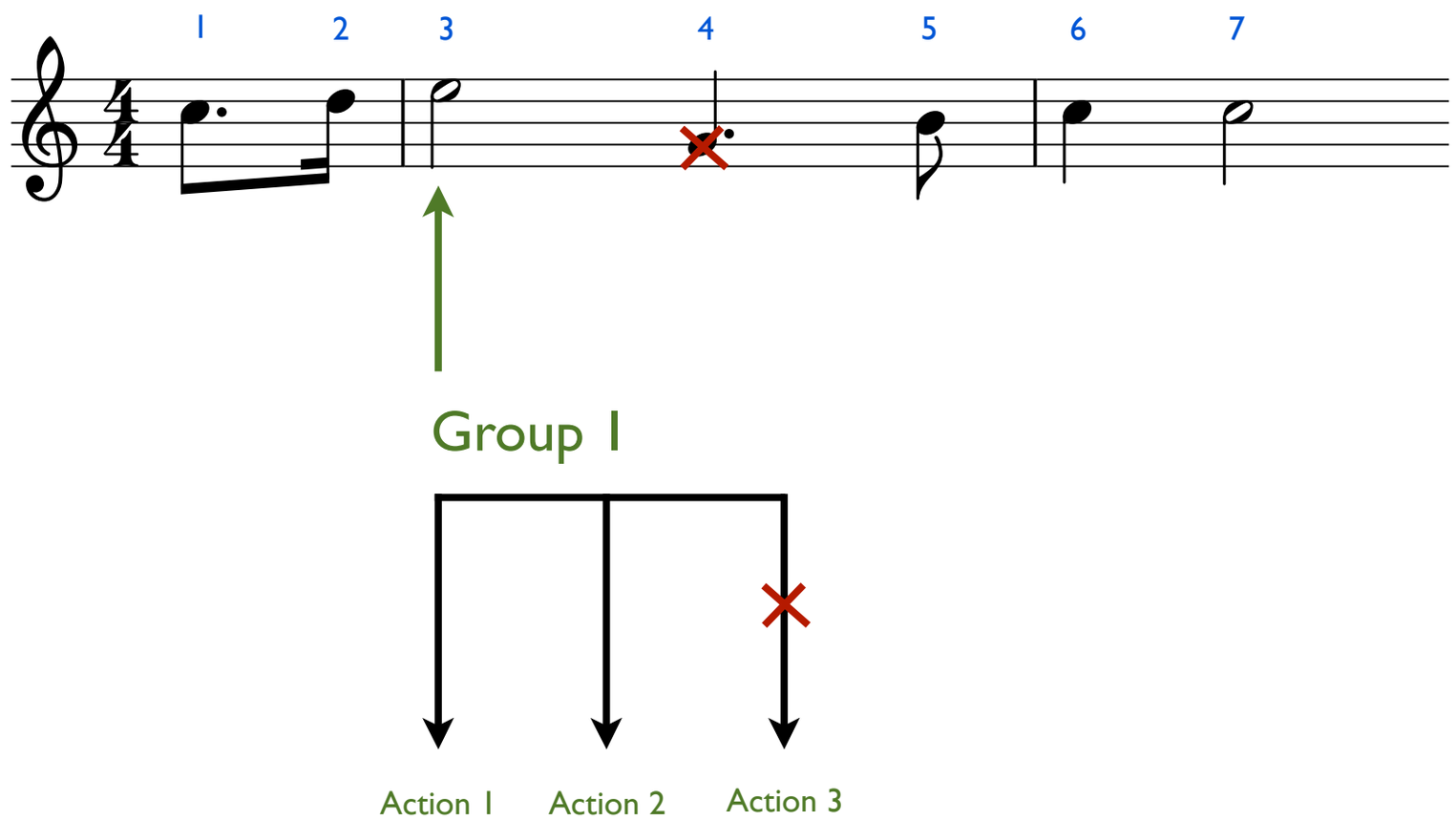
Causal: Actions should be launched immediately when the system recognizes the absence of the triggering event



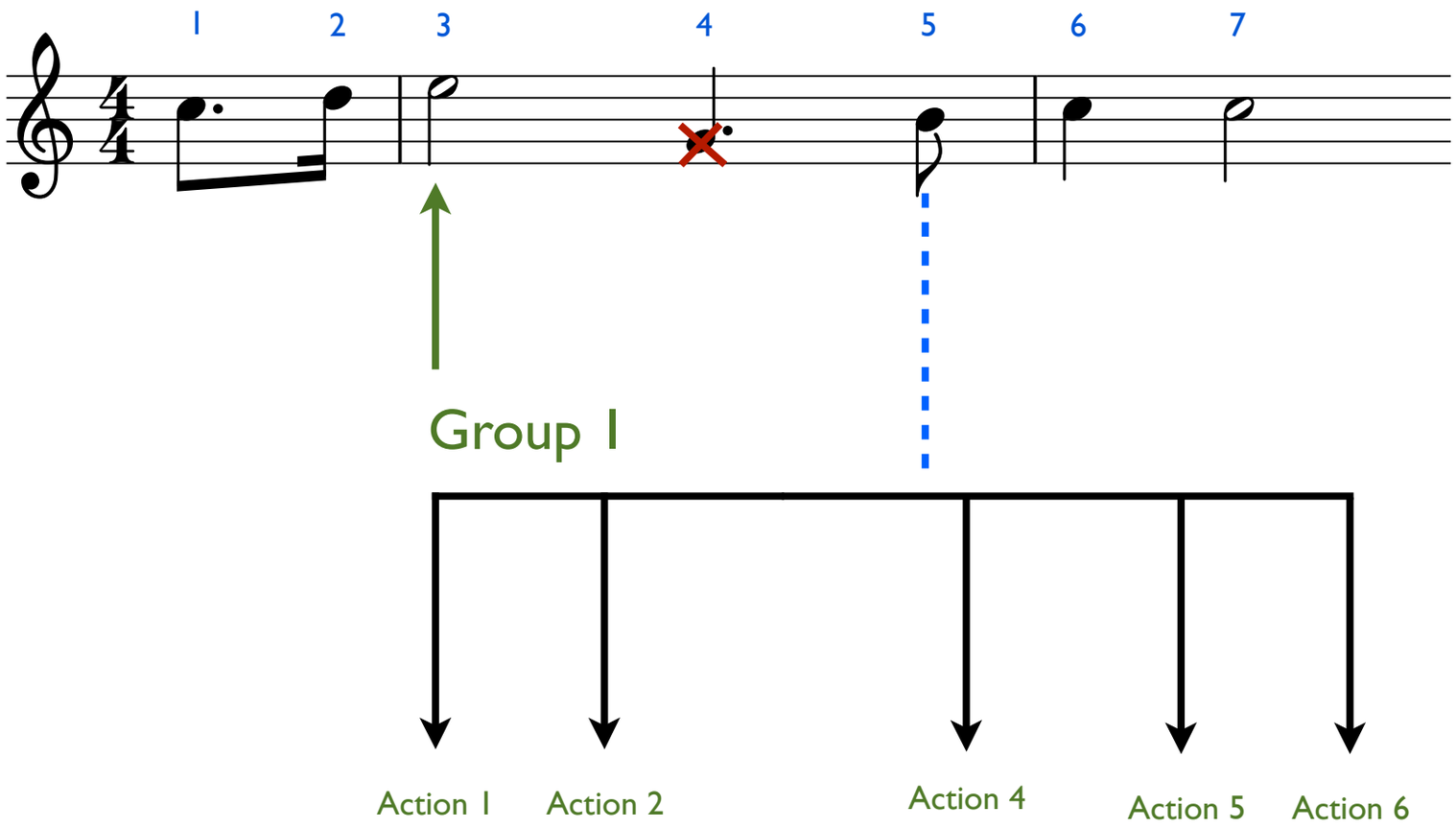
Causal: Actions should be launched immediately when the system recognizes the absence of the triggering event



Partial: Actions should be dismissed in the absence of the triggering event



Partial: Actions should be dismissed in the absence of the triggering event



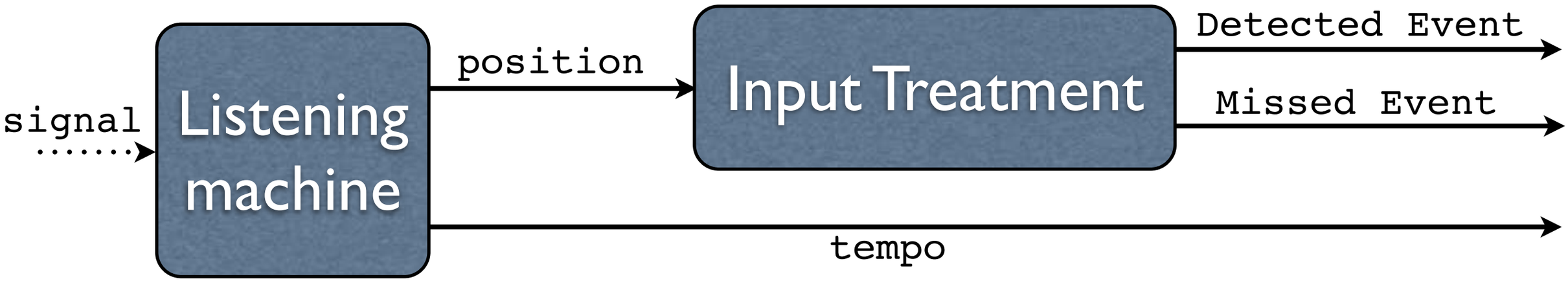
Partial: Actions should be dismissed in the absence of the triggering event

Language Characteristics

- A global logical time relative to the tempo
- Specify electronic actions with:
  - synchronization strategies
  - error handling strategies
- **Composer friendly**

Semantics

Detected and Missed Event

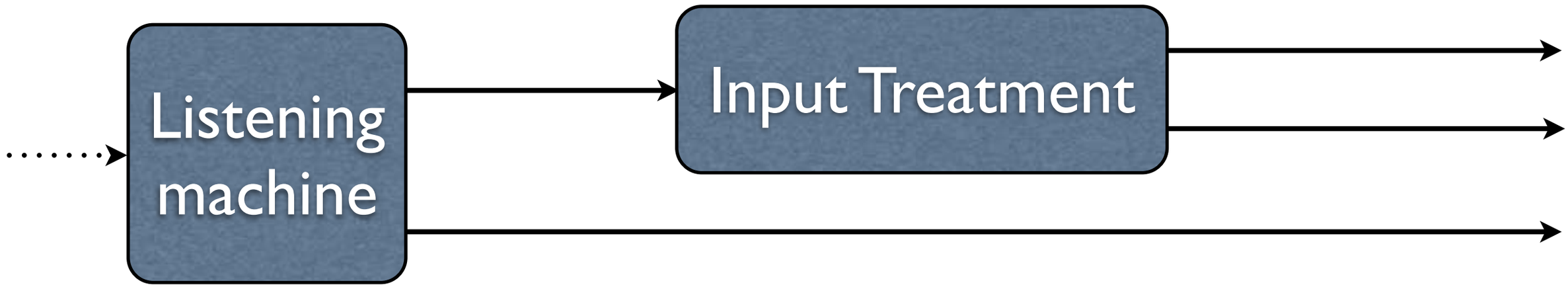


D: the set of detected instrumental events

For each missed event i we associate the next detected event

$$\mathcal{M}(i) = \min\{j \in D \mid j > i\}$$

Detected and Missed Event

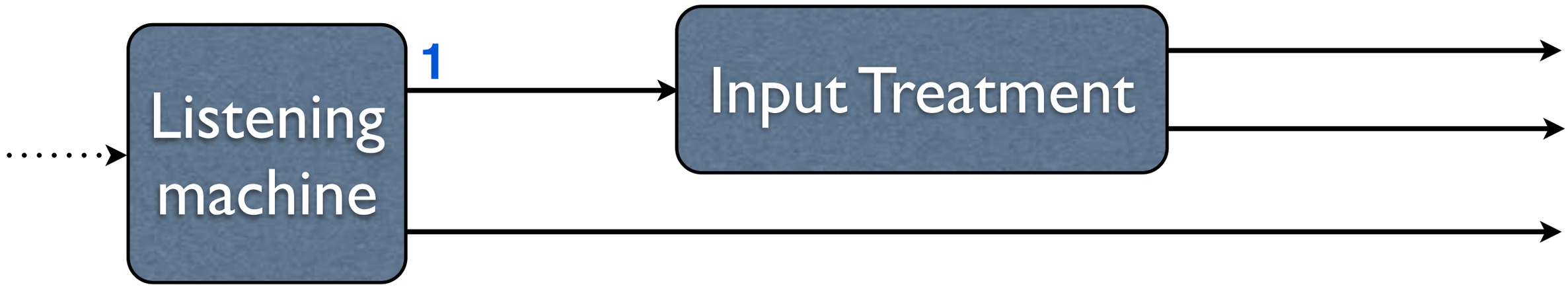


D: the set of detected instrumental events

For each missed event i we associate the next detected event

$$\mathcal{M}(i) = \min\{j \in D \mid j > i\}$$

Detected and Missed Event

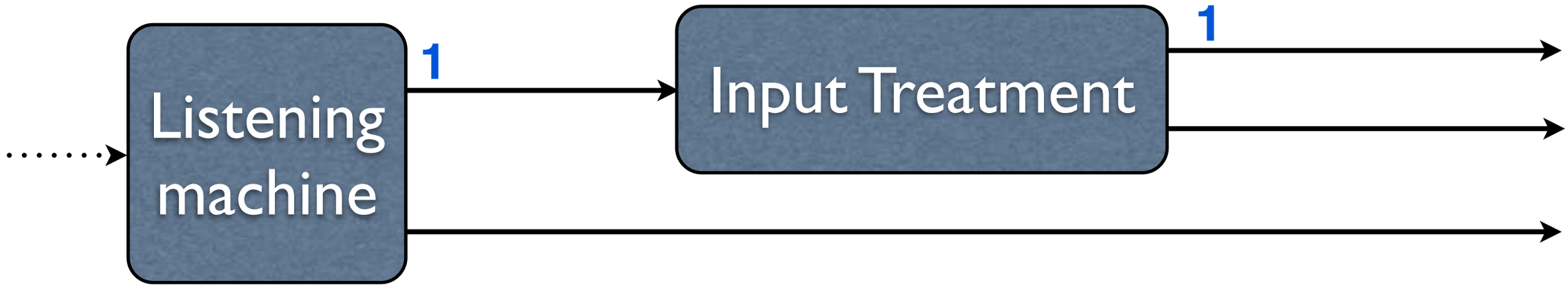


D: the set of detected instrumental events

For each missed event i we associate the next detected event

$$\mathcal{M}(i) = \min\{j \in D \mid j > i\}$$

Detected and Missed Event

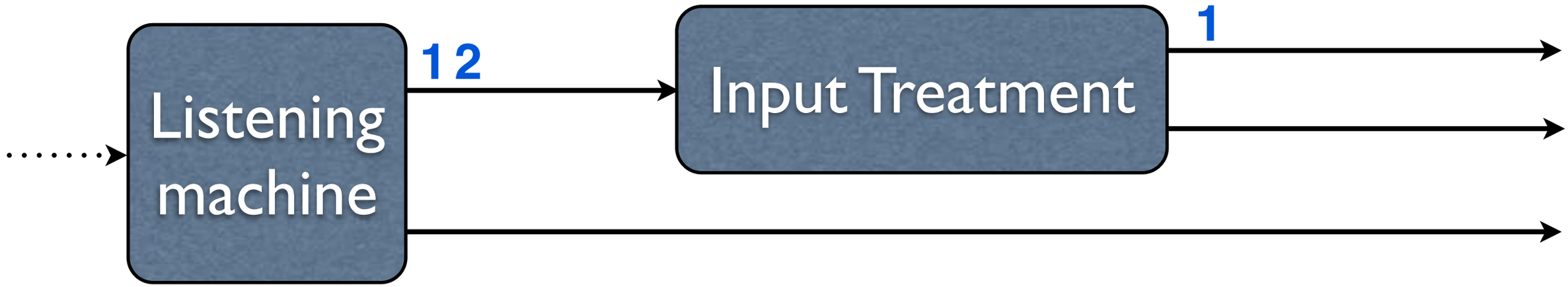


D: the set of detected instrumental events

For each missed event i we associate the next detected event

$$\mathcal{M}(i) = \min\{j \in D \mid j > i\}$$

Detected and Missed Event

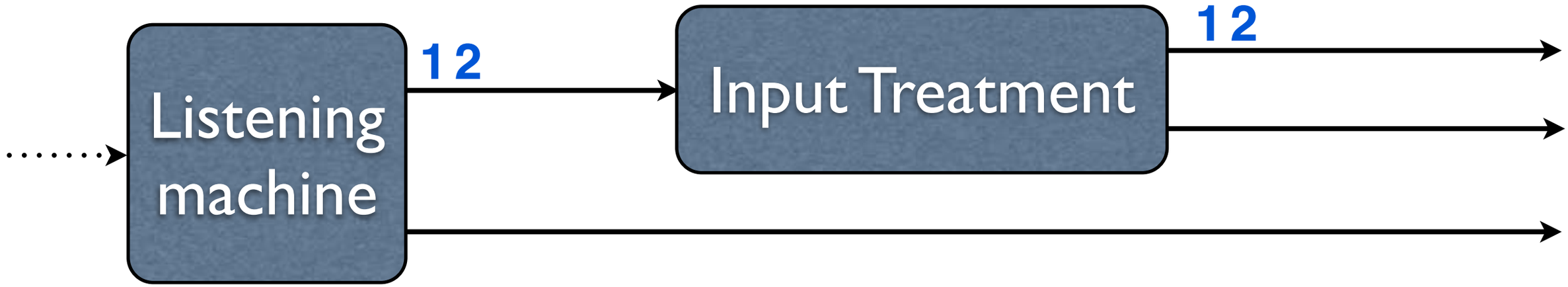


D: the set of detected instrumental events

For each missed event i we associate the next detected event

$$\mathcal{M}(i) = \min\{j \in D \mid j > i\}$$

Detected and Missed Event

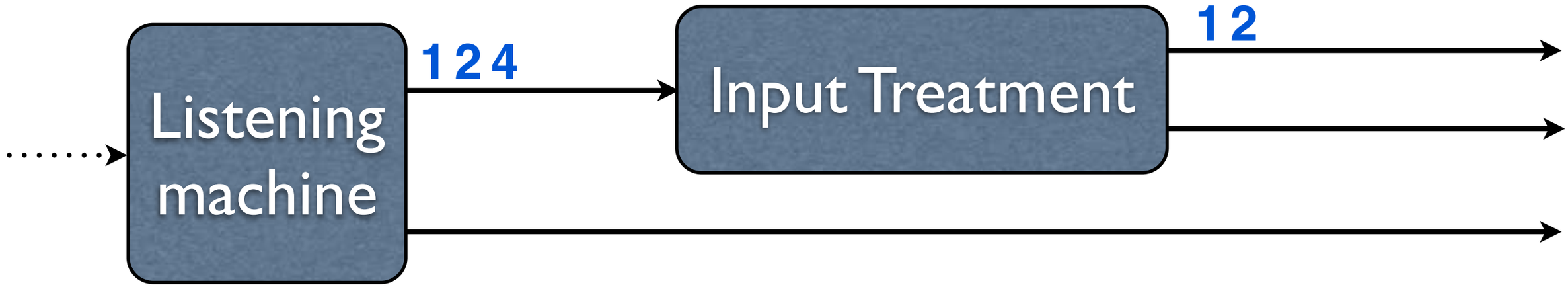


D: the set of detected instrumental events

For each missed event i we associate the next detected event

$$\mathcal{M}(i) = \min\{j \in D \mid j > i\}$$

Detected and Missed Event

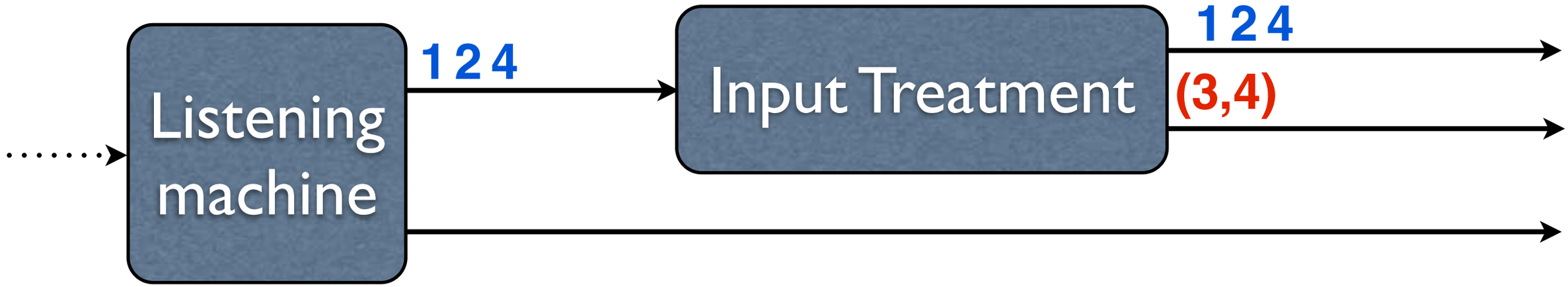


D: the set of detected instrumental events

For each missed event i we associate the next detected event

$$\mathcal{M}(i) = \min\{j \in D \mid j > i\}$$

Detected and Missed Event

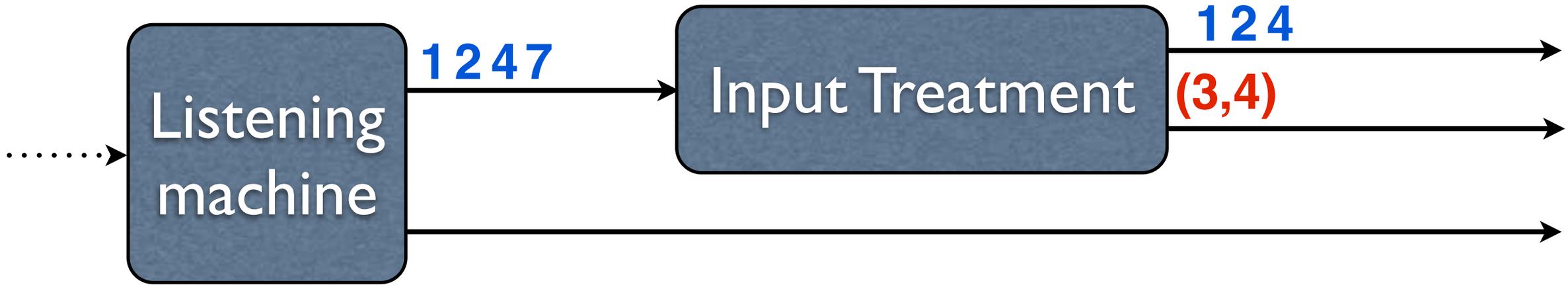


D: the set of detected instrumental events

For each missed event i we associate the next detected event

$$\mathcal{M}(i) = \min\{j \in D \mid j > i\}$$

Detected and Missed Event

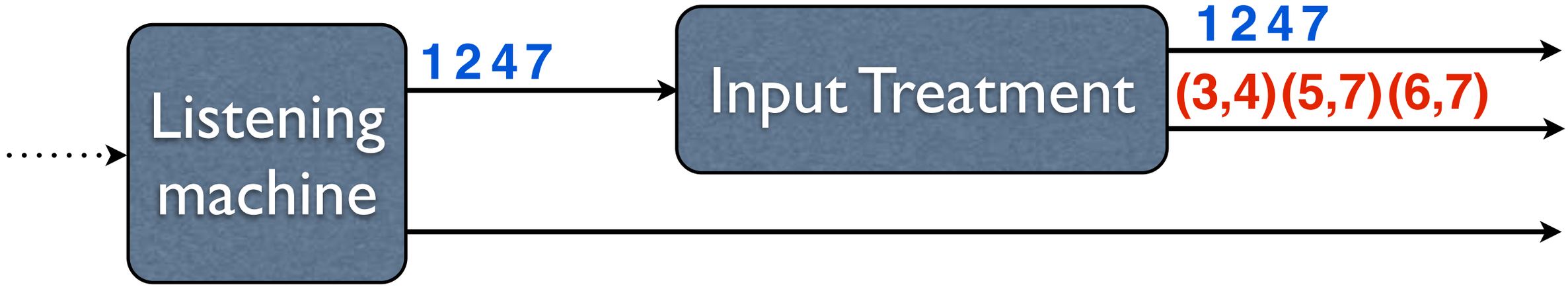


D: the set of detected instrumental events

For each missed event i we associate the next detected event

$$\mathcal{M}(i) = \min\{j \in D \mid j > i\}$$

Detected and Missed Event



D: the set of detected instrumental events

For each missed event i we associate the next detected event

$$\mathcal{M}(i) = \min\{j \in D \mid j > i\}$$

Formalization

$$
\begin{array}{rcl}
score & ::= & \varepsilon \mid (event : seq)\ score \\
event & ::= & \texttt{event}\ i\ t \\
seq & ::= & \varepsilon \mid (\delta\ ae)\ seq \\
ae & ::= & action \mid group \\
group & ::= & \texttt{group}\ synchro\ error\ seq \\
synchro & ::= & \texttt{tight} \mid \texttt{loose} \\
error & ::= & \texttt{local} \mid \texttt{global} \mid \texttt{partial} \mid \texttt{causal}
\end{array}
$$

$t \in \mathbb{Q}$ Duration

$\delta \in \mathbb{Q}$ Delay

$i \in \mathbb{N}$ Label

$a \in \mathcal{A}$ Action

A performance $perf$ is a set of triplets (i, δ, a)

D is the set of detected instrumental events

Semantics

$$D \vdash^{exec} score \Rightarrow perf$$

The Three Predicates

$$D \vdash^{exec} score \Rightarrow perf$$

Execute a score

$$D, i, \delta \vdash^{detected} seq \Rightarrow perf$$

Execute a sequence of actions bound to a detected event i with a delay δ

$$D, i, \delta \vdash^{missed} seq \Rightarrow perf$$

Execute a sequence of actions bound to a missed event i with a delay δ

Execution of a score

$$\text{(Empty Score)}\quad \frac{}{D \vdash^{exec} \varepsilon \Rightarrow \varnothing}$$

$$\text{(Exec Score)}\quad \frac{D \vdash^{exec} (\texttt{event}\ i\ t : seq) \to p_1 \qquad D \vdash^{exec} sc \Rightarrow p_2}{D \vdash^{exec} (\texttt{event}\ i\ t : seq)\ sc \Rightarrow p_1 \cup p_2}$$

Binding

$$\text{(Detect)}\quad \frac{i \in D \qquad D, i, 0.0 \mid\!\!\overset{detected}{\vdash} seq \Rightarrow p}{D \mid\!\!\overset{exec}{\vdash} (\texttt{event}\ i\ t : seq) \to p}$$

$$\text{(Miss)}\quad \frac{i \notin D \qquad D, i, 0.0 \mid\!\!\overset{missed}{\vdash} seq \Rightarrow p}{D \mid\!\!\overset{exec}{\vdash} (\texttt{event}\ i\ t : seq) \to p}$$

Execution: Atomic Actions

$$\text{(Detected Action)} \quad \frac{}{D, i, \delta \mid\!\!\xrightarrow{detected} a \to (i, \delta, a)}$$

$$\text{(Missed Action)} \quad \frac{\mathcal{M}(i) = j \qquad \delta' = \max(0.0, \mathcal{E}(i) + \delta - \mathcal{E}(j))}{D, i, \delta \mid\!\!\xrightarrow{missed} a \to (j, \delta', a)}$$

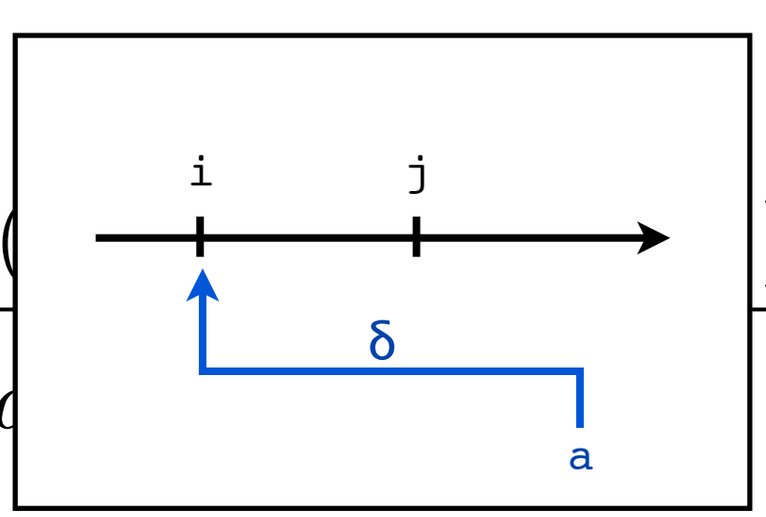
$\mathcal{E}(i)$: date of event i

$\mathcal{M}(i) = \min\{j \in D \mid j > i\}$
Error detection: i is missed
j is the first detection after i

Execution: Atomic Actions

$$\text{(Detected Action)} \quad \frac{}{D, i, \delta \overset{detected}{\vdash} a \to (i, \delta, a)}$$

$$\text{(Missed Action)} \quad \frac{\mathcal{M}(i) = j \qquad \delta' = \max(\quad)}{D, i, \delta \overset{missed}{\vdash} a}$$



$\mathcal{E}(i)$: date of event i

$\mathcal{M}(i) = \min\{j \in D \mid j > i\}$

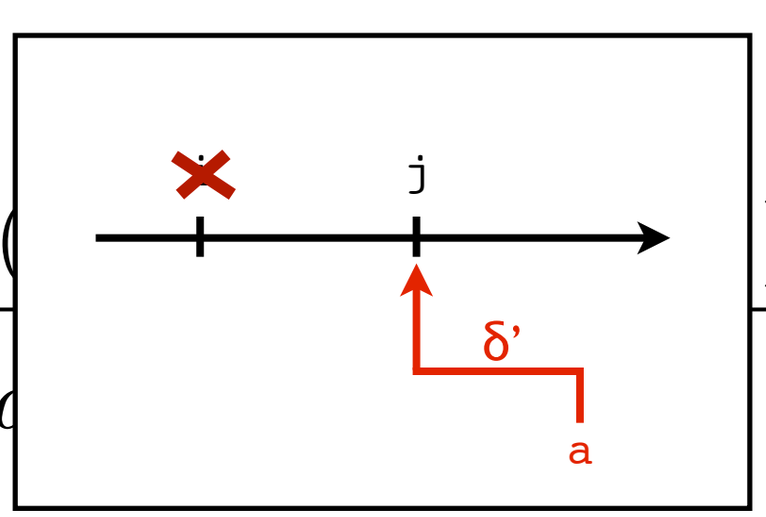
Error detection: i is missed

j is the first detection after i

Execution: Atomic Actions

$$\text{(Detected Action)} \quad \frac{}{D, i, \delta \overset{detected}{\vdash} a \to (i, \delta, a)}$$

$$\text{(Missed Action)} \quad \frac{\mathcal{M}(i) = j \qquad \delta' = \max(\quad)}{D, i, \delta \overset{missed}{\vdash} a}$$



$\mathcal{E}(i)$: date of event i

$\mathcal{M}(i) = \min\{j \in D \mid j > i\}$

Error detection: i is missed

j is the first detection after i

Execution: Atomic Actions

$$\text{(Detected Action)} \quad \frac{}{D, i, \delta \mid\!\!\xrightarrow{detected} a \to (i, \delta, a)}$$

$$\text{(Missed Action)} \quad \frac{\mathcal{M}(i) = j \qquad \delta' = \max(0.0, \mathcal{E}(i) + \delta - \mathcal{E}(j))}{D, i, \delta \mid\!\!\xrightarrow{missed} a \to (j, \delta', a)}$$

$\mathcal{E}(i)$: date of event i

$\mathcal{M}(i) = \min\{j \in D \mid j > i\}$
Error detection: i is missed
j is the first detection after i

Implementation

Synchronous Embedding

- **Several experiments**
  Heptagon, Lucid Synchrone, ReactiveML
- **Why ReactiveML?**
  - **Functional, typed language, on top of OCaml**
    recursion and higher order processes
  - **Dynamic features**
    difficult to get with Lustre/Esterel/…
    new interactions, live coding

ReactiveML

OCaml extended with synchronous features à la Esterel

[Mandel-Pouzet 2005]

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

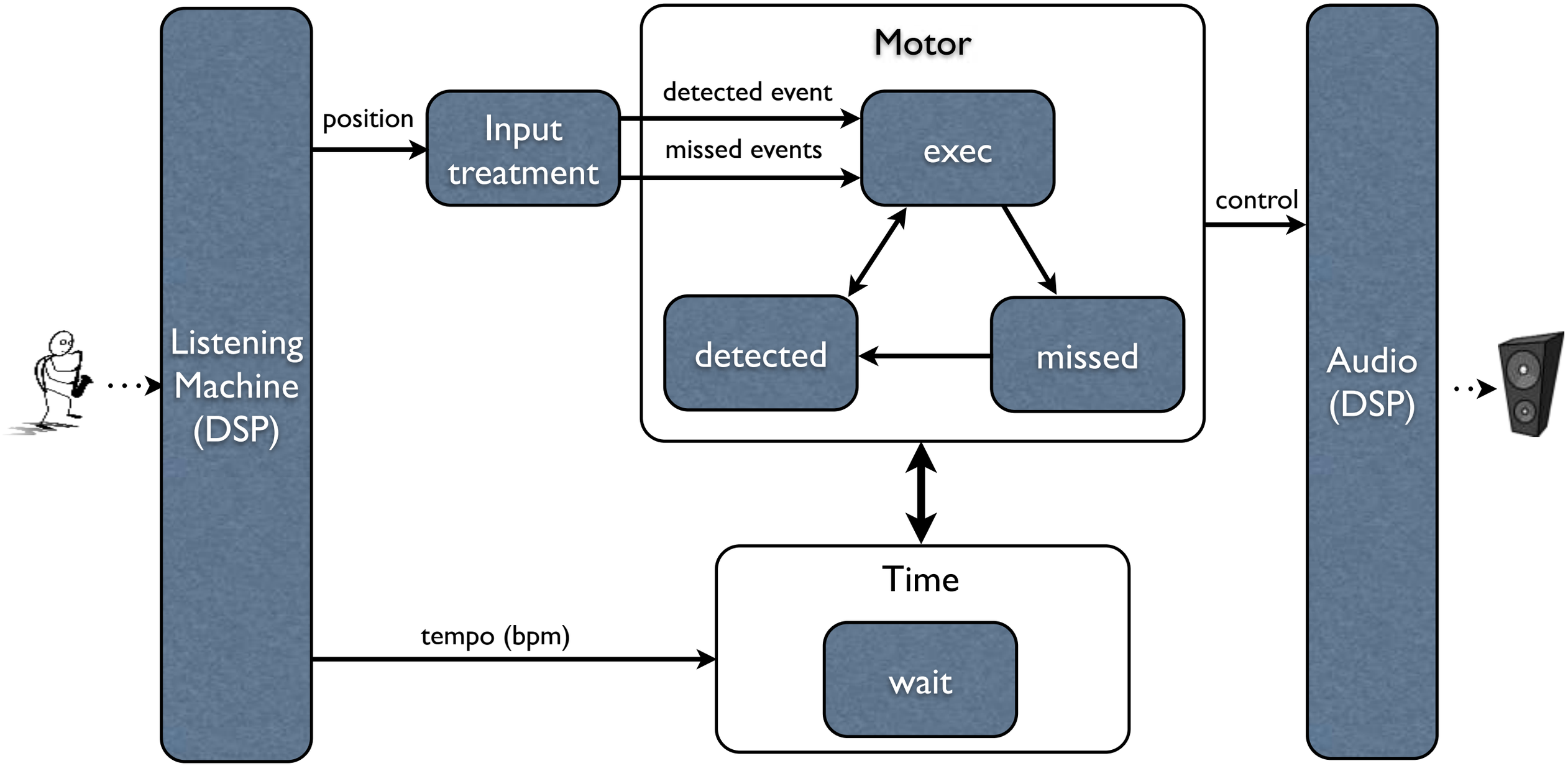
Definition: `signal` *`<id>`*

Emission: `emit` *`<id>`*

Waiting: `await` *`<id>`*

Broadcast communication between processes

Architecture



Execution of a score

$$\text{(Empty Score)}\quad \frac{}{D \vdash^{exec} \varepsilon \Rightarrow \emptyset}$$

$$\text{(Exec Score)}\quad \frac{D \vdash^{exec} (\texttt{event}\ i\ t : seq) \rightarrow p_1 \qquad D \vdash^{exec} sc \Rightarrow p_2}{D \vdash^{exec} (\texttt{event}\ i\ t : seq)\ sc \Rightarrow p_1 \cup p_2}$$

Execution of a score

(Empty Score) $$\frac{}{D \overset{exec}{\vdash} \varepsilon \Rightarrow \varnothing}$$

(Exec Score) $$\frac{D \overset{exec}{\vdash} (\texttt{event}\ i\ t : seq) \rightarrow p_1 \qquad D \overset{exec}{\vdash} sc \Rightarrow p_2}{D \overset{exec}{\vdash} (\texttt{event}\ i\ t : seq)\ sc \Rightarrow p_1 \cup p_2}$$

```
let rec process exec score =
  match score with
  | [] -> (* rule (Empty Score) *) ()
  | se::sc ->
      (* rule (Exec Score) *)
      run (exec_score_event se) ||
      run (exec sc)
```

Execution of a score

(Empty Score) $\dfrac{}{D \overset{exec}{\vdash} \varepsilon \Rightarrow \varnothing}$

(Exec Score) $\dfrac{D \overset{exec}{\vdash} (\texttt{event}\ i\ t : seq) \to p_1 \qquad D \overset{exec}{\vdash} sc \Rightarrow p_2}{D \overset{exec}{\vdash} (\texttt{event}\ i\ t : seq)\ sc \Rightarrow p_1 \cup p_2}$

```
let rec process exec score =
  match score with
  | [] -> (* rule (Empty Score) *) ()
  | se::sc ->
      (* rule (Exec Score) *)
      run (exec_score_event se) ||
      run (exec sc)
```

parallel composition

Binding

$$\text{(Miss)}\ \frac{i \notin D \qquad D, i, 0.0 \mid\!\!\overset{missed}{\vdash} seq \Rightarrow p}{D \mid\!\!\overset{exec}{\vdash} (\texttt{event}\ i\ t : seq)\ sc \to p} \qquad \text{(Detect)}\ \frac{i \in D \qquad D, i, 0.0 \mid\!\!\overset{detected}{\vdash} seq \Rightarrow p}{D \mid\!\!\overset{exec}{\vdash} (\texttt{event}\ i\ t : seq) \to p}$$

```
let rec process exec_score_event se =
  let status = run (wait_event se.event) in
match status with
| Detected ->
    (* rule (Detect) *)
    run (exec_seq (detected i) 0.0 se.seq)
| Missed(j) ->
    (* rule (Miss) *)
    run (exec_seq (missed i j) 0.0 se.seq)
```

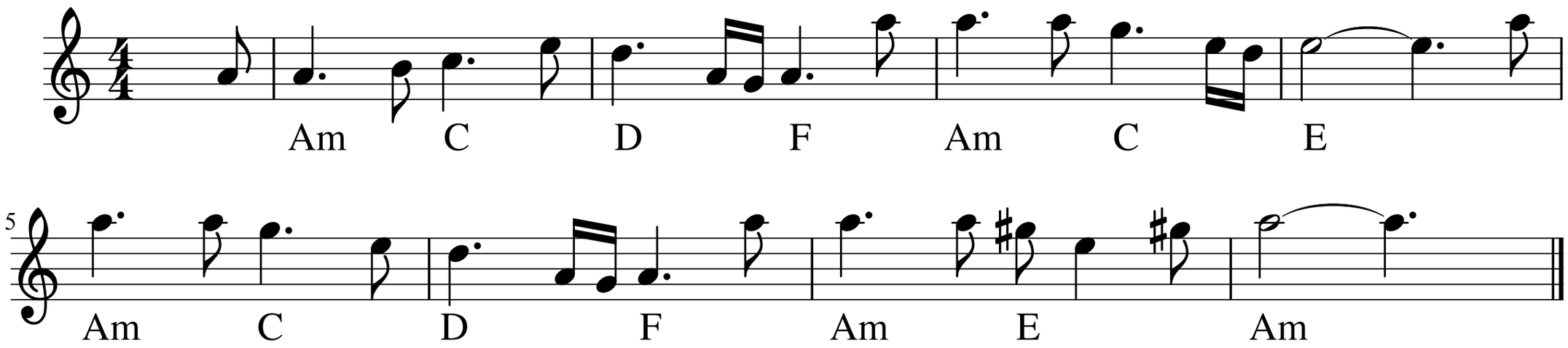
Applications

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 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

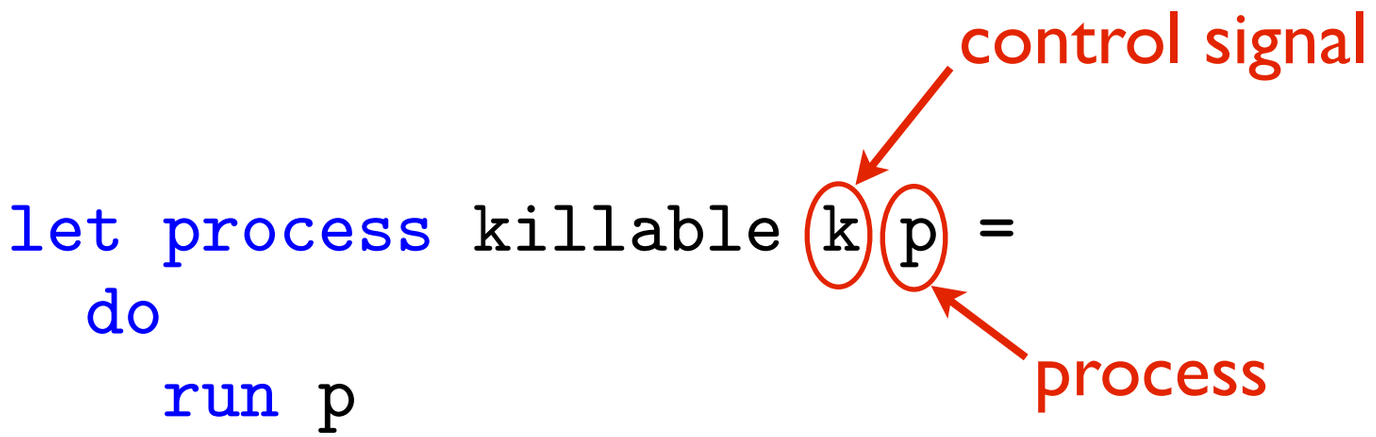
Example of a higher-order process

```
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
```

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

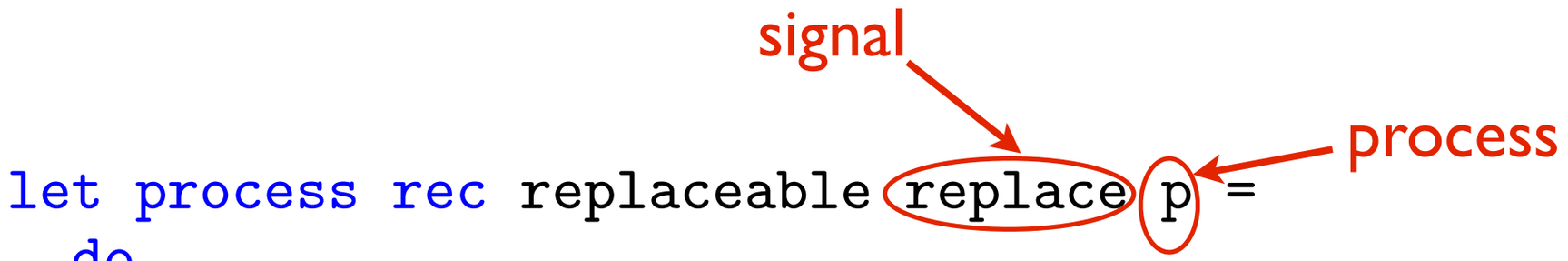
Example of a recursive higher-order process

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

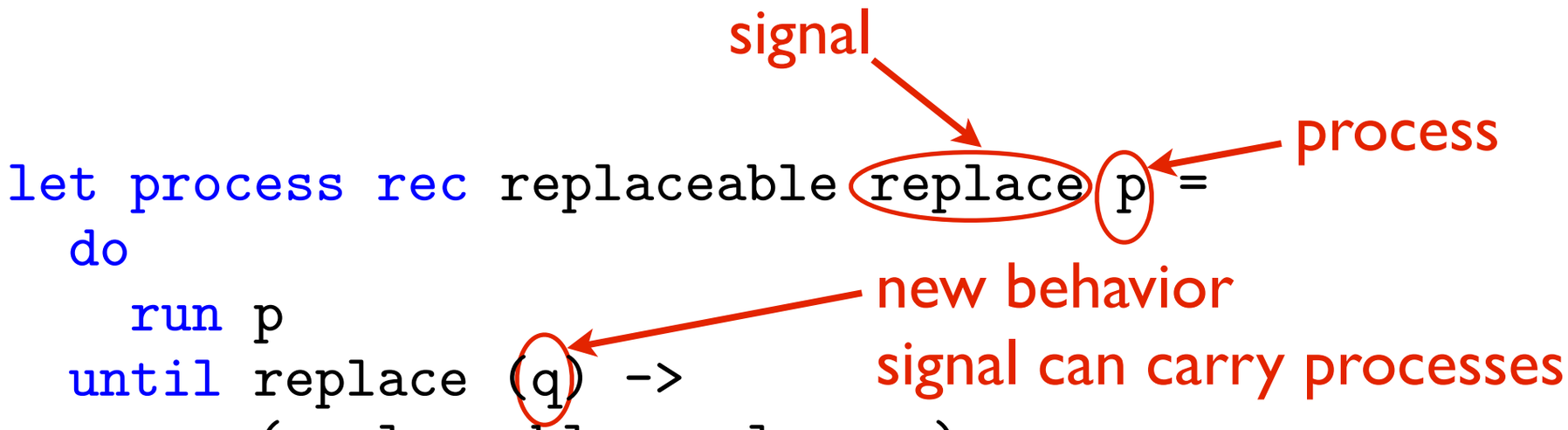
Example of a recursive higher-order process




```
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
```

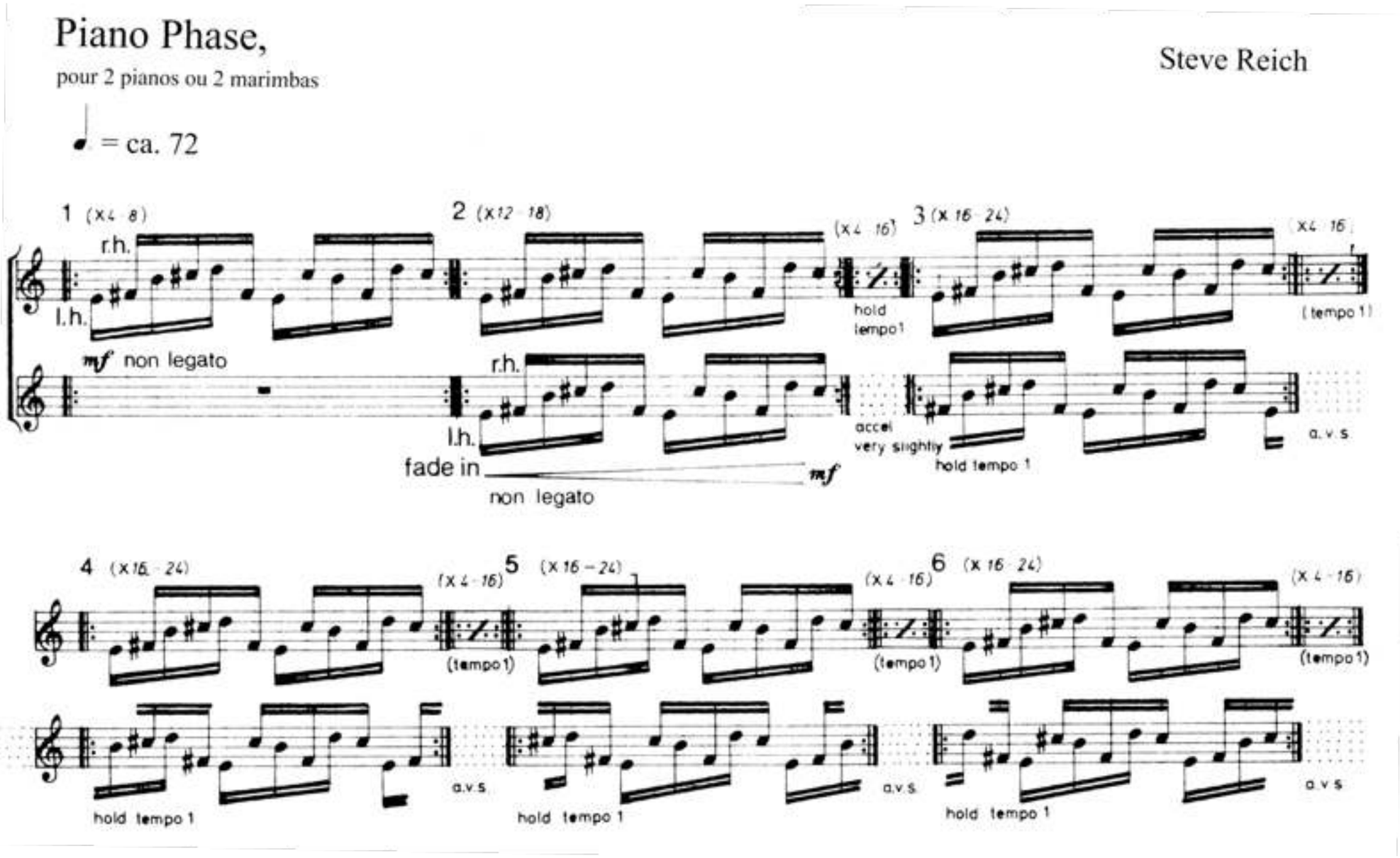
New Reactive Behaviors

Example: Steve Reich's Piano Phase

Piano Phase ...

Bob

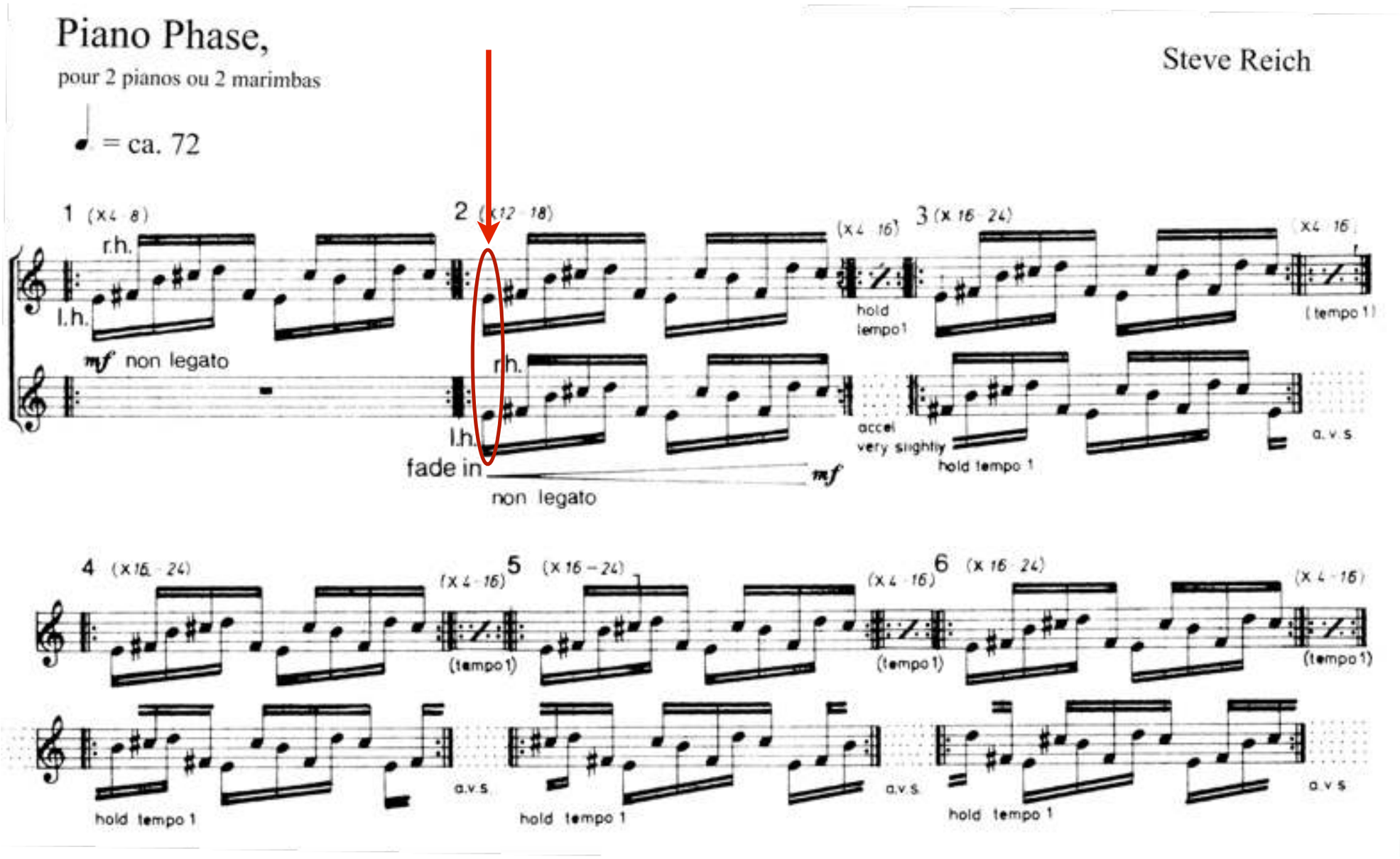
Alice



Piano Phase ...

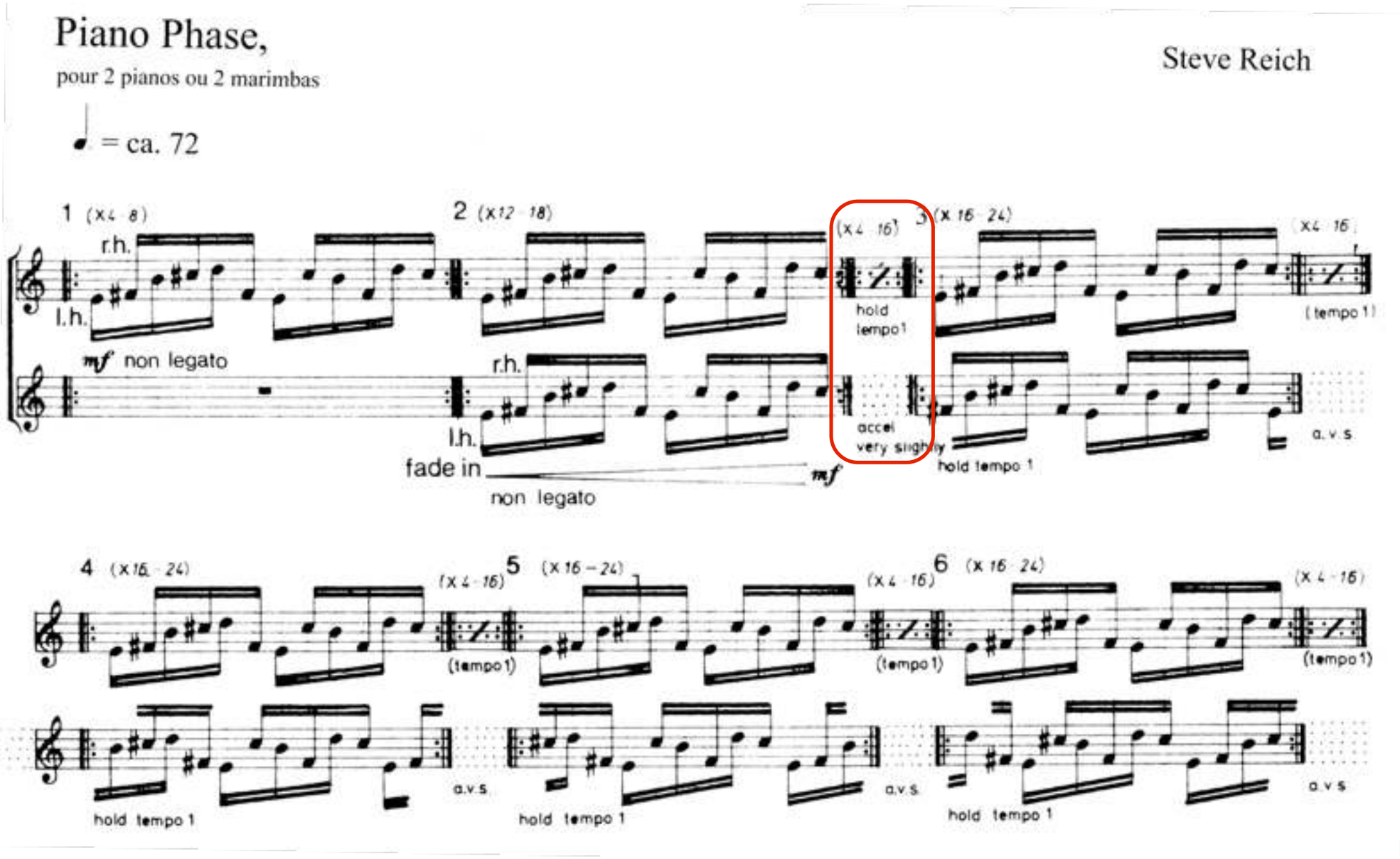
Bob

Alice



Synchronization

Piano Phase ...



Bob

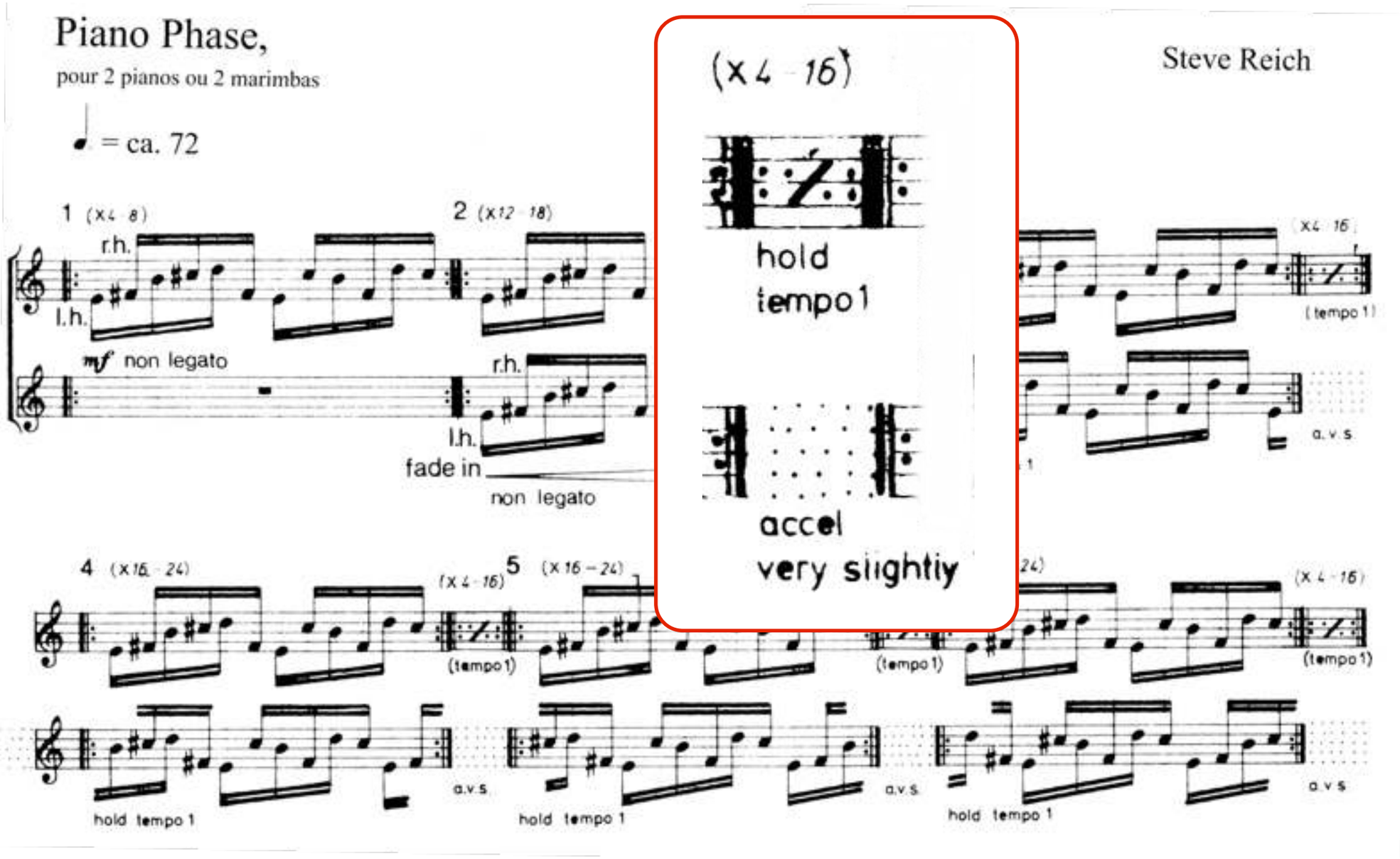
Alice

Desynchronization

Piano Phase ...

Bob

Alice

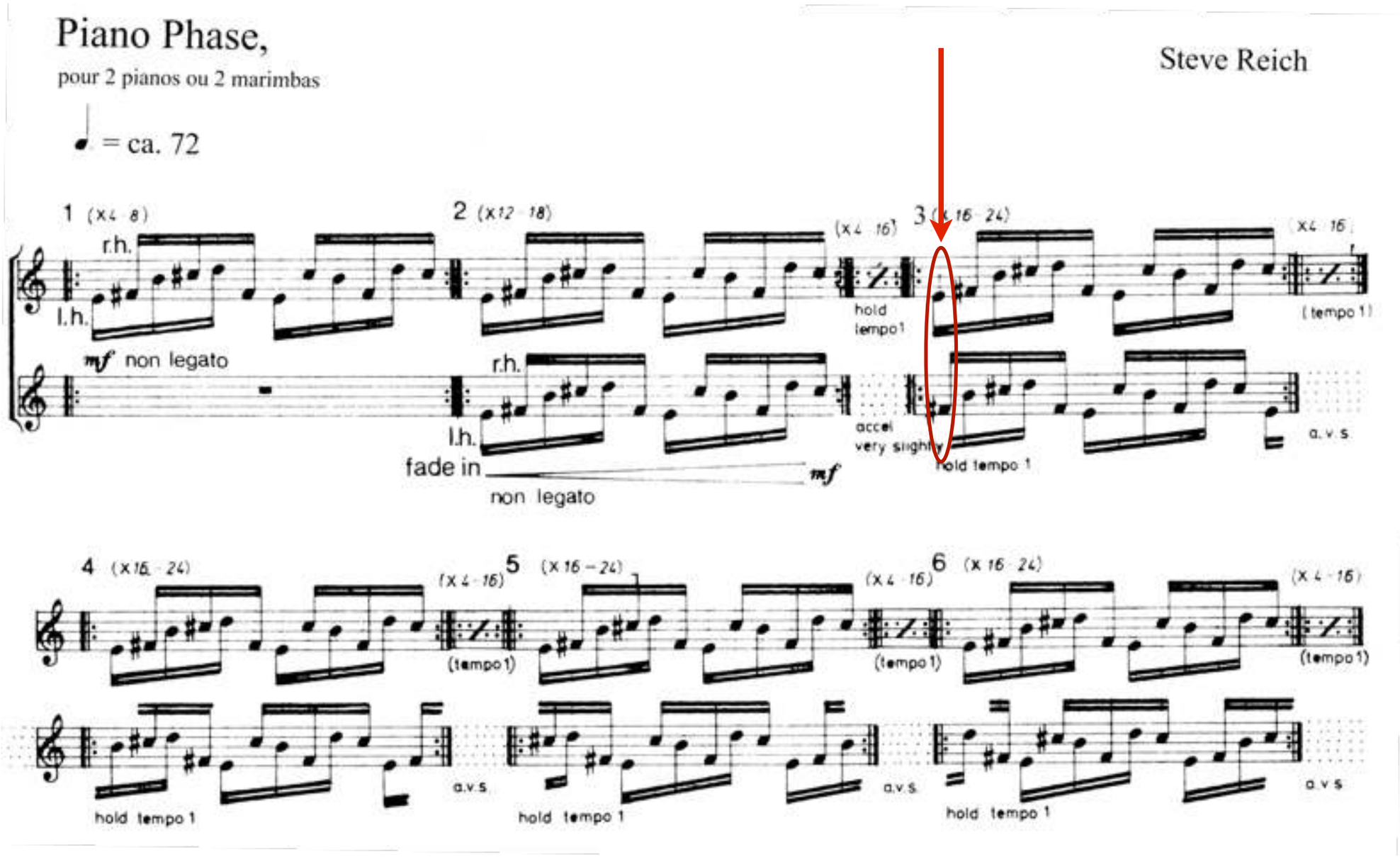


Desynchronization

Piano Phase ...

Bob

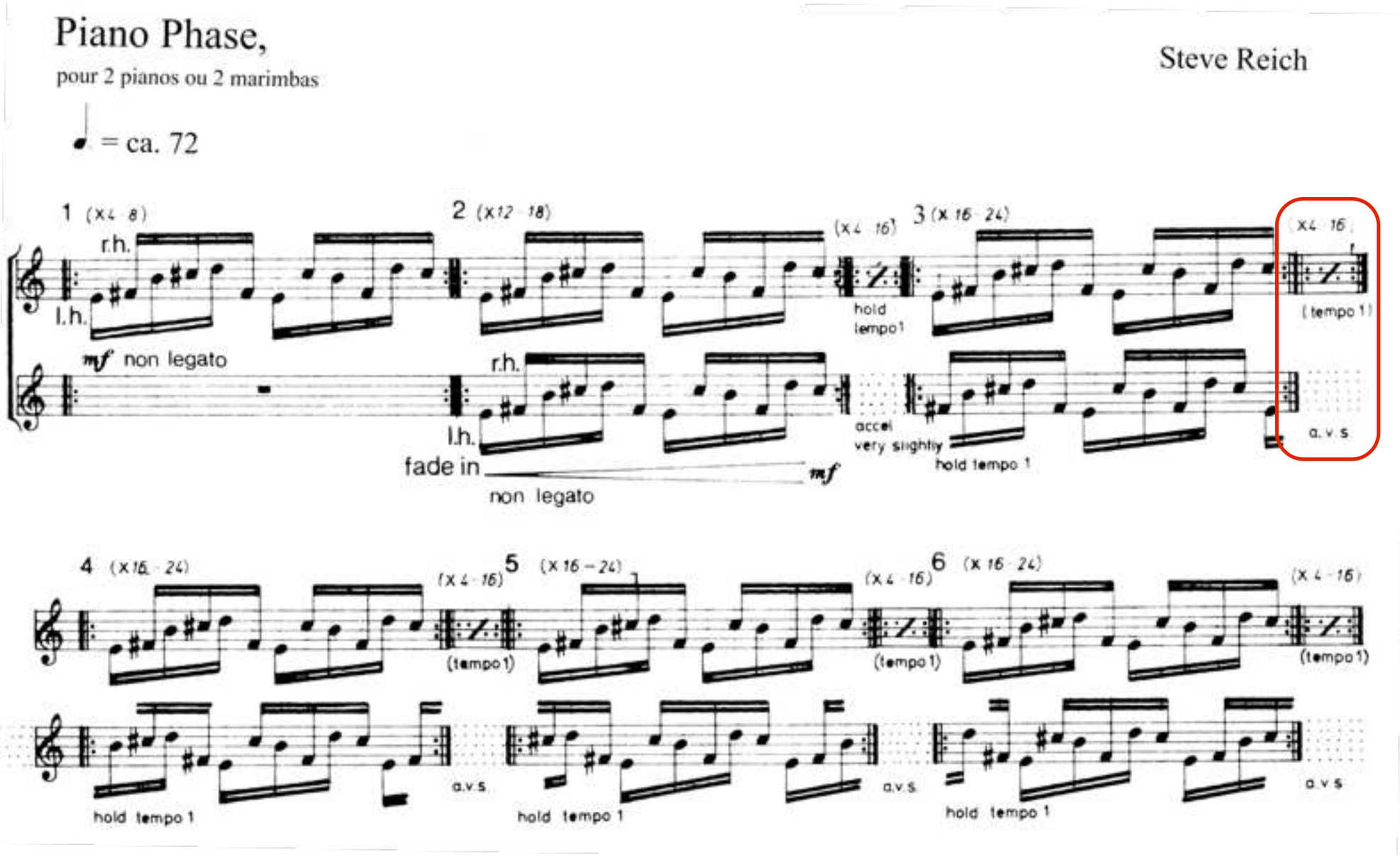
Alice



Piano Phase ...

Bob

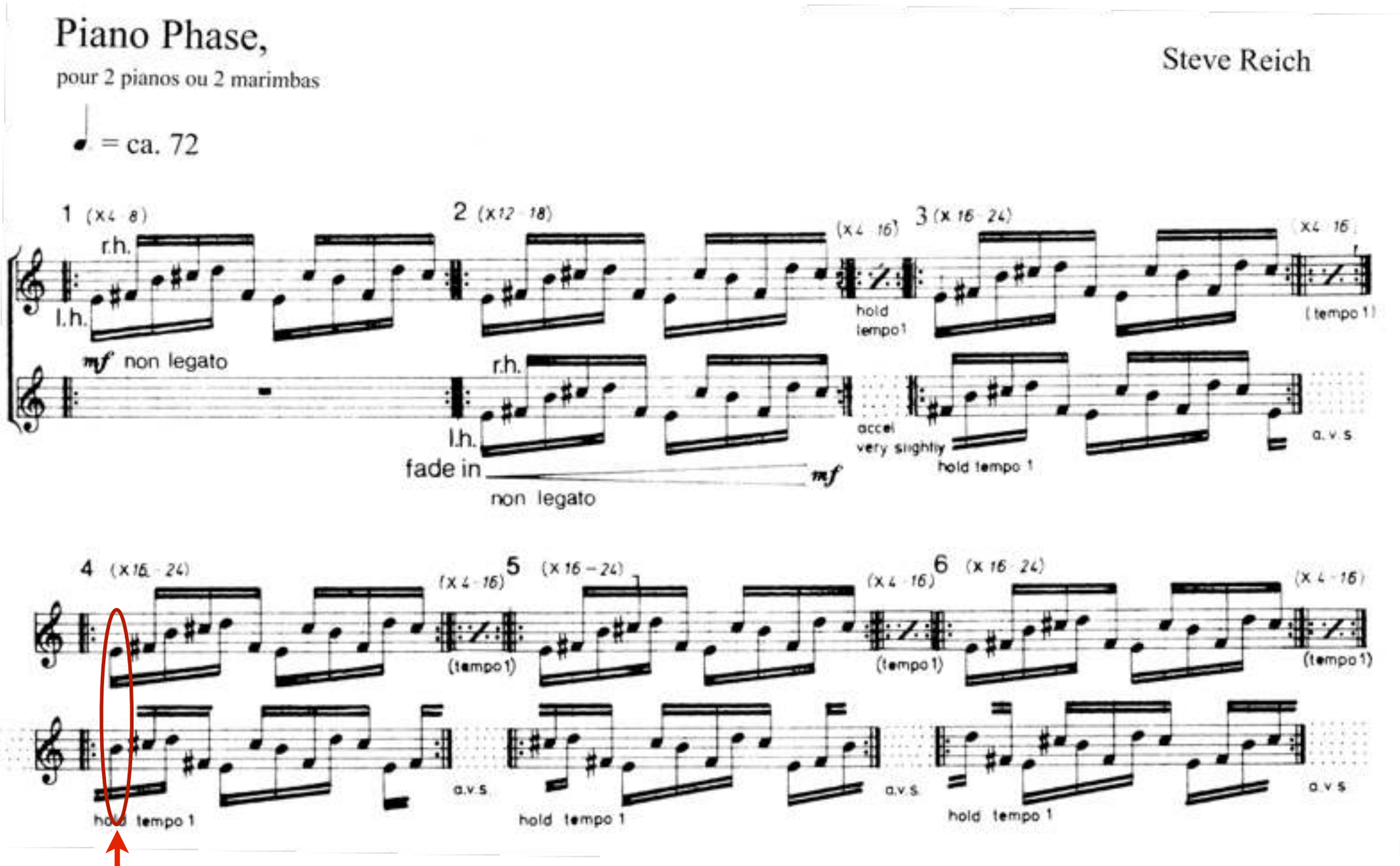
Alice



Piano Phase ...

Bob

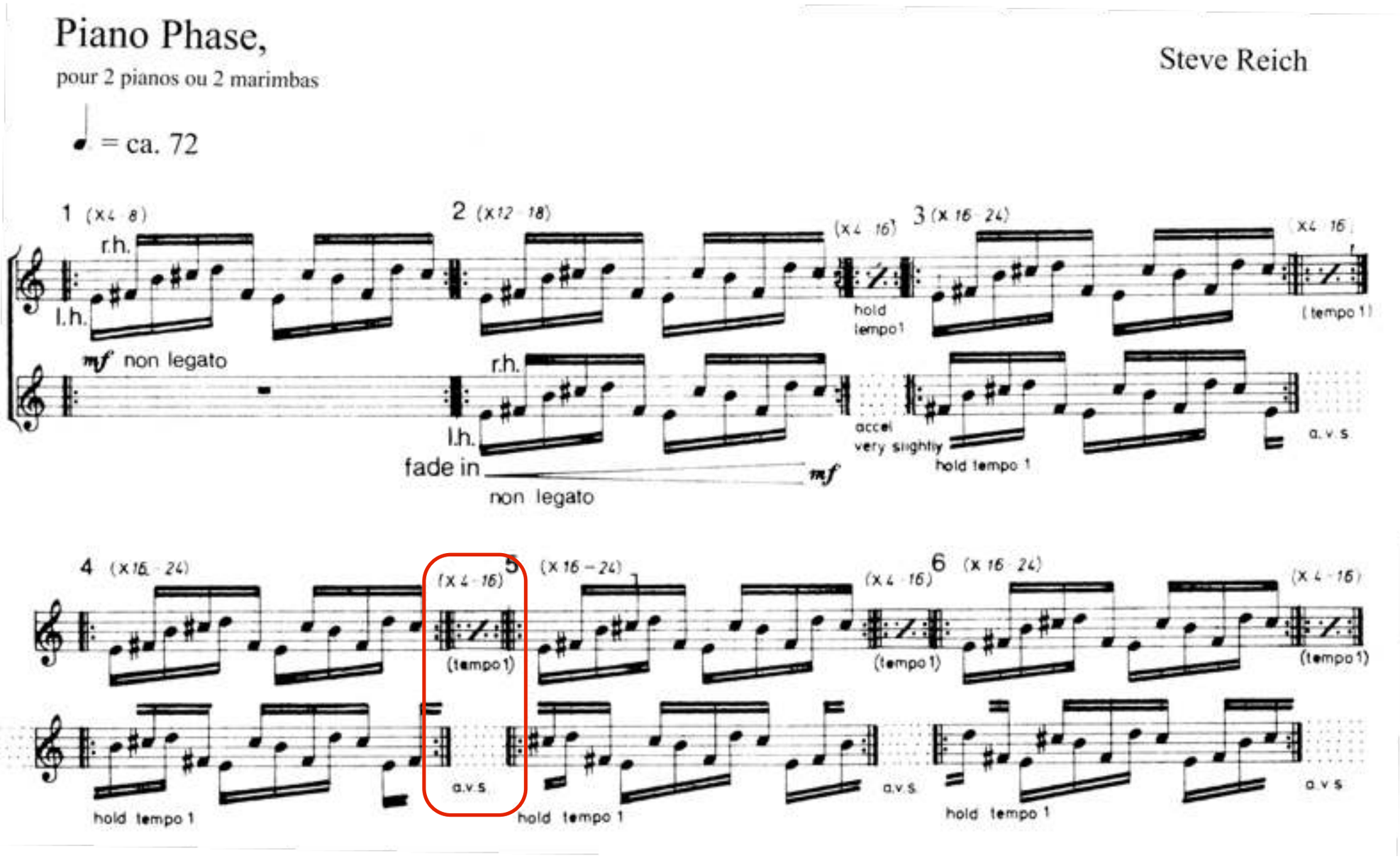
Alice



Piano Phase ...

Bob

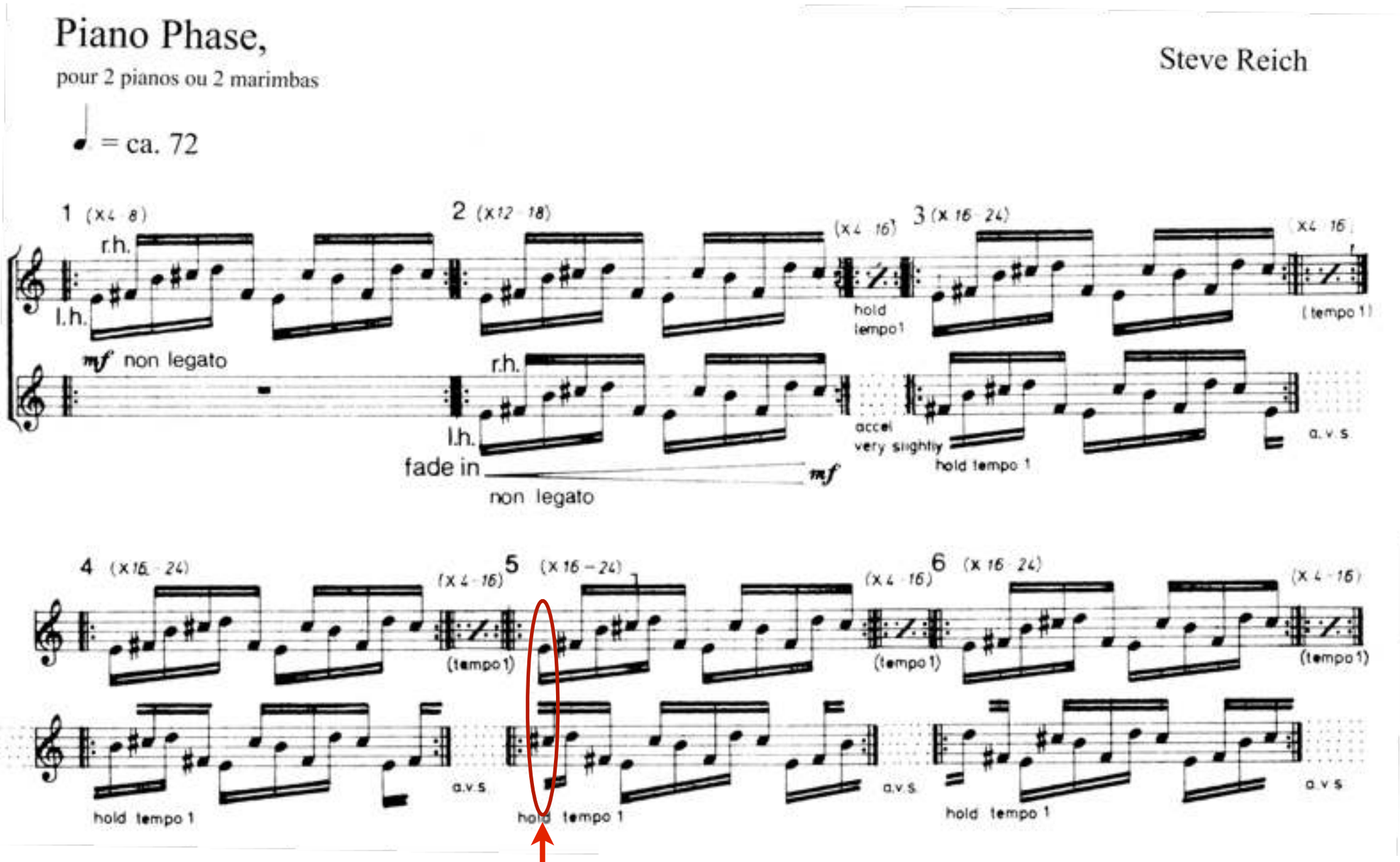
Alice



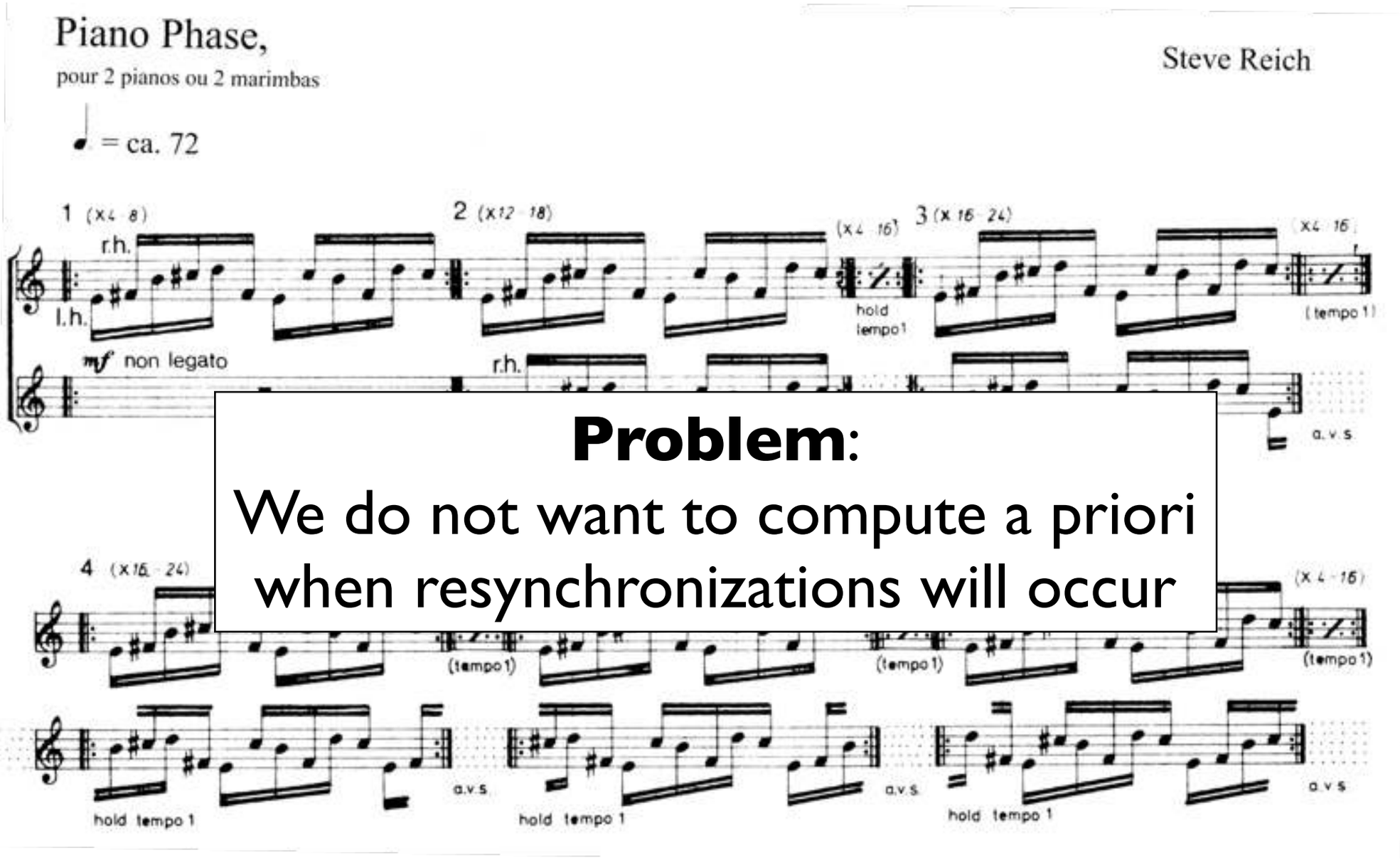
Piano Phase ...

Bob

Alice



Piano Phase ...



Bob

Alice

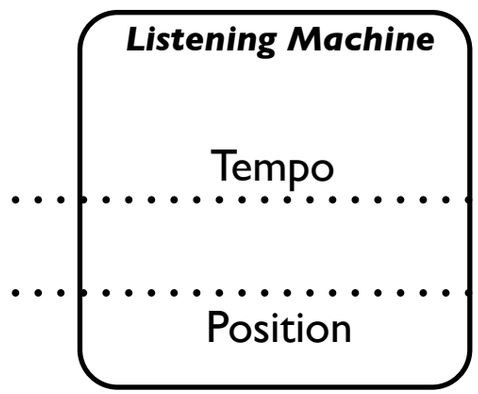
Problem:

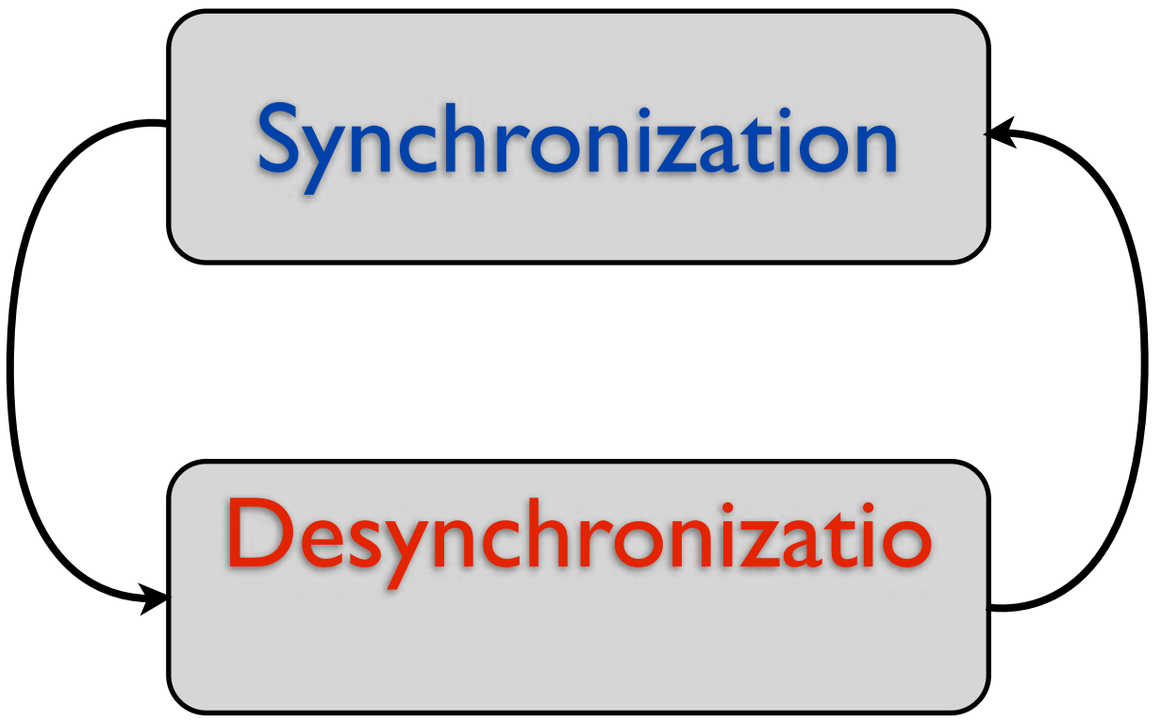
We do not want to compute a priori when resynchronizations will occur

… in Mixed Music

Live musician
Plays the constant speed part

Electronic
Handles the desynchronization





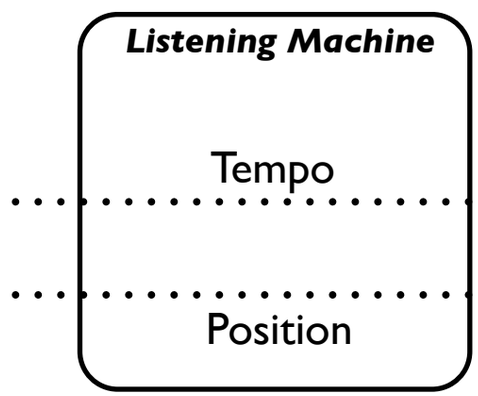
Bob

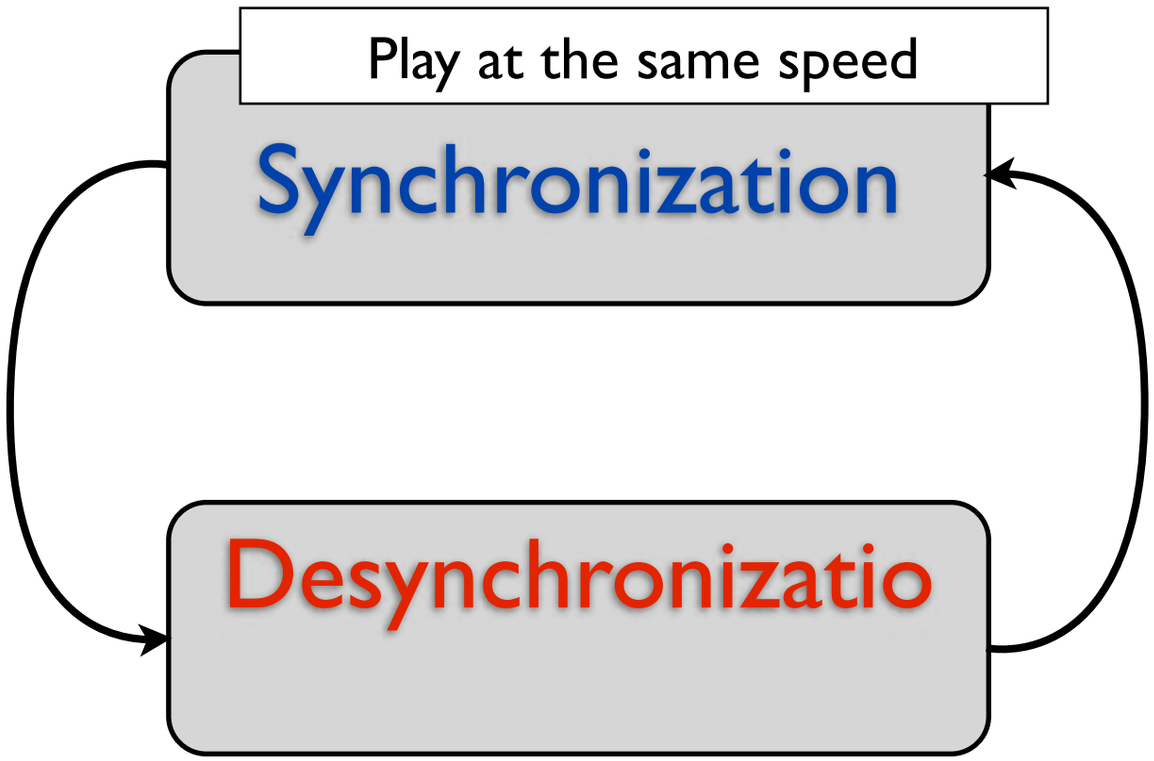
Alice

… in Mixed Music

Live musician
Plays the constant speed part

Electronic
Handles the desynchronization





Bob

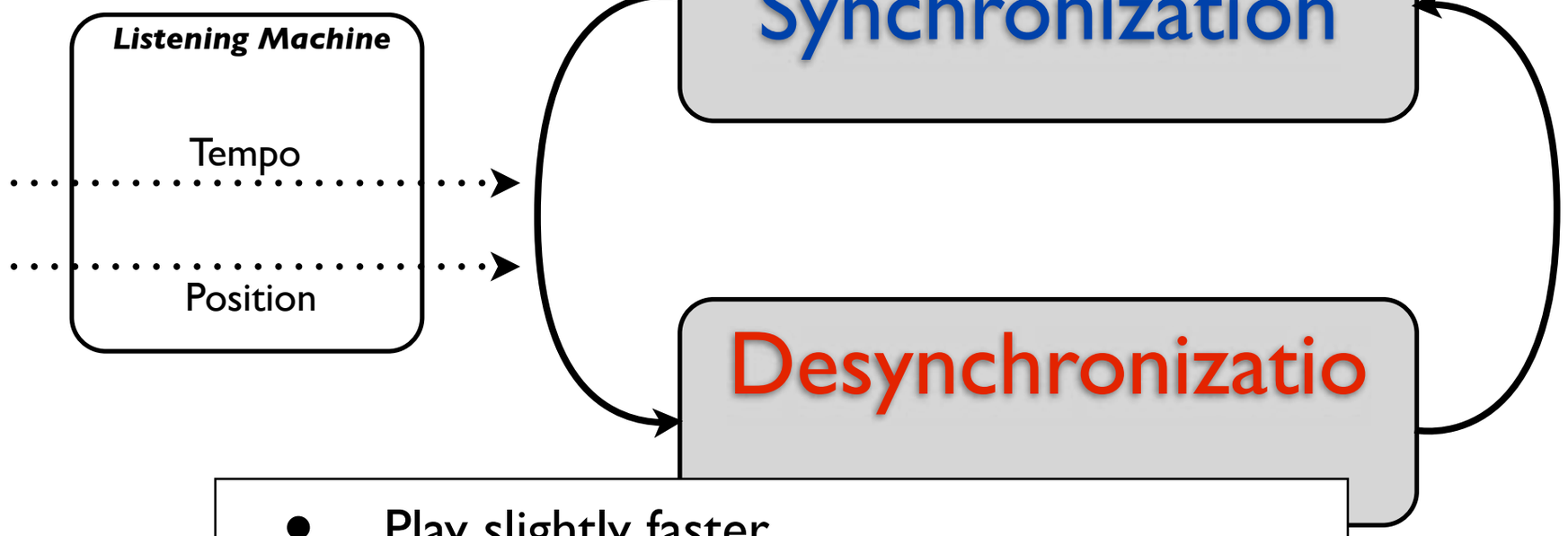
Alice

… 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
```

Conclusion

- **Link with the synchronous model** [EMSOFT 2013]
  - An executable semantics for Antescofo
  - A sequencer efficient enough w.r.t. human ear
  - Embedding in a synchronous reactive language
- **Applications** [FARM 2013]
  - Live coding
  - Prototyping new features:
    new attributes, reactive behaviors, ...

To continue...

http://reactiveml.org/reactive_asco

References

[FARM'13] G. Baudart, L. Mandel, and M. Pouzet. Programming mixed music in ReactiveML. In Workshop on Functional Art, Music, Modeling and Design, 2013.

[EMSOFT'13] G. Baudart, F. Jacquemard, L. Mandel, and M. Pouzet. A synchronous embedding of Antescofo, a domain-specific language for interactive mixed music. In Conference on Embedded Software, 2013.

[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.

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

[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.

[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.