Antescofo

Synchronous Interpretations of a Language for Mixed Music

Supervisors

F. Jacquemard : *MuSync, INRIA*

M. Pouzet : *PARKAS, ENS*

Guillaume BAUDART

SYNCHRON'12

Objectives

- Study and formalization of a language for mixed music: the language of Antescofo
- Link with synchronous languages

Antescofo

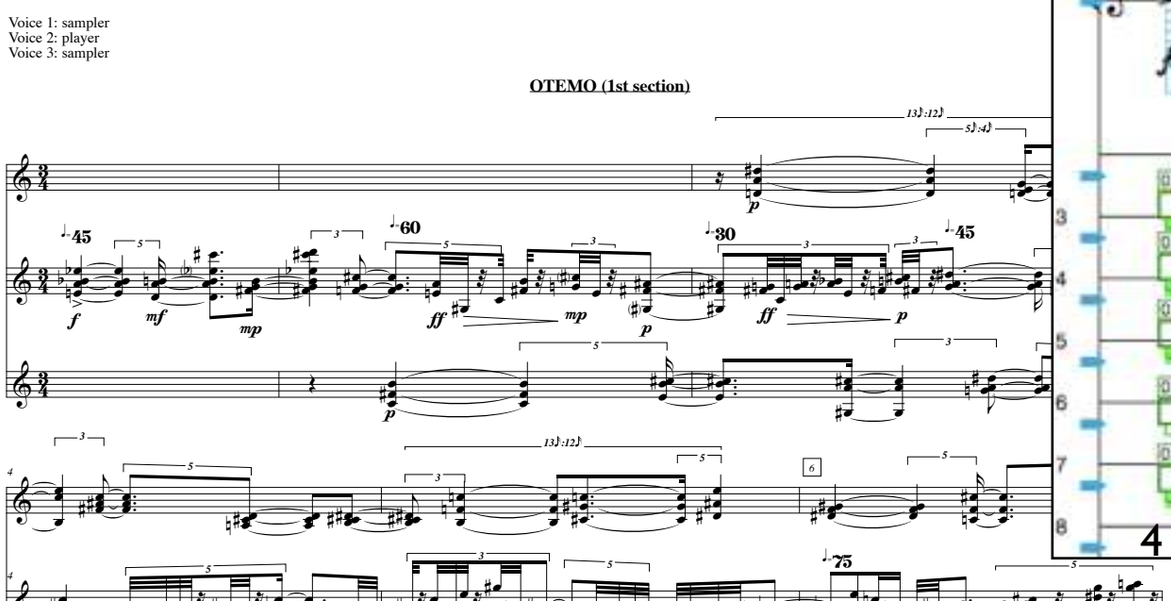
[Cont 2010]

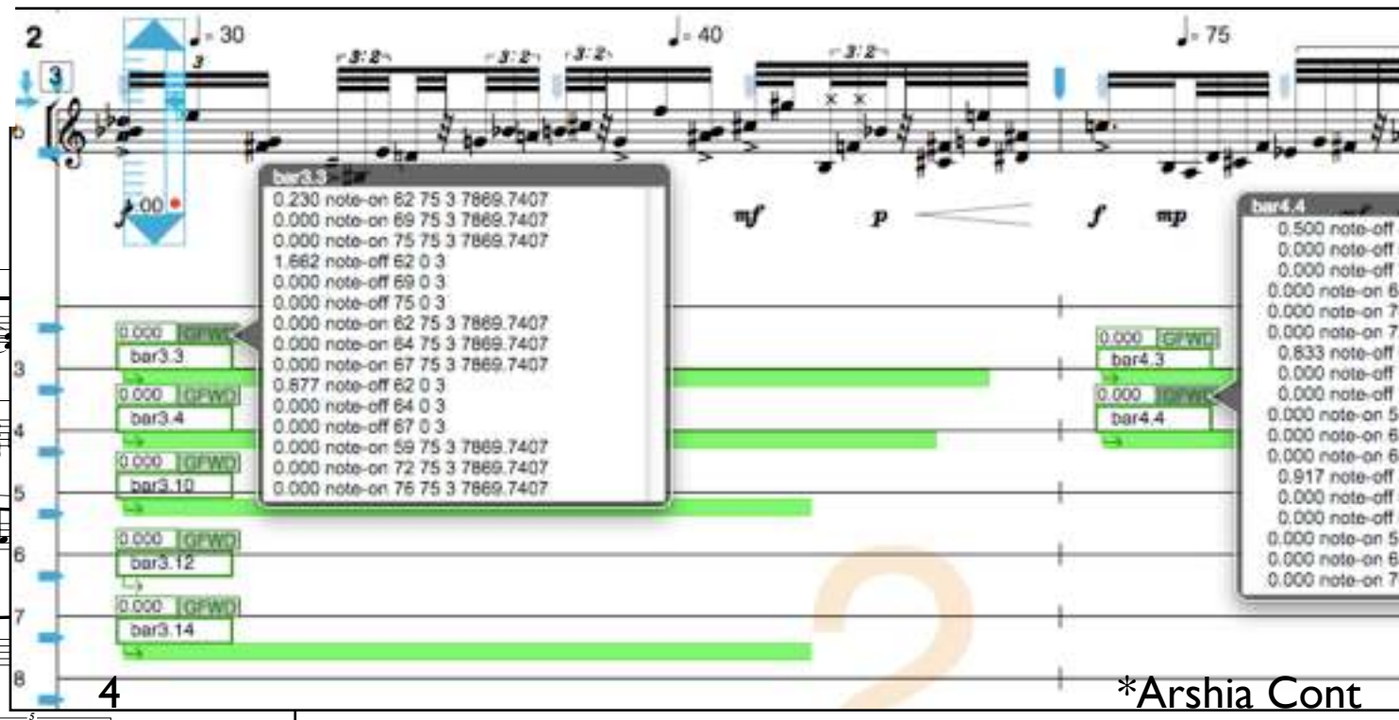
- Score Follower: Position, Tempo
- Reactive Sequencer:
  Automatic accompaniment

Mixed Music

Performance/Real Time

Composition/Program

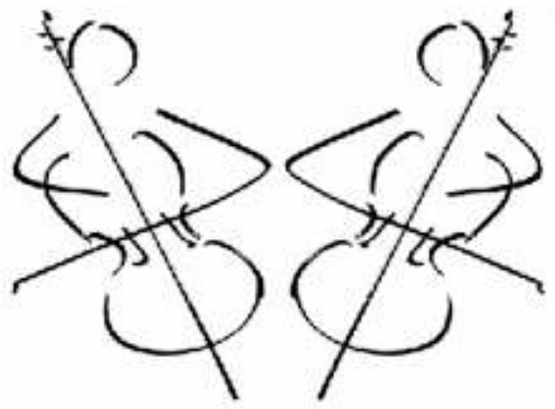




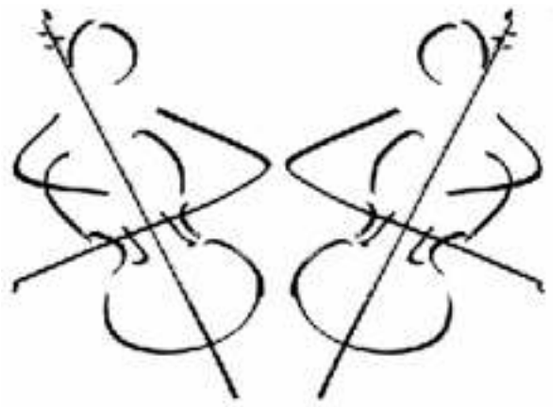
4

*Arshia Cont

Antescofo Architecture



Antescofo Architecture



signal

Max/MSP

Control Messages

Listening Machine

bpm

event

Sequencer

Reactive System!

Partition antescofo

Pre-treatment

Real Time Communication

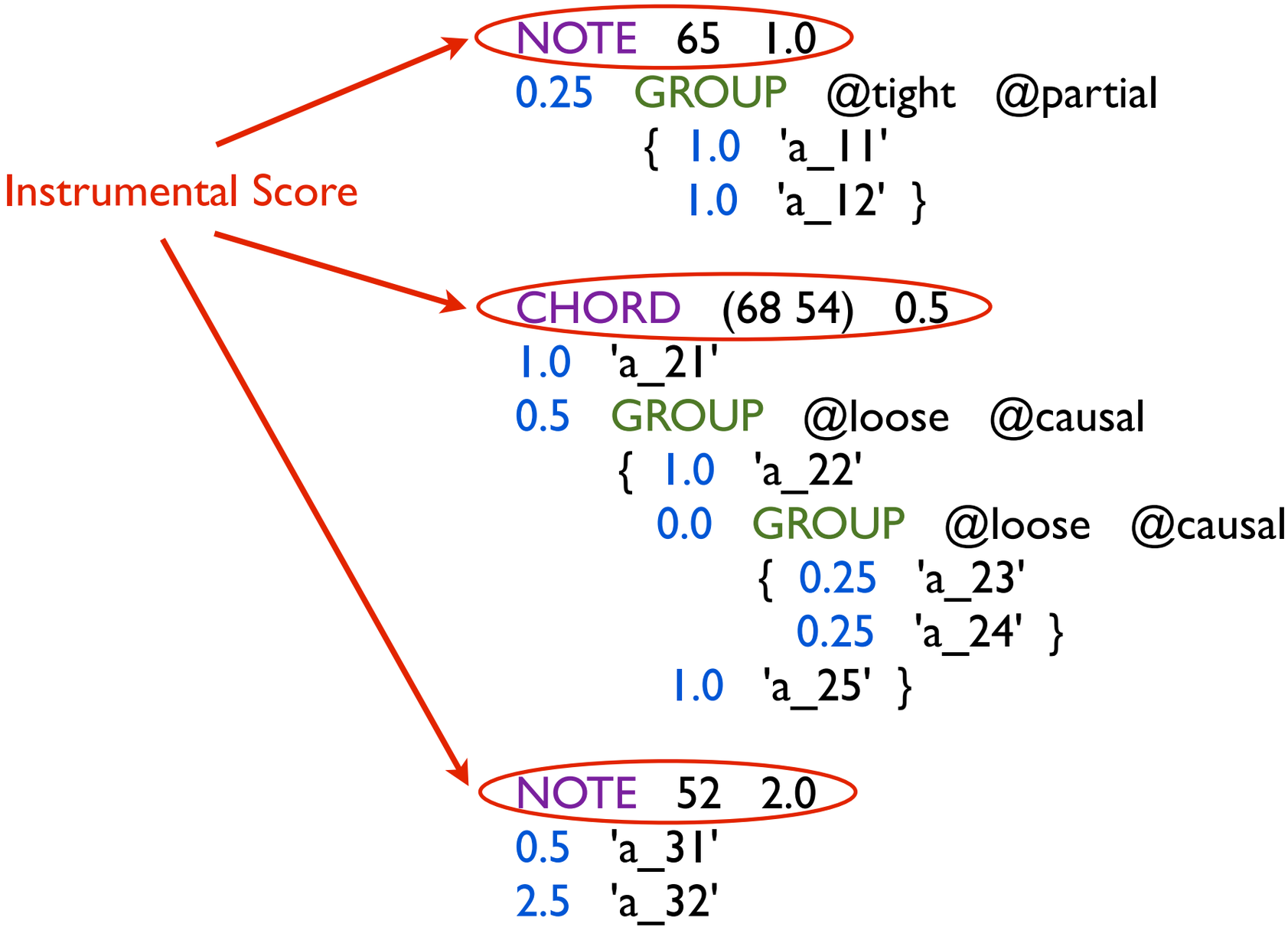
The Language of Antescofo

Goal: Jointly specify electronic and instrumental parts

```
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 Language of Antescofo

Goal: Jointly specify electronic and instrumental parts



The Language of Antescofo

Goal: Jointly specify electronic and instrumental parts

```
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 Language of Antescofo

Goal: Jointly specify electronic and instrumental parts

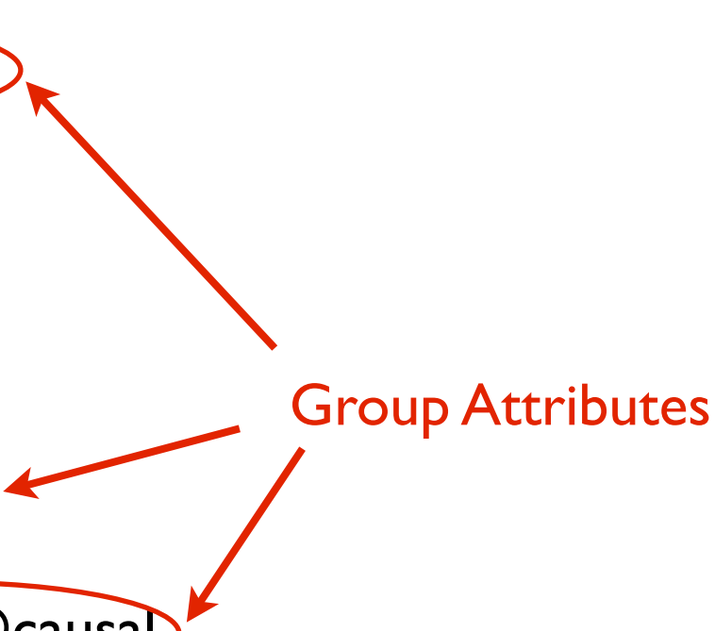
```
NOTE   65   1.0
0.25   GROUP   @tight   @partial
          {  1.0   'a_11'
             1.0   'a_12'  }

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 Language of Antescofo

Goal: Jointly specify electronic and instrumental parts

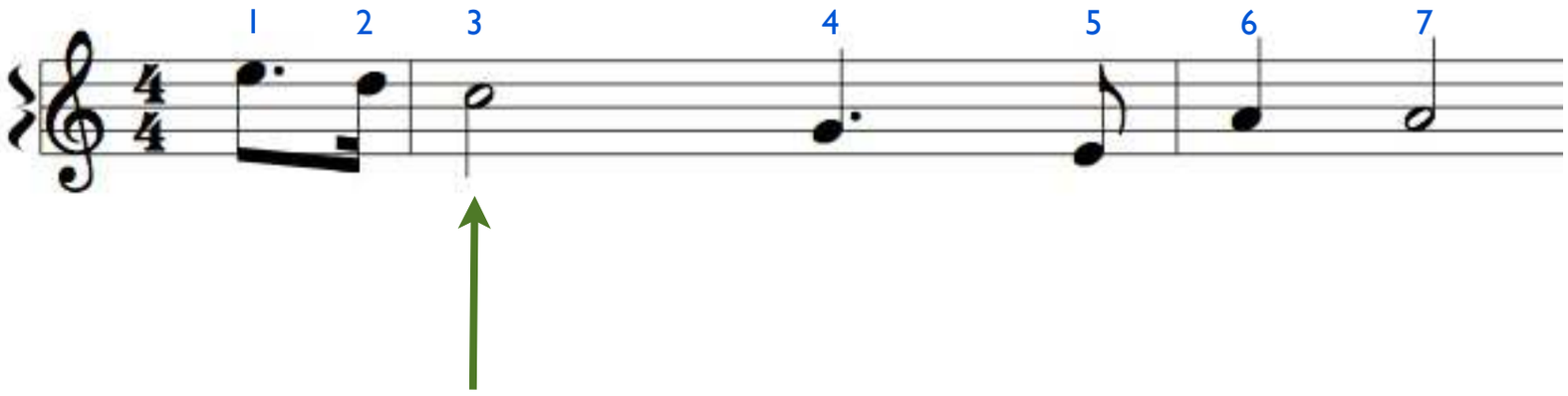
```
NOTE   65   1.0
0.25   GROUP   @tight   @partial
          {  1.0   'a_11'
             1.0   'a_12'  }

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'
```



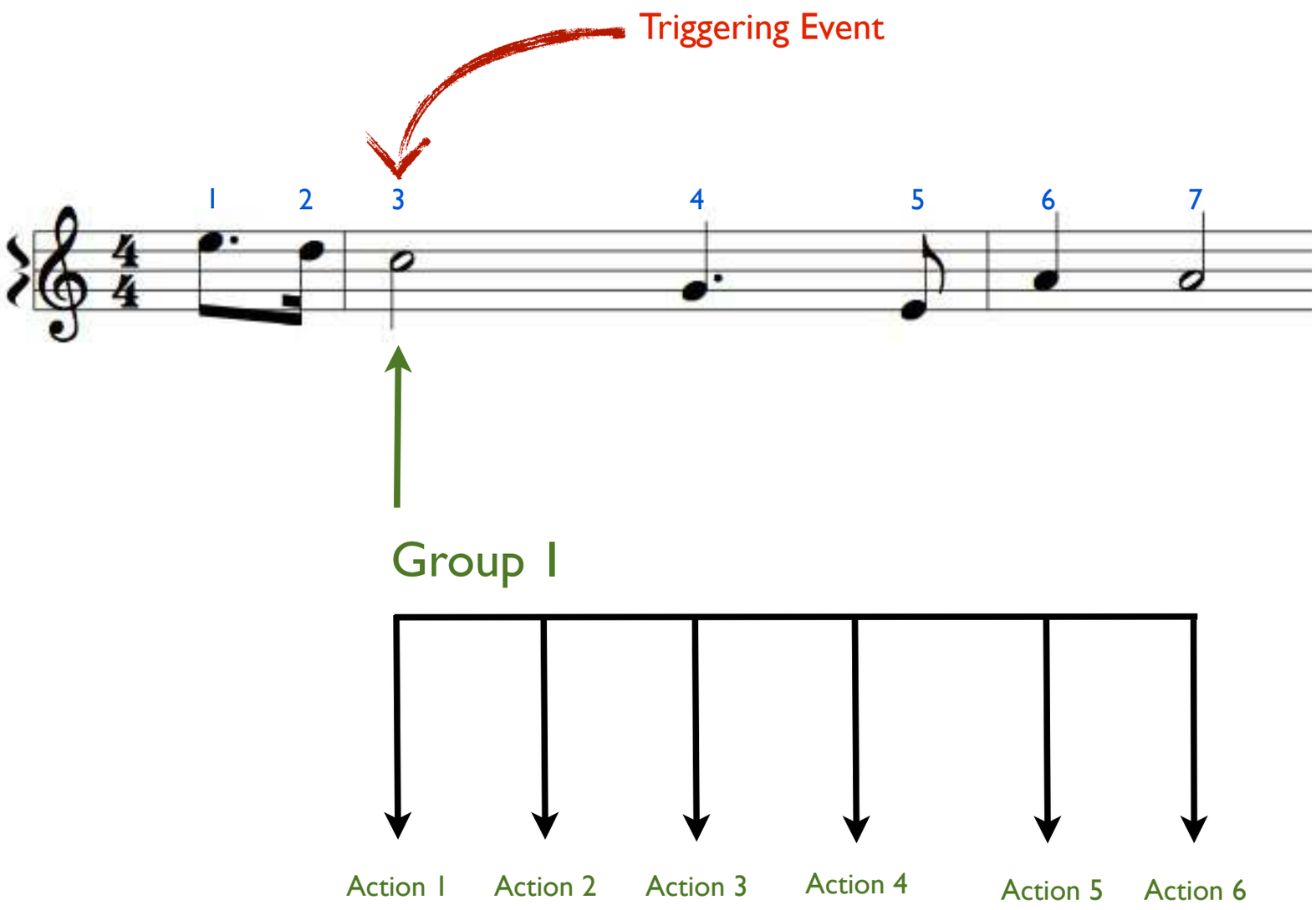
Language Objectives

- Specify a ‘critical’ reactive system
- Time programming
- Real time synchronization
- Error handling strategies

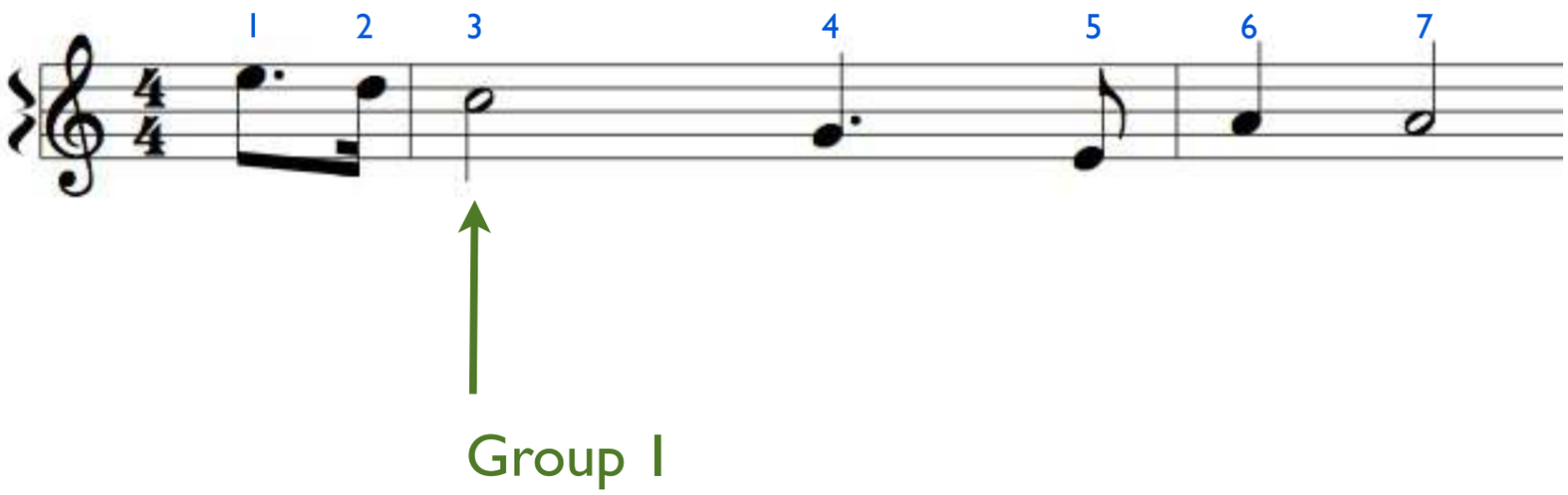
Synchronization Strategies



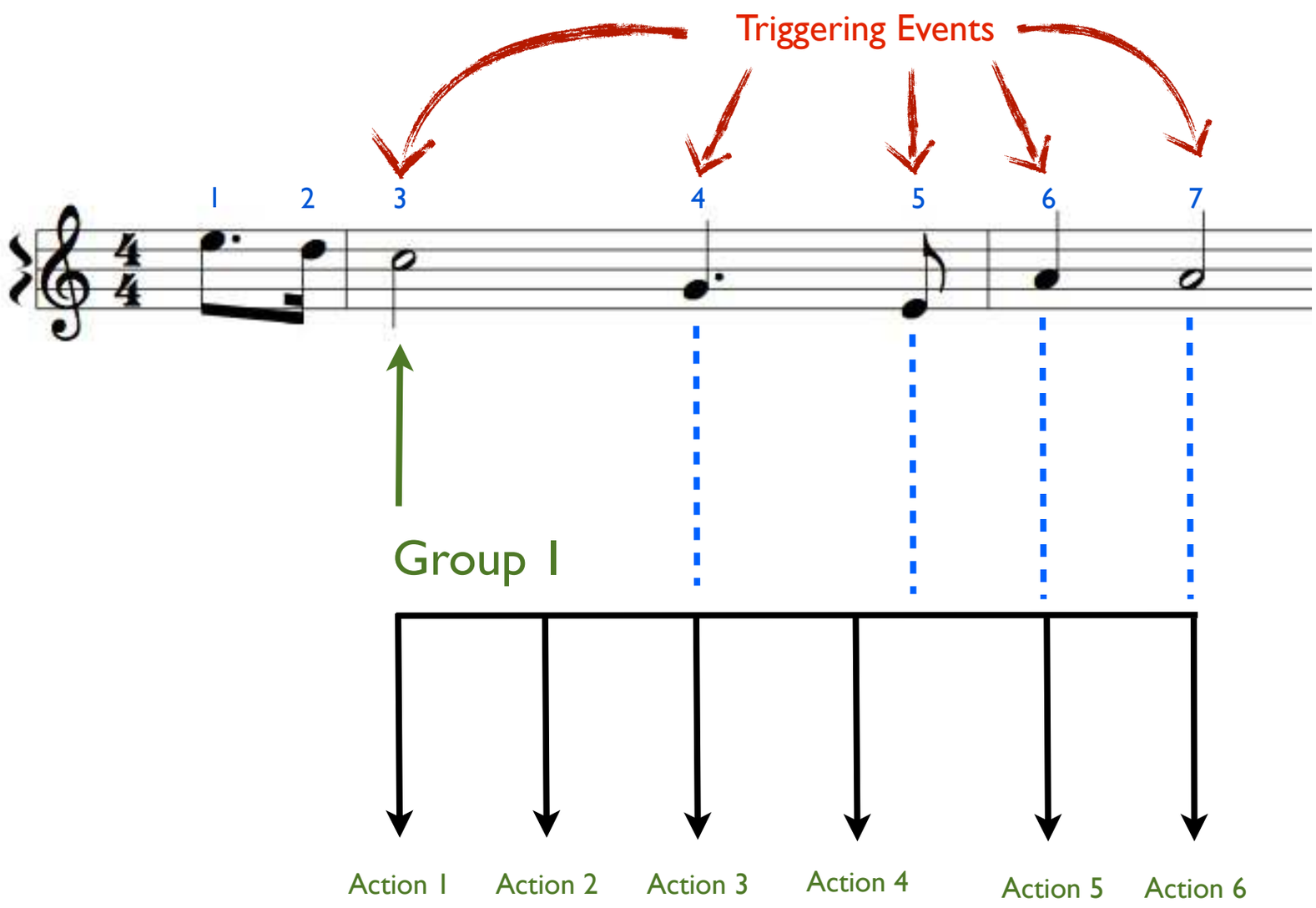
Loose: Synchronization with the tempo stream.



Loose: Synchronization with the tempo stream.

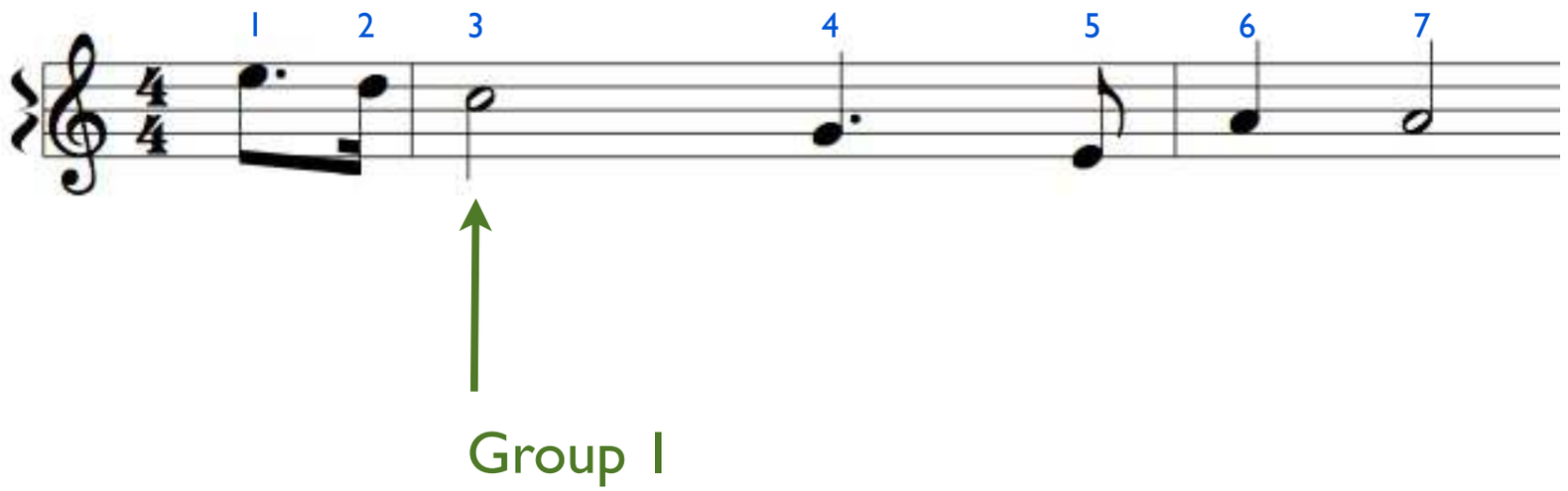


Tight: Synchronization
with tempo and events stream.

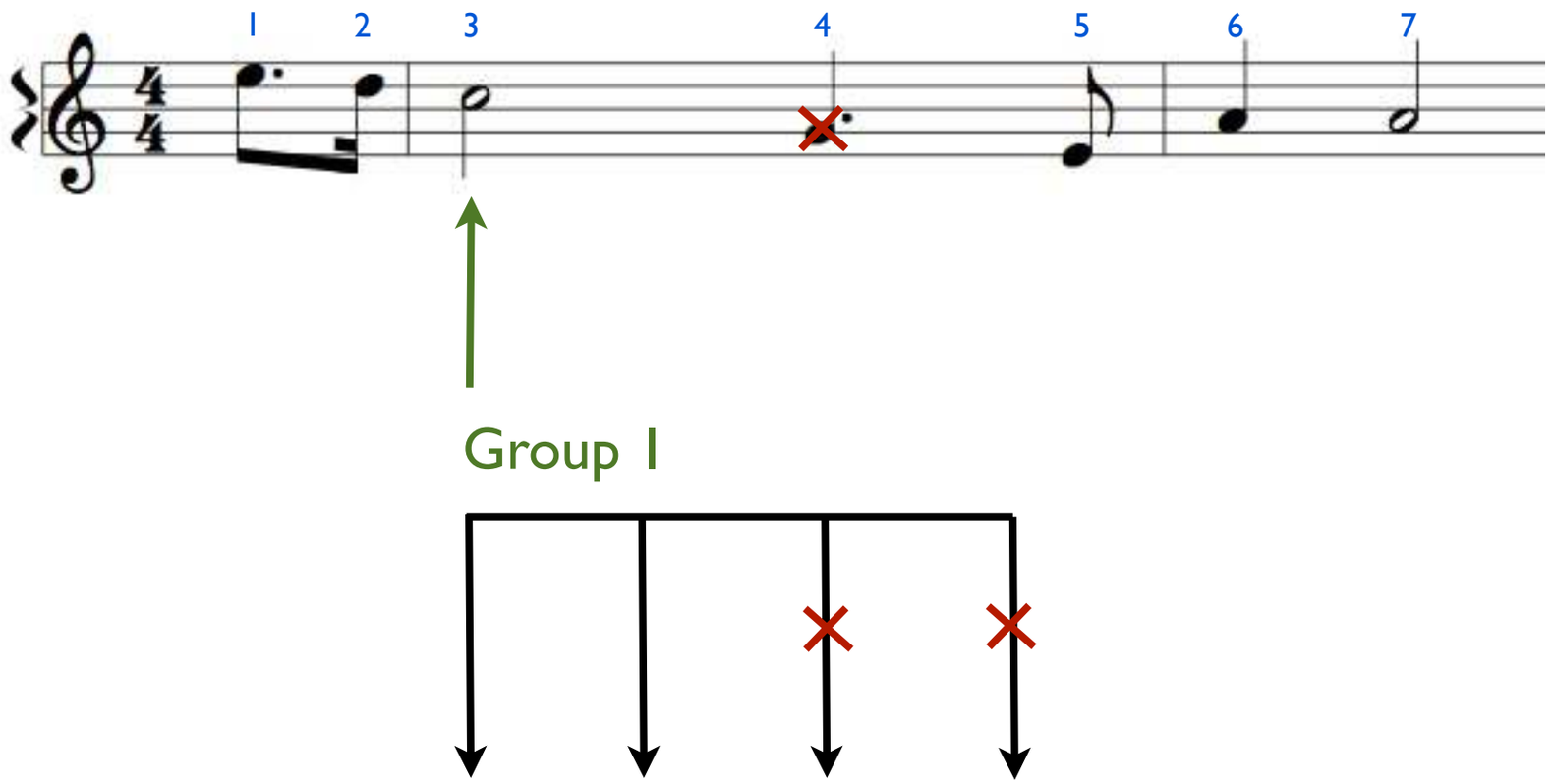


Tight: Synchronization
with tempo and events stream.

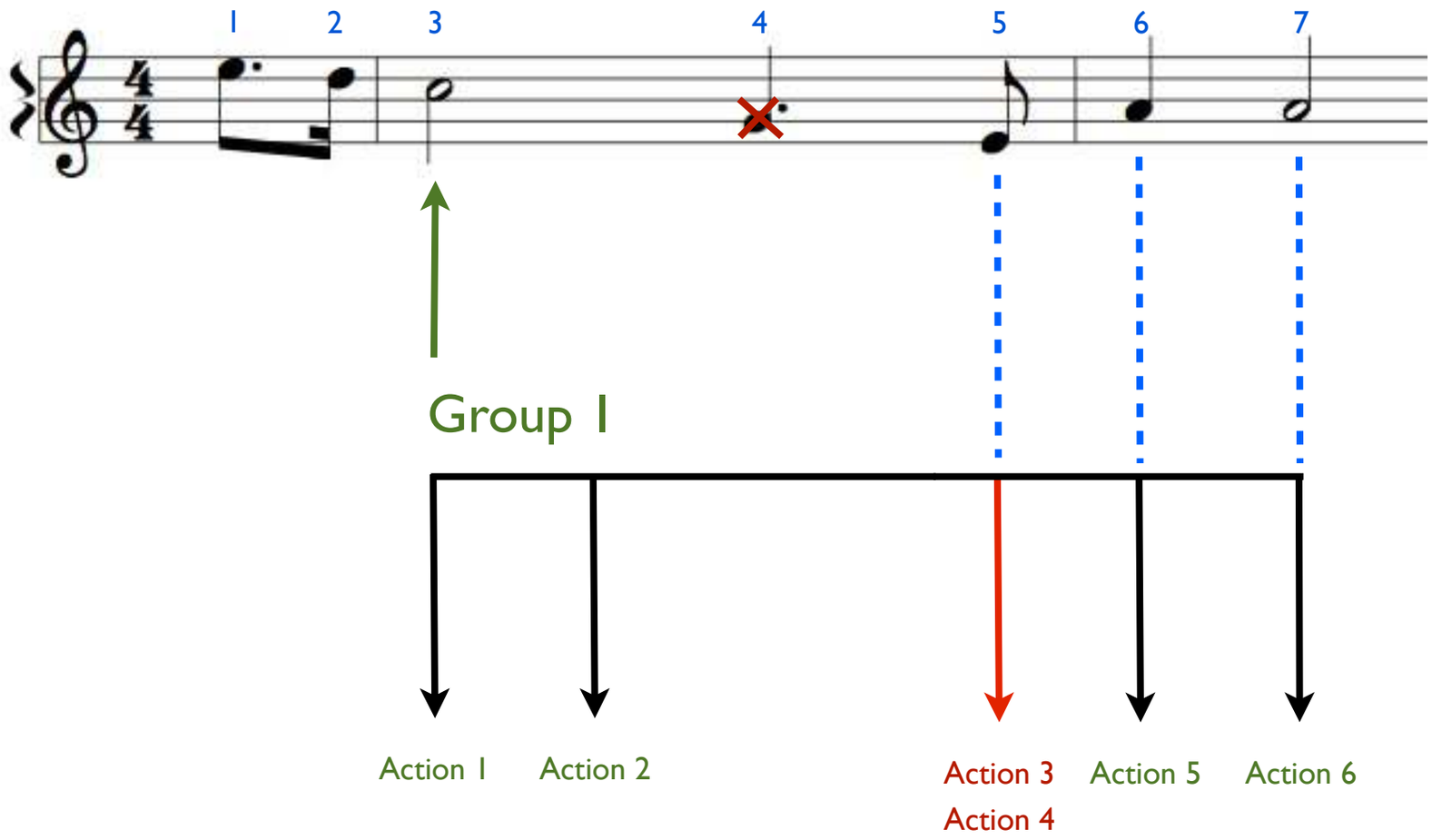
Error Handling Strategies



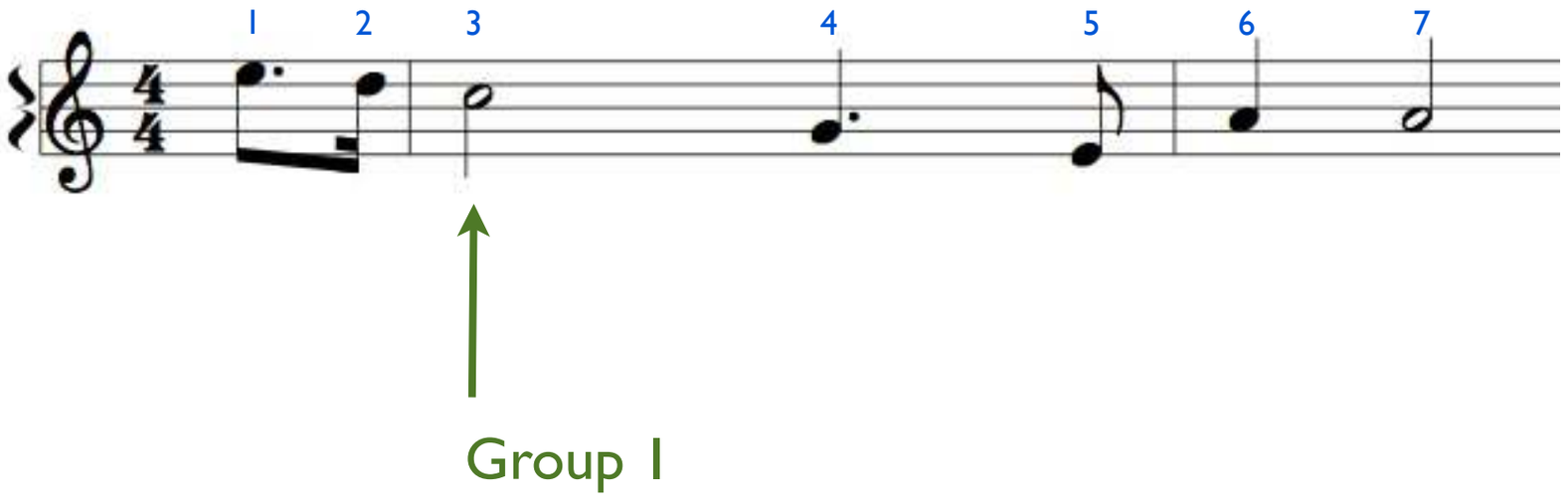
Causal: Actions should be launch immediately when the system recognize the absence of the triggering event



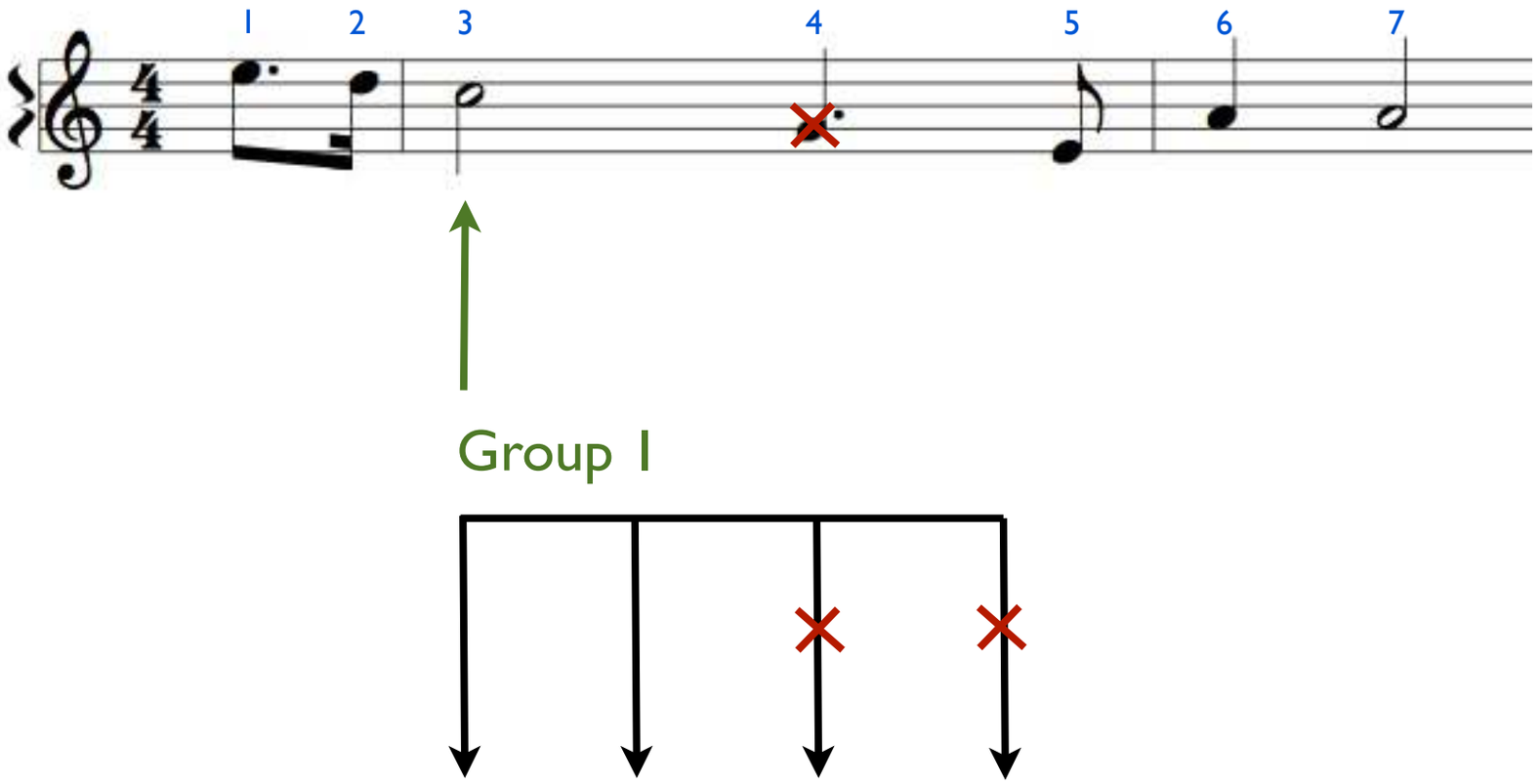
Causal: Actions should be launch immediately when the system recognize the absence of the triggering event



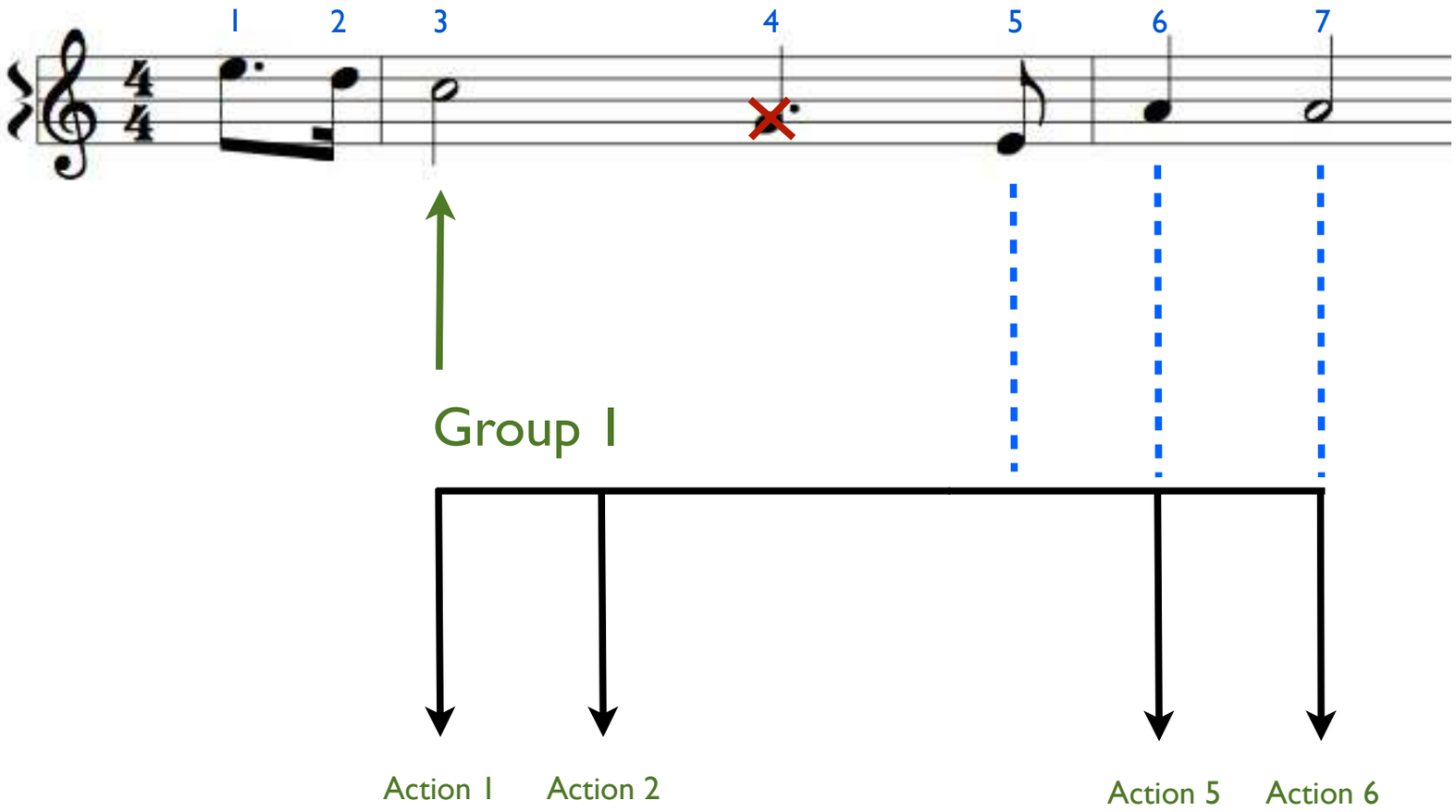
Causal: Actions should be launch immediately when the system recognize the absence of the triggering event



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

Model

$$
\begin{aligned}
a &::= A(\delta, m) \\
g &::= G(\delta, sync, err, ae^*) \\
ae &::= a|g \\
sync &::= T|L \\
err &::= P|C \\
\delta &::= delay \\
m &::= message \\
i &::= int \\
se &::= (i : ae) \\
sc &::= se^* \\
p &::= (i, \delta, m)^*
\end{aligned}
$$

Model

$$
\begin{array}{rcl}
a & ::= & A(\delta, m) \\
g & ::= & G(\delta, sync, err, ae^*) \\
ae & ::= & a|g \\
sync & ::= & T|L \\
err & ::= & P|C \\
\delta & ::= & delay \\
m & ::= & message \\
i & ::= & int \\
se & ::= & (i : ae) \\
sc & ::= & se^* \\
p & ::= & (i, \delta, m)^*
\end{array}
$$

Dating $\mathcal{E} : int \rightarrow delay$

Detections $D : int^*$

Errors $M : int \rightarrow int$

Model

$$
\begin{array}{rcl}
a & ::= & A(\delta, m) \\
g & ::= & G(\delta, sync, err, ae^*) \\
ae & ::= & a|g \\
sync & ::= & T|L \\
err & ::= & P|C \\
\delta & ::= & delay \\
m & ::= & message \\
i & ::= & int \\
se & ::= & (i : ae) \\
sc & ::= & se^* \\
p & ::= & (i, \delta, m)^*
\end{array}
$$

Dating $\mathcal{E} : int \rightarrow delay$

Detections $D : int^*$

Errors $M : int \rightarrow int$

Objective

$$D, M, sc \vdash^{\text{exec}} p$$

Execution Rules

$$D, M, sc \vdash^{\text{exec}} p$$

Execute a score

$$D, M, sc \vdash^{\text{detect}}_{\delta,\, i} a \to p$$

Execute an action a
attached to a detected event i
with a delay δ

$$D, M, sc \vdash^{\text{missed}}_{\delta,\, i} a \to p$$

Execute an action a
attached to a missed event i
with a delay δ

Execution

EMPTY SCORE

$$\frac{}{D, M, \varnothing \vdash^{\text{exec}} \varnothing}$$

DETECTED

$$\frac{i \in D \qquad D, M, sc \vdash^{\text{detect}}_{0,\, i} ae \to p_1 \qquad D, M, sc \vdash p_2}{D, M, sc \cup (i : ae) \vdash^{\text{exec}} p_1 \cup p_2}$$

MISSED

$$\frac{i \notin D \qquad D, M, sc \vdash^{\text{missed}}_{0,\, i} ae \to p_1 \qquad D, M, sc \vdash p_2}{D, M, sc \cup (i : ae) \vdash^{\text{exec}} p_1 \cup p_2}$$

Execution

DETECTED ACTION

$$D, M, sc \xrightarrow[\delta,\, i]{\text{detect}} A(\delta_a, m) \rightarrow \{(i, \delta + \delta_a, m)\}$$

MISSED ACTION

$$M(i) = j$$

$$D, M, sc \xrightarrow[\delta,\, i]{\text{missed}} A(\delta_a, m) \rightarrow \{j, max(0, \delta + \delta_a + \mathcal{E}(i) - \mathcal{E}(j)), m\}$$

An Interpret in Reactive ML

Reactive ML

[Mandel 2005]

- A Reactive extension to OCaml
- Synchronous constructs (|| ;)
- Communication between processus via signals

Logical Time

```
let process wait_abs dur =
  let d = int_of_float (dur /. clock_step) in
  for i=d downto 1 do pause;
```

- Wait a duration in millisecond
- Compilation in sampling mode
  (clock_step : argument for the compiler)

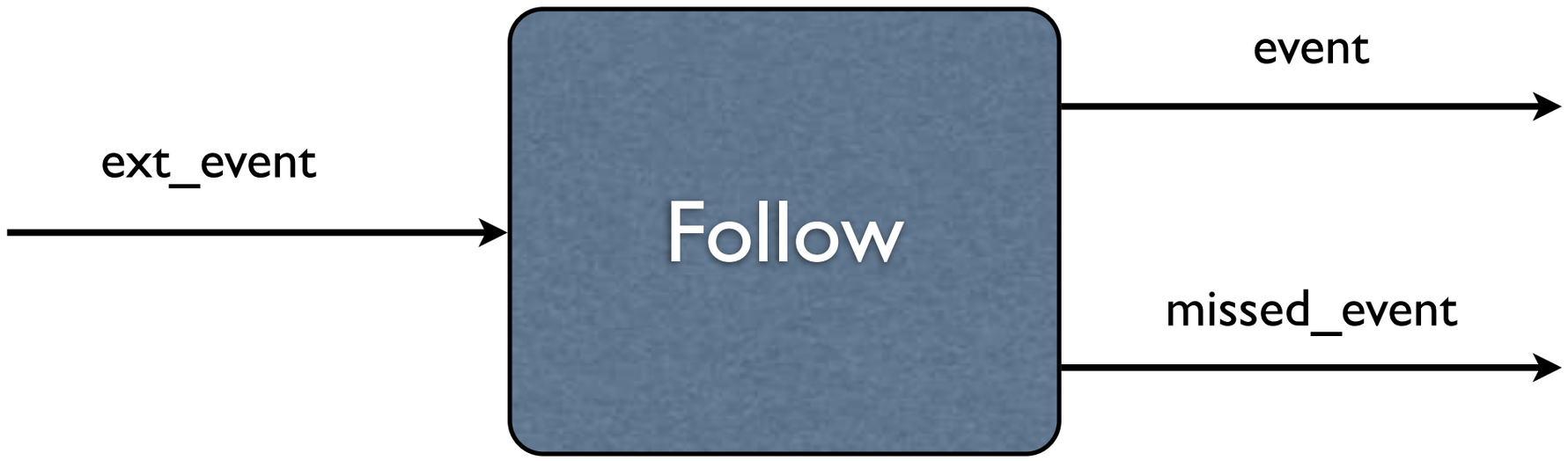
Logical Time

```
let process wait_abs dur =
  let d = int_of_float (dur /. clock_step) in
  for i=d downto 1 do pause;
```

```
let process ab d1 d2 =
  (run (wait_abs d1); print a)
  ||
  (run (wait_abs d2); print b)
```

Signals

```
signal ext_event default 0 gather fun x y -> x;;
signal event default 0 gather fun x y -> x;;
signal missed_event default [] gather fun x y -> x;;
```



Signals

```
signal ext_event default 0 gather fun x y -> x;;
signal event default 0 gather fun x y -> x;;
signal missed_event default [] gather fun x y -> x;;
```

```
let process follow ext_event event missed_event =
  loop
    await ext_event (ev) in
    let pre_event = last ?event in
    emit event ev;
    emit missed_event (nat (p_event+1) (event-1));
  end
```

Signals

Default value

```
signal ext_event default 0 gather fun x y -> x;;
signal event default 0 gather fun x y -> x;;
signal missed_event default [] gather fun x y -> x;;
```

```
let process follow ext_event event missed_event =
  loop
    await ext_event (ev) in
    let pre_event = last ?event in
    emit event ev;
    emit missed_event (nat (p_event+1) (event-1));
  end
```

Signals

Accumulation function

```
signal ext_event default 0 gather fun x y -> x;;
signal event default 0 gather fun x y -> x;;
signal missed_event default [] gather fun x y -> x;;
```

```
let process follow ext_event event missed_event =
  loop
    await ext_event (ev) in
    let pre_event = last ?event in
    emit event ev;
    emit missed_event (nat (p_event+1) (event-1));
  end
```

Signals

```
signal ext_event default 0 gather fun x y -> x;;
signal event default 0 gather fun x y -> x;;
signal missed_event default [] gather fun x y -> x;;
```

```
let process follow ext_event event missed_event =
  loop
    await ext_event (ev) in
    let pre_event = last ?event in
    emit event ev;
    emit missed_event (nat (p_event+1) (event-1));
  end
```

Await a reception

Signals

```
signal ext_event default 0 gather fun x y -> x;;
signal event default 0 gather fun x y -> x;;
signal missed_event default [] gather fun x y -> x;;
```

```
let process follow ext_event event missed_event =
  loop
    await ext_event (ev) in
    let pre_event = last ?event in
    emit event ev;
    emit missed_event (nat (p_event+1) (event-1));
  end
```

Emit on a signal

Signals

```
signal ext_event default 0 gather fun x y -> x;;
signal event default 0 gather fun x y -> x;;
signal missed_event default [] gather fun x y -> x;;
```

```
let process follow ext_event event missed_event =
  loop
    await ext_event (ev) in
    let pre_event = last ?event in
    emit event ev;
    emit missed_event (nat (p_event+1) (event-1));
  end
```

Last received value

Signals

```
signal ext_event default 0 gather fun x y -> x;;
signal event default 0 gather fun x y -> x;;
signal missed_event default [] gather fun x y -> x;;
```

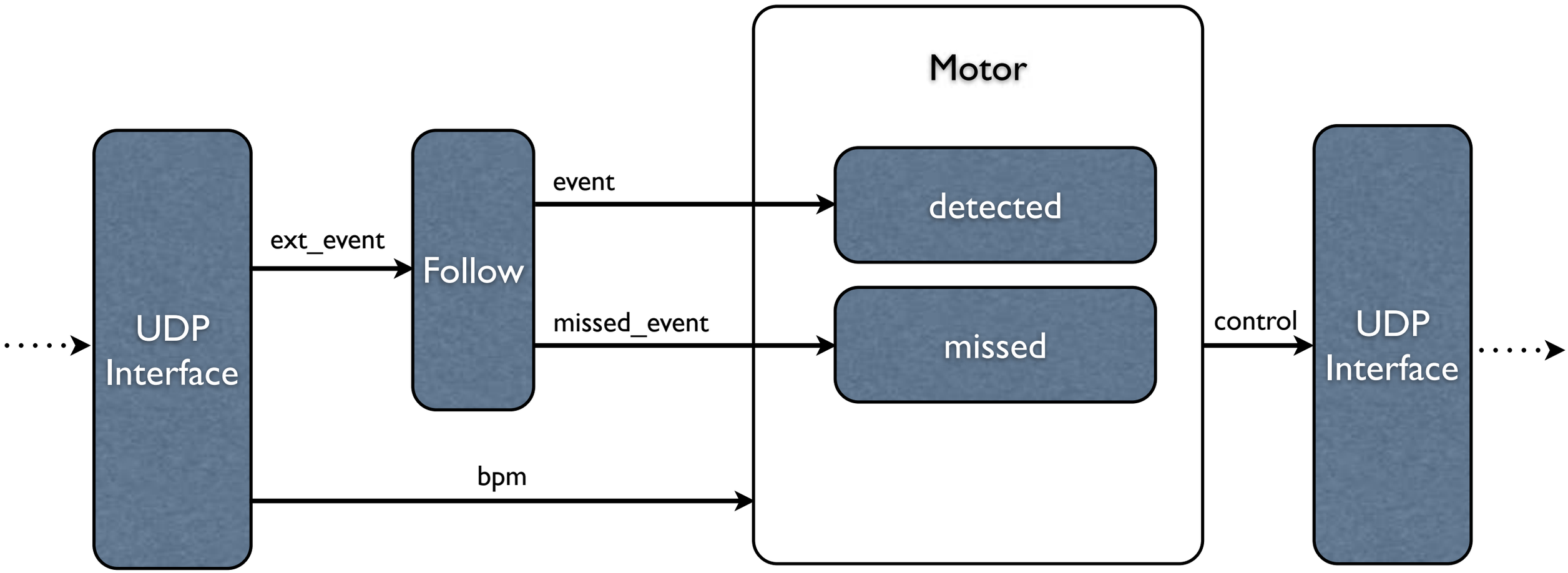
```
let process follow ext_event event missed_event =
  loop
    await ext_event (ev) in
    let pre_event = last ?event in
    emit event ev;
    emit missed_event (nat (p_event+1) (event-1));
  end
```

For each emission

Why Reactive ML ?

- Time programming expressivity
- High level programming
- Recursivity: Hierarchical structure
- Lazy waiting: tempo changes

Architecture



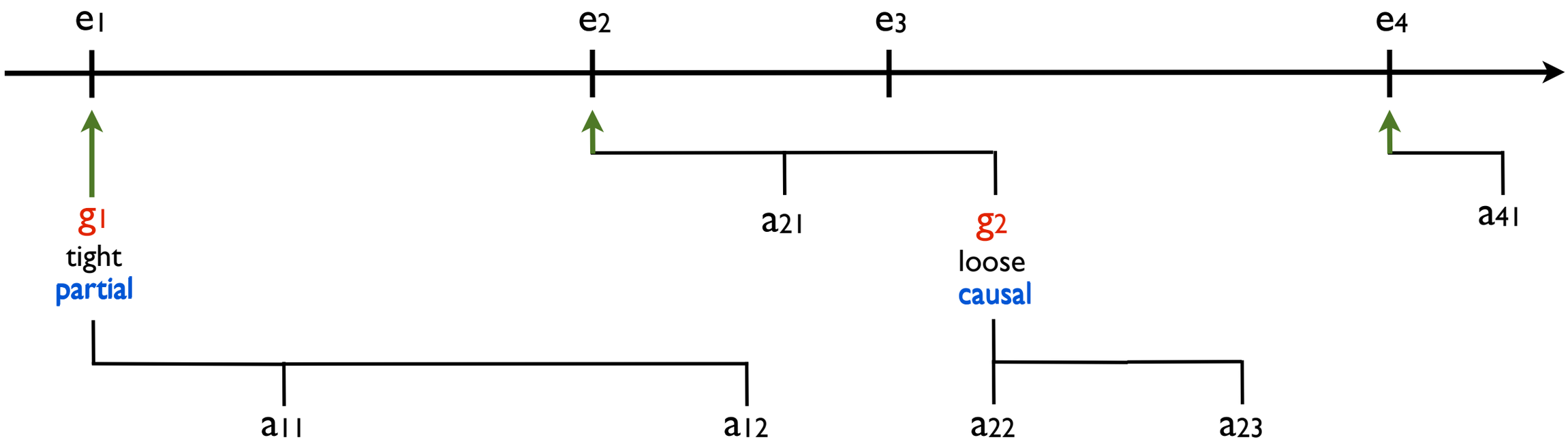
Types

```
(** Basic action *)
type action =
    { action_delay : delay;
      action_body : message; }

(** Asco group *)
type group =
    { group_delay : delay;
      group_synchro : sync;
      group_error : err;
      group_body : asco_event list; }

(** Generic asco event *)
and asco_event =
    Group of group
  | Action of action
```

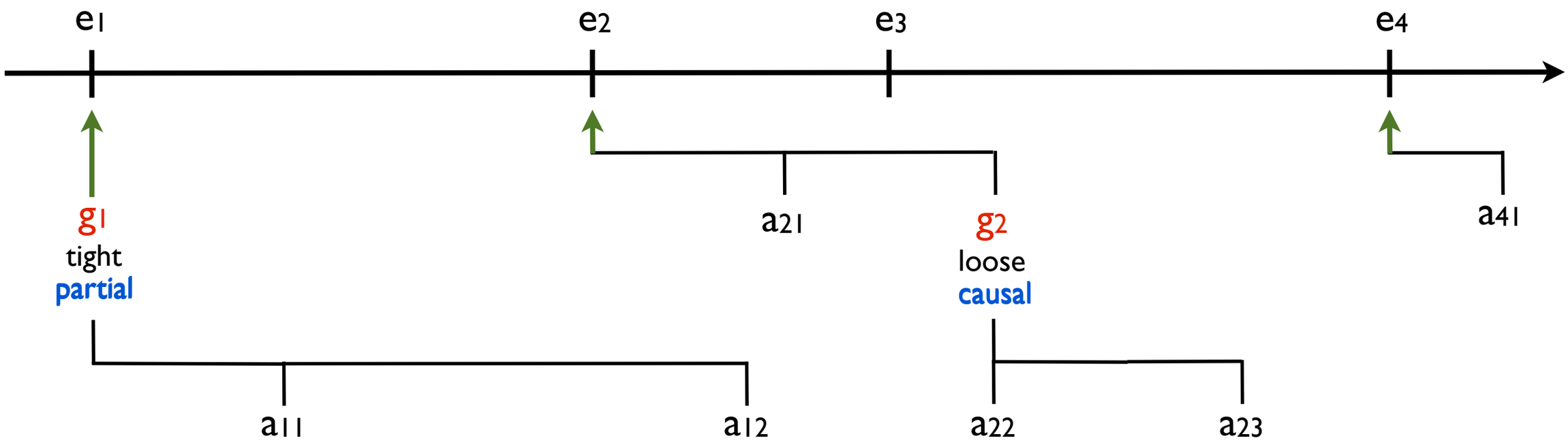
Execution

```
let rec process exec ae ev =
  (await event (e) when (ev = e) in
  run detect ae)
  ||
  (await missed_event (me) when (List.mem ev me) in
  run missed ae)

and process detect ae = ...

and process missed ae = ...
```

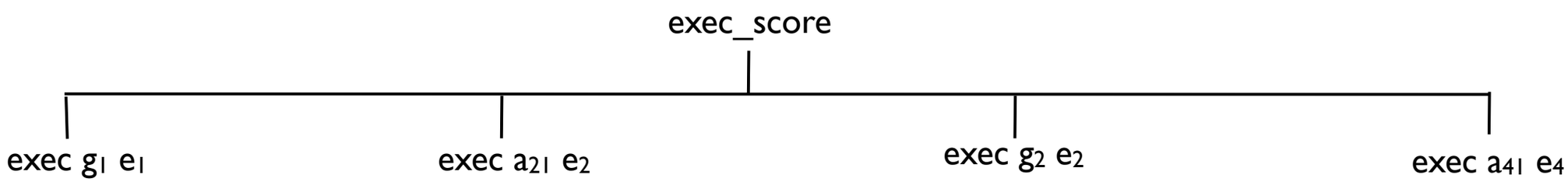
- Three mutually recursive processus
- Dynamical construction of the processus tree

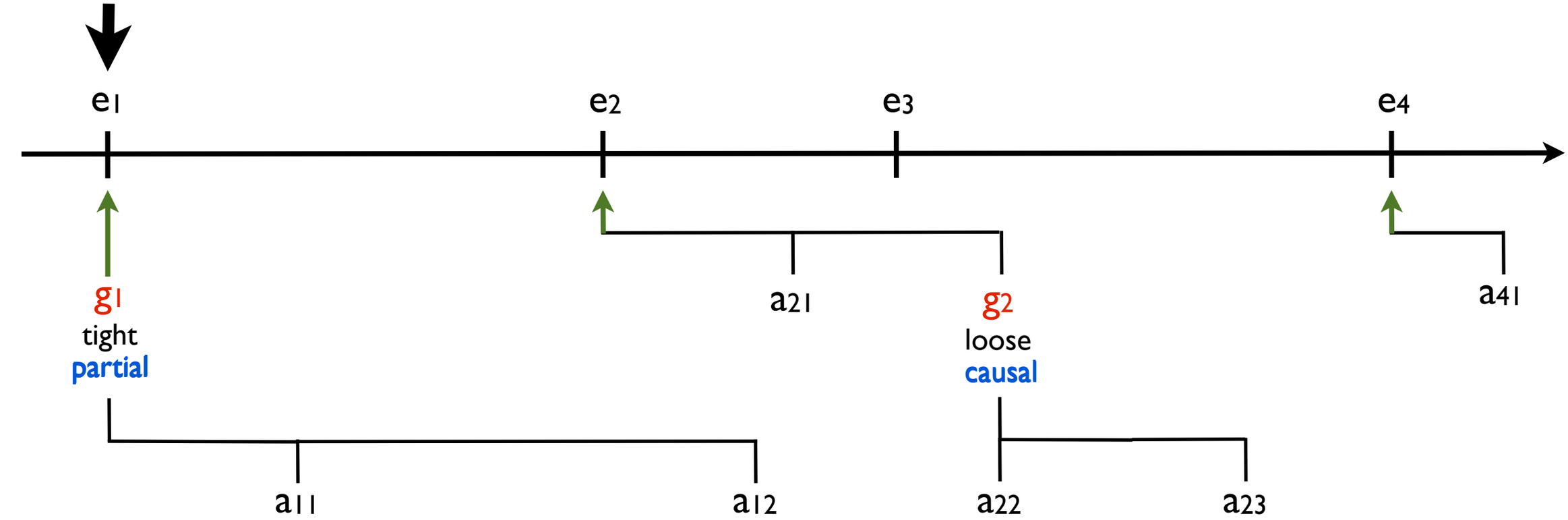


Score example

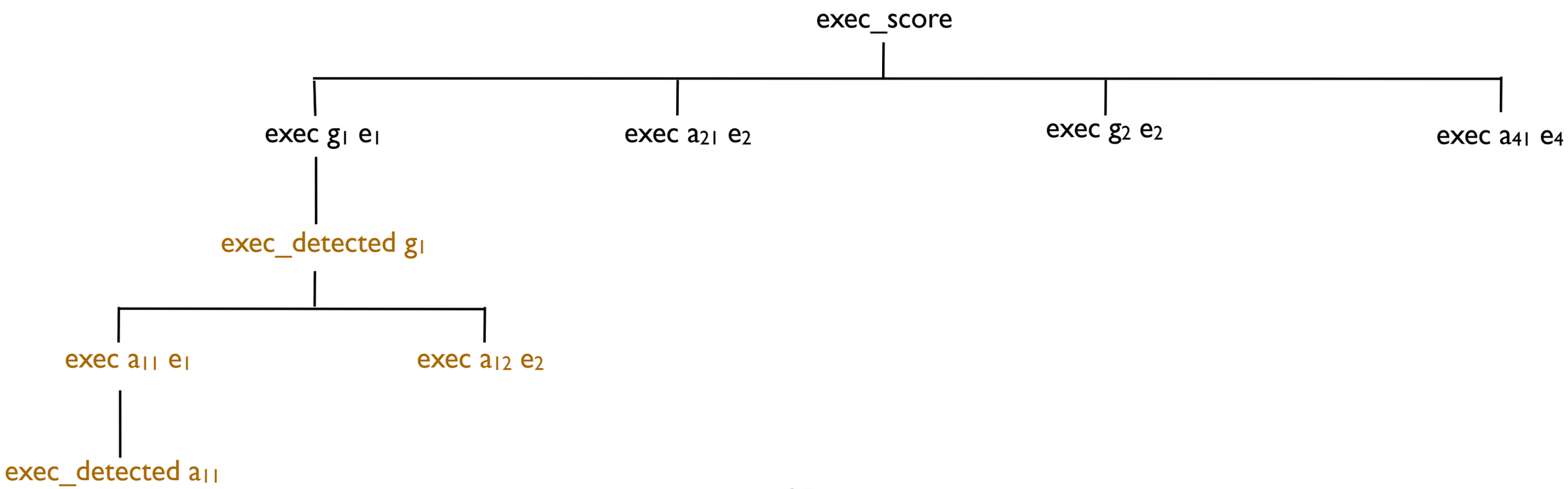


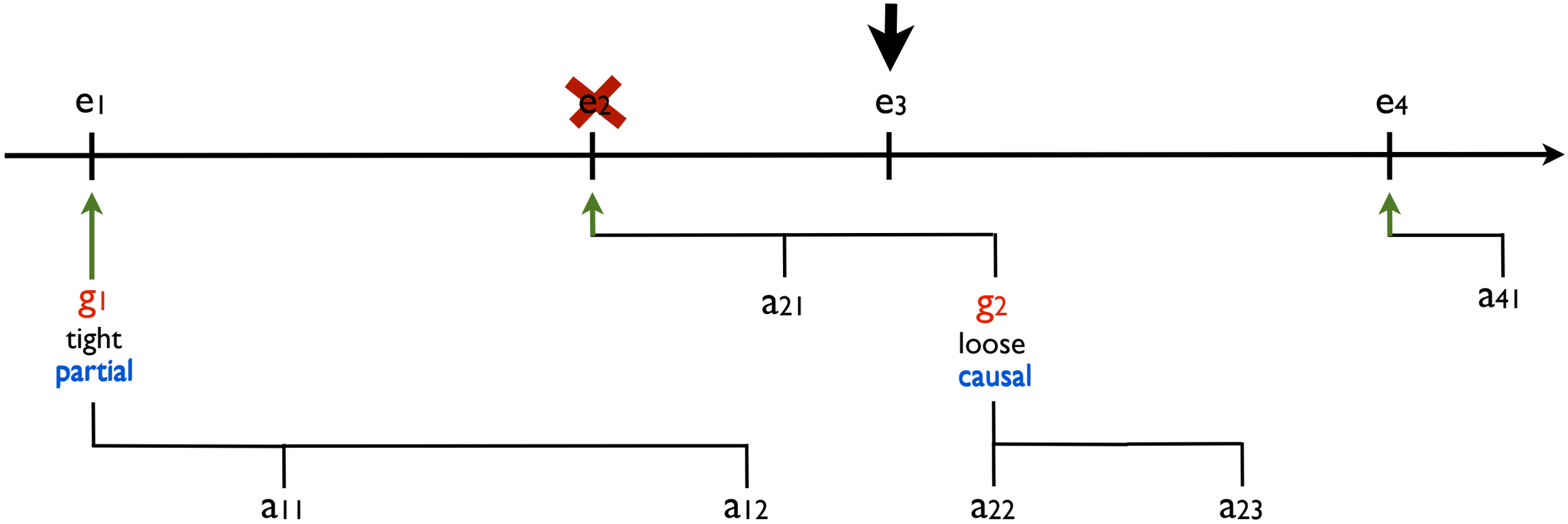
Score example



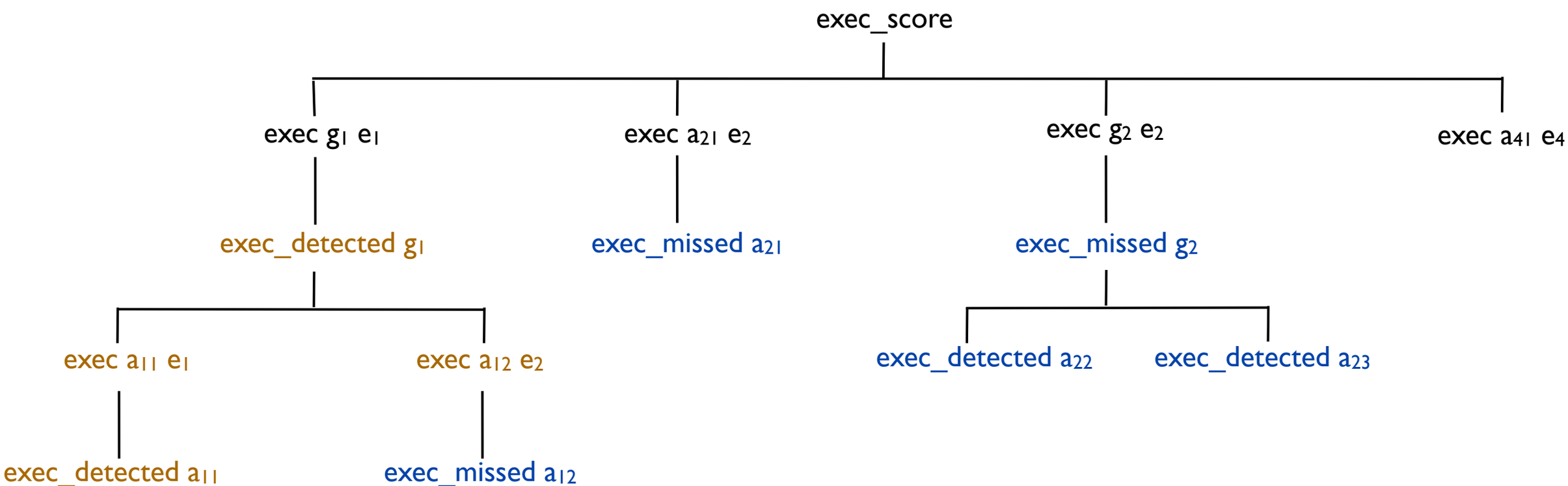


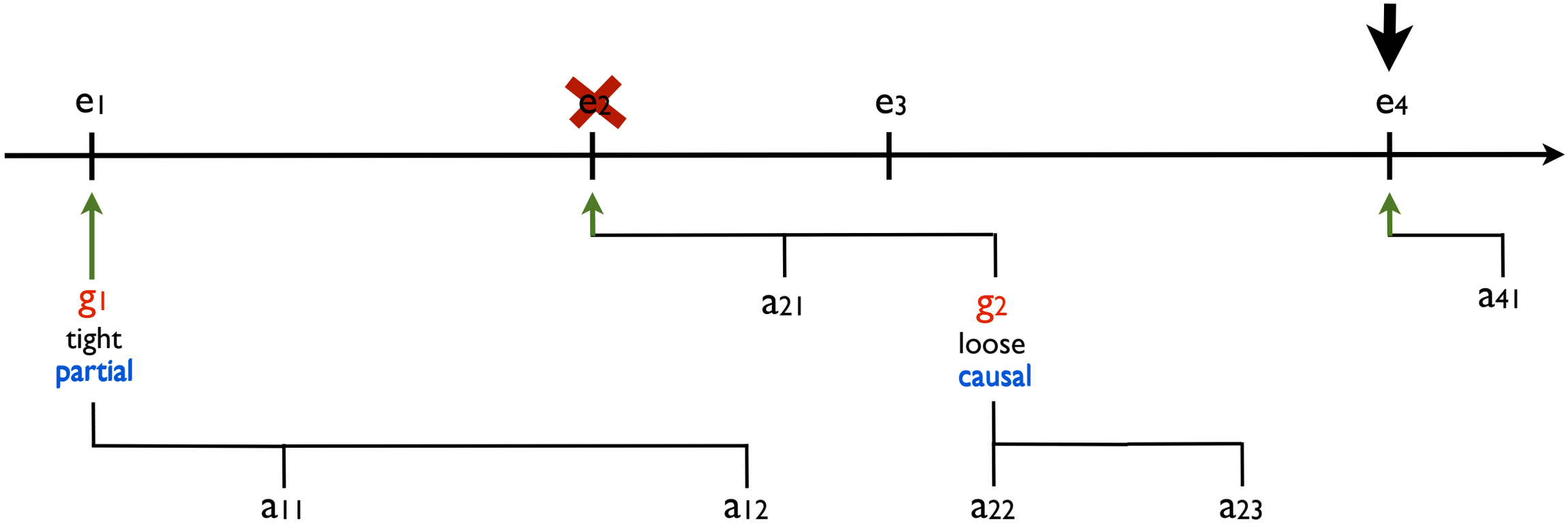
Score example



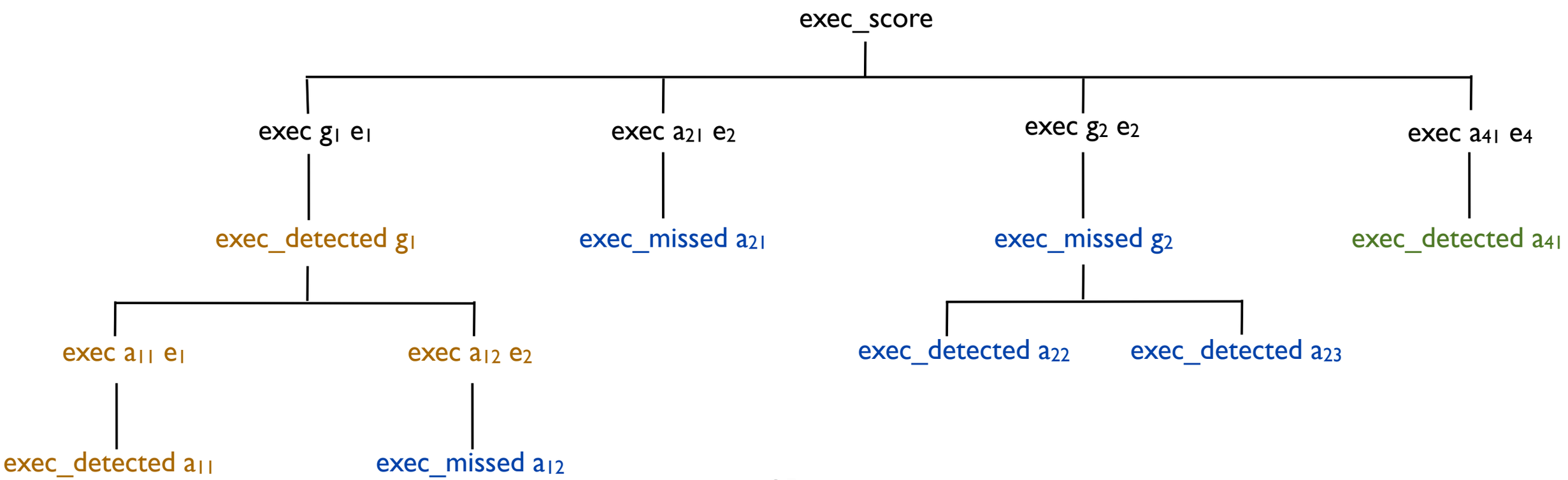


Score example





Score example



Summary

- Direct translation of the model
- Implementation close to the specifications
- Powerful tool for prototyping

An Interpret in Heptagon

Heptagon

- A Lustre/Scade like language
- Clock inference
- With advanced constructions (automata...)

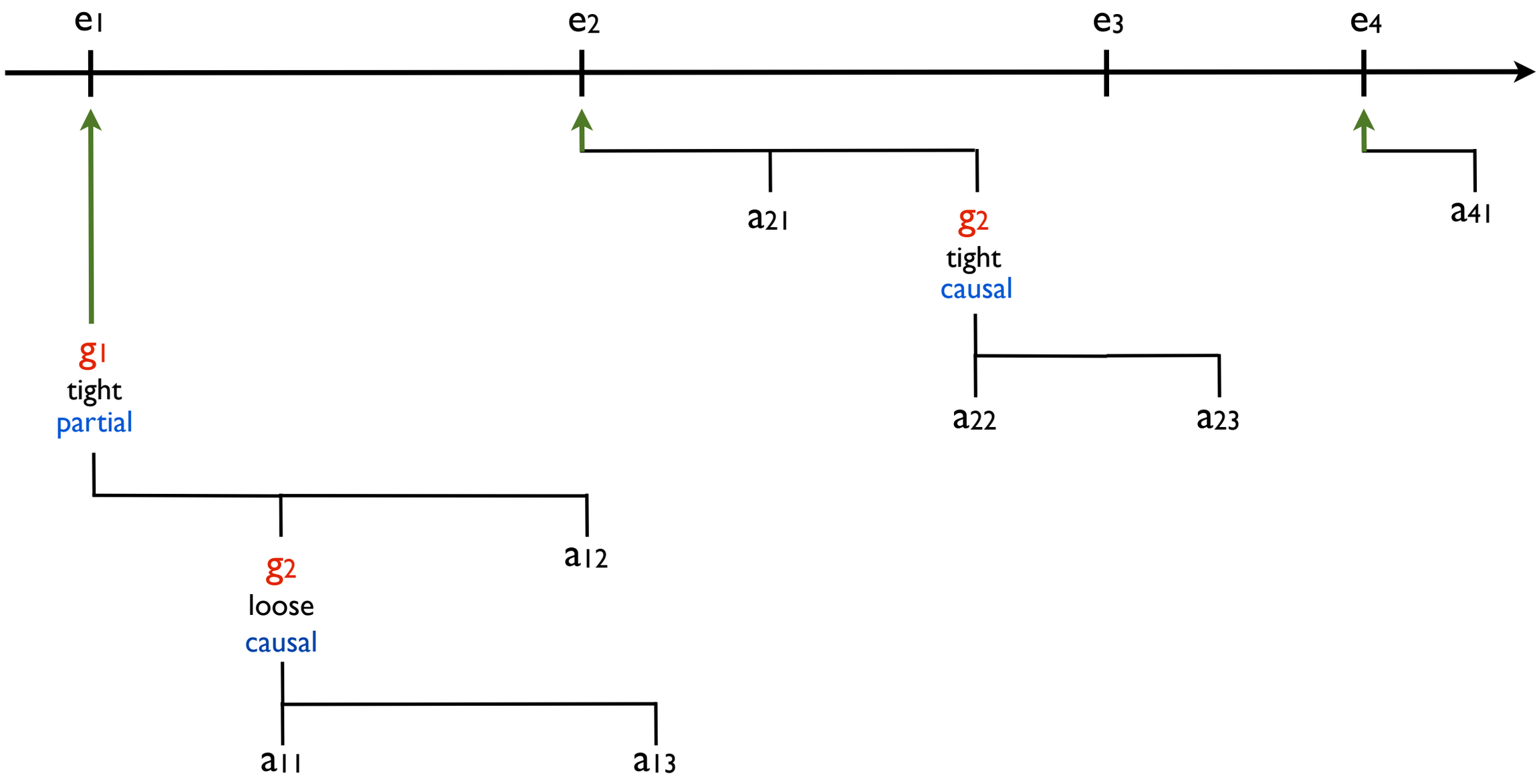
Why Heptagon ?

- Explore another point of view
- Generate C: Efficiency
- No dynamic allocation
- More guarantees on the runtime behavior (concert is a ‘critical situation’)

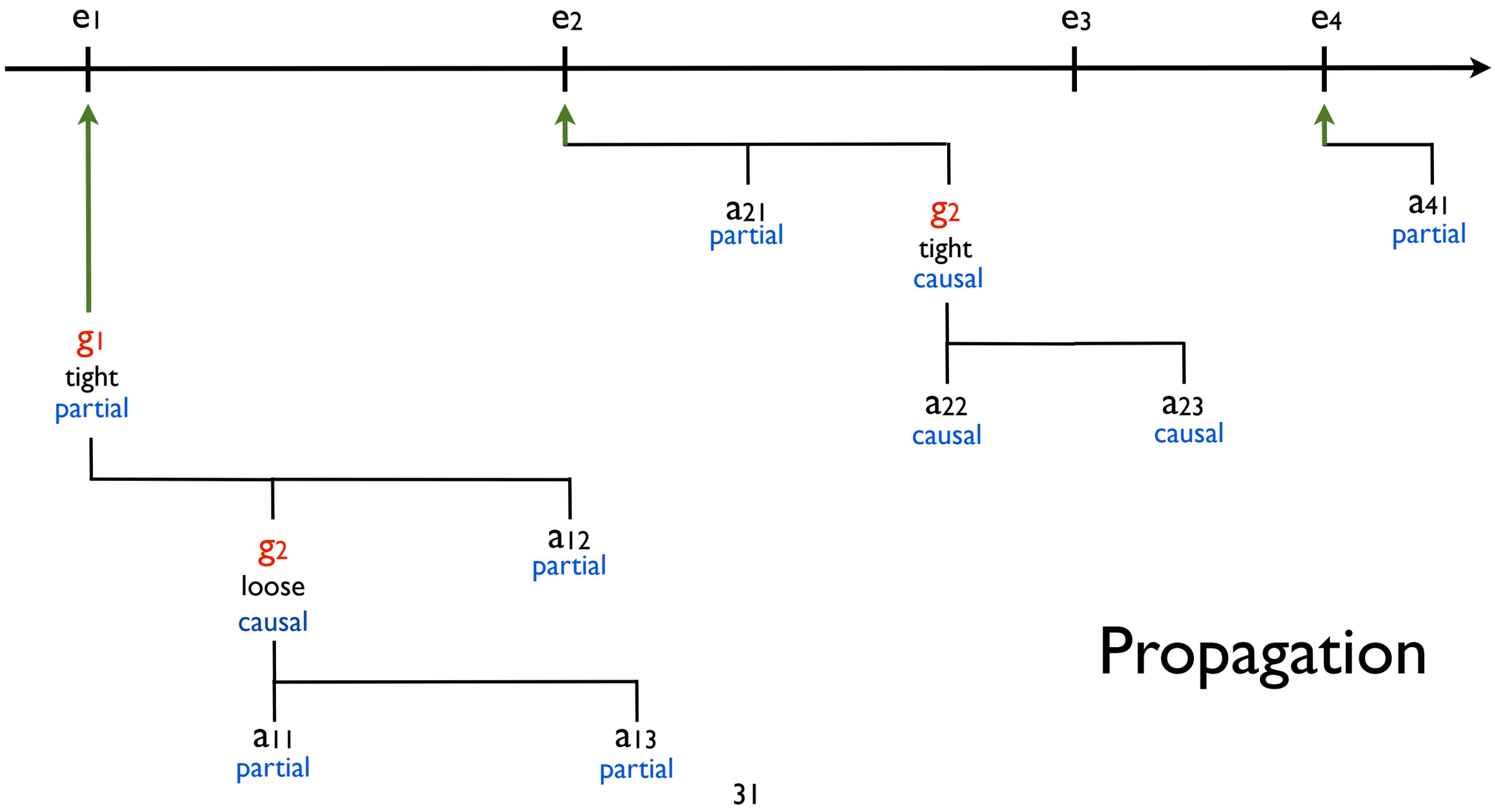
Normalization

- Propagation of error handling attributes
- Actions repartition: triggering events
- Flattening: no more groups

Example

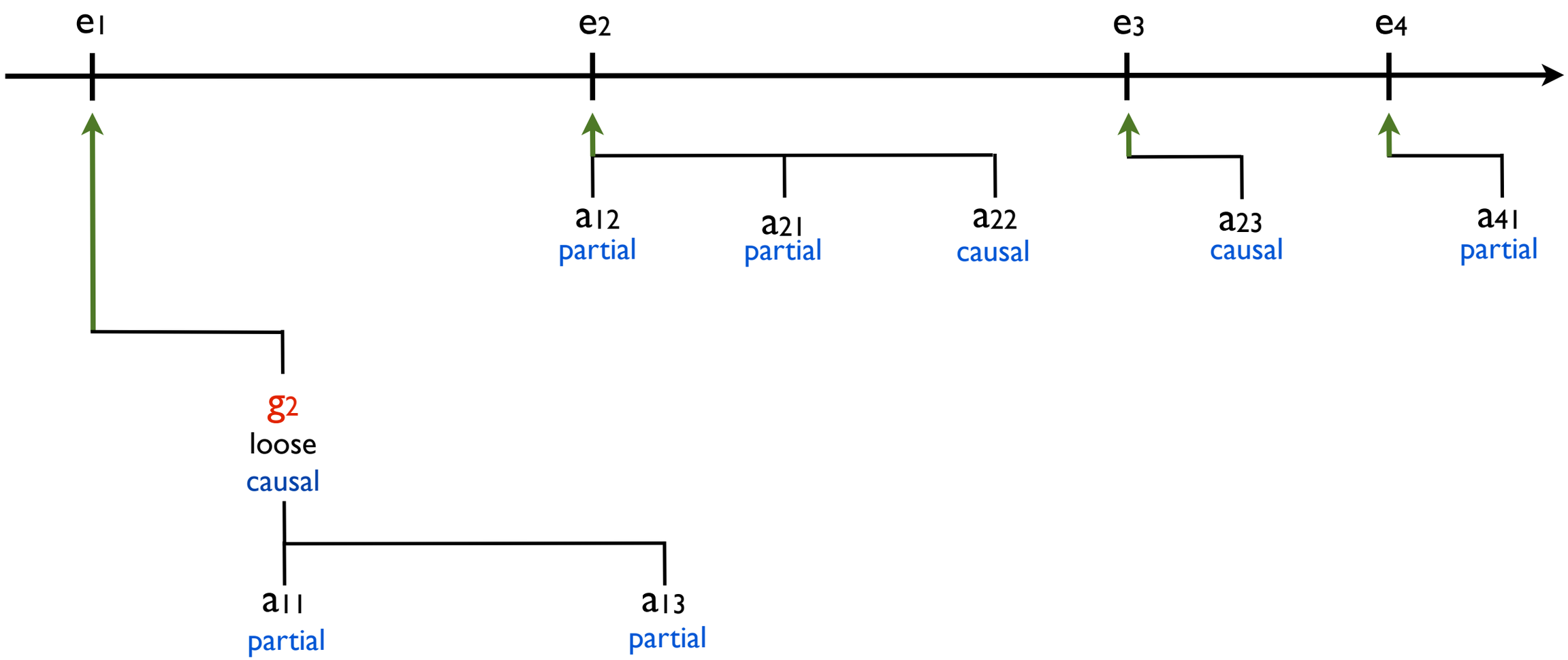

e1
e2
e3
e4
a21
g2
tight
causal
a41
g1
tight
partial
a22
a23
g2
loose
causal
a12
a11
a13

Example



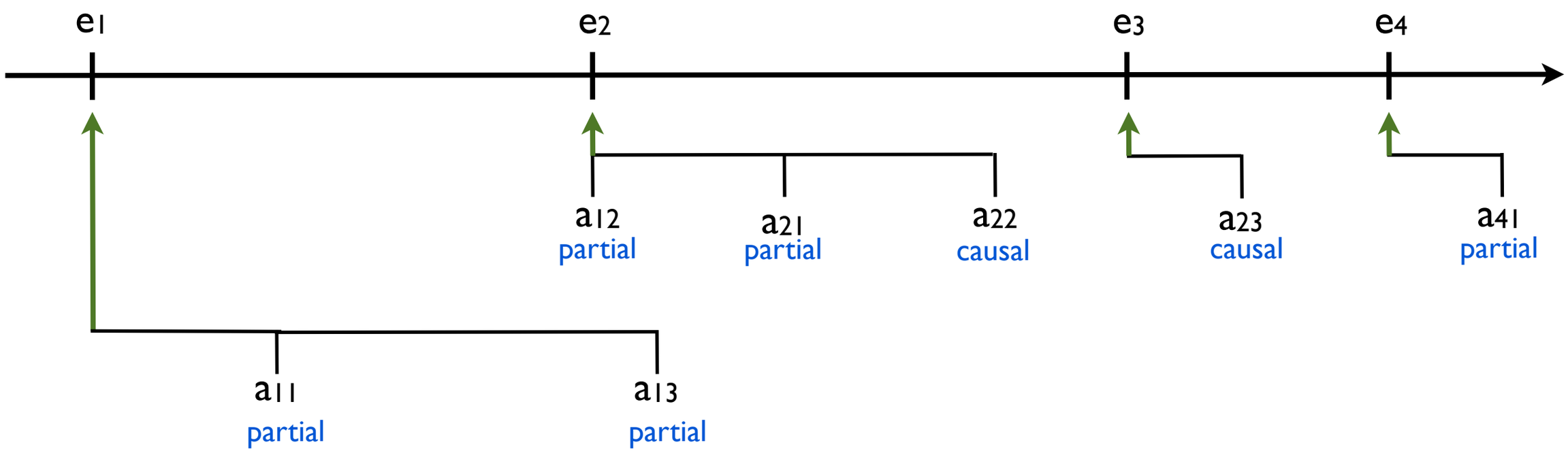
Propagation

Example



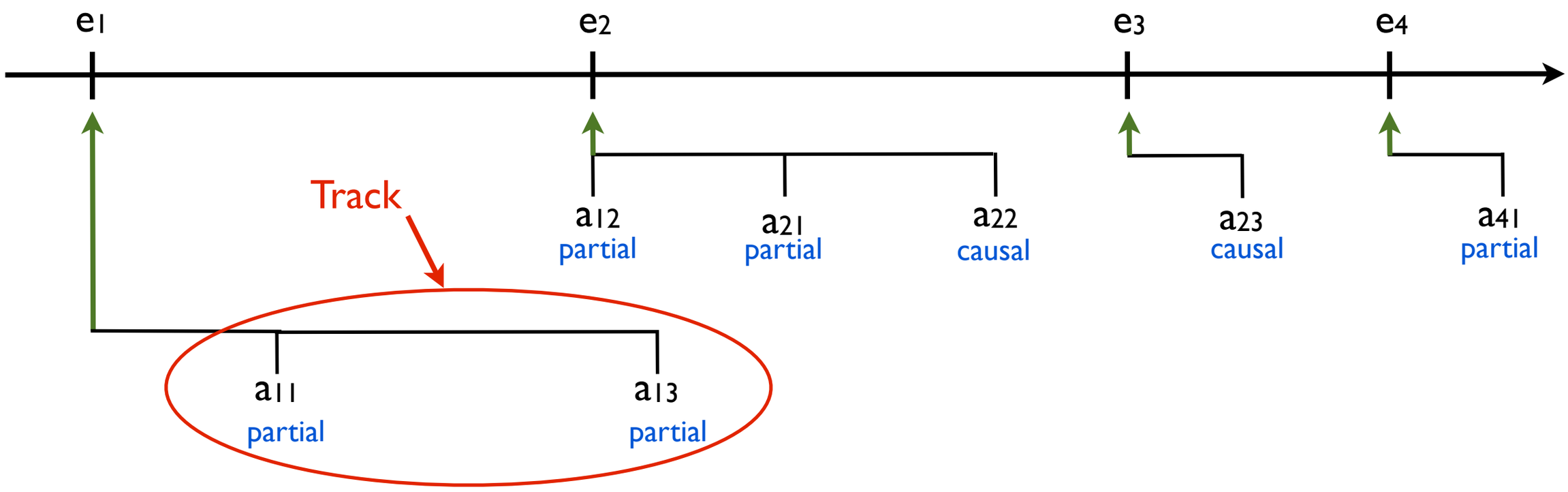
Repartition

Example

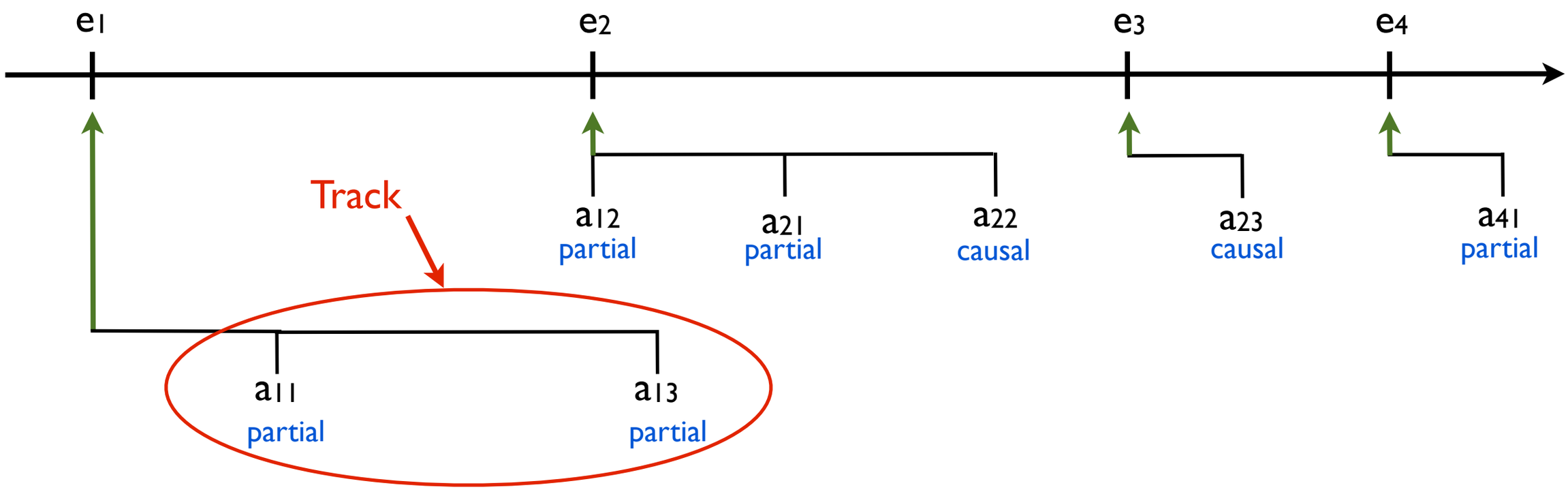


Flattening

Example


e1
e2
e3
e4
Track
a12
partial
a21
partial
a22
causal
a23
causal
a41
partial
a11
partial
a13
partial

Example



Score: Normalized and stored in a static structure.

Execution

```
node exec(tr:track) returns (o:bool)
var td,tm:bool;
let
  automaton
    state Wait
      do o = false;
      unless td then Detect
      unless tm then Missed
    state Detect
      do o = send(bpm,tr);
    state Missed
      do o = send(bpm,update(tr));
  end;
  td = testDetect(getEvent(tr),event);
  tm = testMissed(getEvent(tr),missed_event);
tel
```

Conclusion

- A Semantics for Antescofo
- Two synchronous points of view
- Two Antescofo Interprets

Perspectives

- Prototyping new features for Antescofo
  - New synchronization and error handling strategies
  - Variables, loops, etc...
- The Heptagon experiment
  - Prove some properties on the score
  - Performance could enable audio synthesis