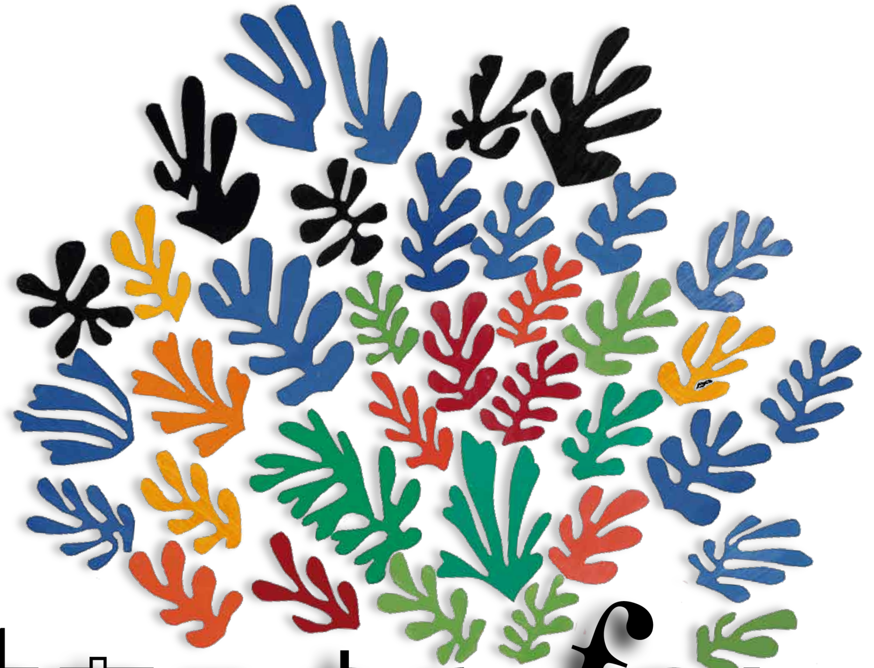


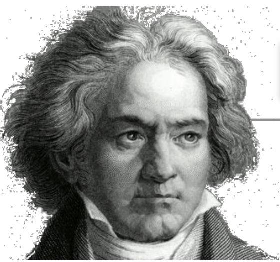
Antescofo

Les avancées du langage

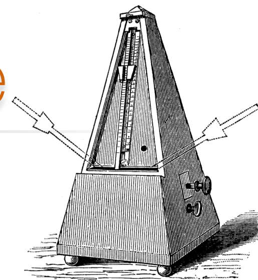
Jean-Louis Giavitto / IRCAM UMR 9912 STMS – CNRS – INRIA équipe MUTANT

- Un jeu à trois
- Aligner les lignes de temps: la synchronisation
- Paver la ligne de temps: structure de contrôle et de données

LE PROBLÈME



Le compositeur, le musicien et la machine



alignment with an event

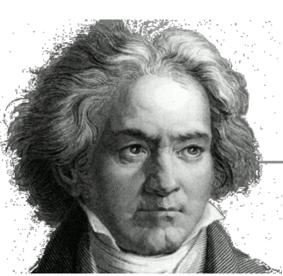
alignment with an event + tempo

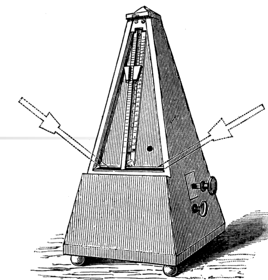
choosing an event

alignment with several events + several tempo

Giavitto / IRCAM UMR 99 CNRS – INRIA équipe M

Multiples timelines



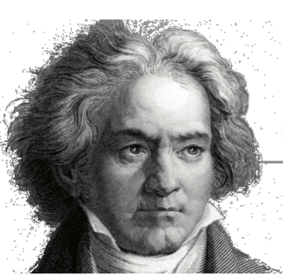


alignment with an event

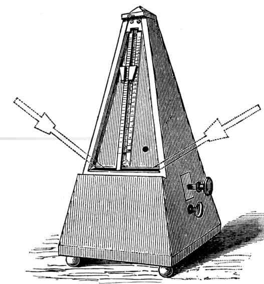
alignment with an event + tempo

choosing an event

alignment with several events + several tempo



Aligner les timelines



synchronisation

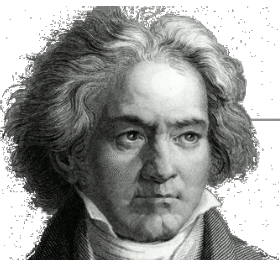
alignment with an event

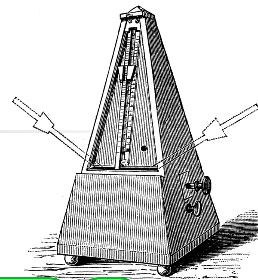
alignment with an event + tempo

choosing an event

alignment with several events + several tempo

Construire la « timeline »





alignment with an event

alignment with an event + tempo

choosing an event

alignment with several events + several tempo

Plan

1. Temps multiples
2. Repérage et coordination des repères temporels
3. Constructions
   a. structures de contrôle
   b. structures de données

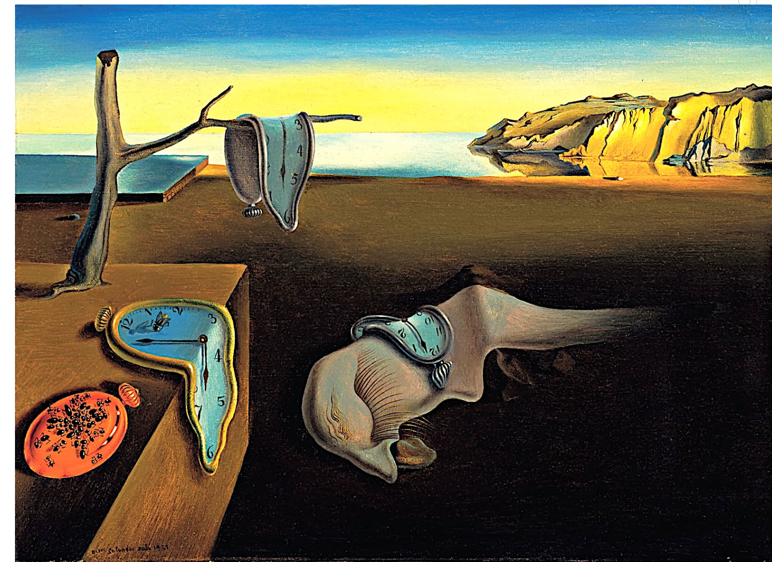
Jean-Louis Giavitto / IRCAM UMR 9912 STMS – CNRS – INRIA équipe MUTANT

1. TEMPS MULTIPLES

Du temps unique aux temps multiples

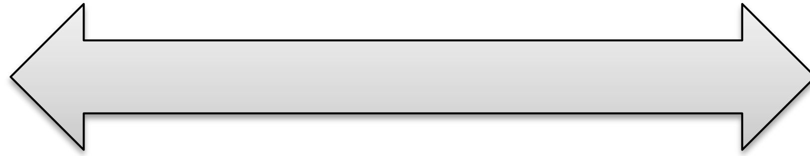
- temps unique : *une horloge externe objective*
  - les événements arrivent *dans* le temps
  - temps newtonien, unités temporelles fongibles
  - un temps partagé prescriptif
    (qui n'est éventuellement que partiellement connu)
- temps multiples : *pluralités co-dépendantes*
  - les événements définissent le temps
    (Bluedorn: epochal time is defined by events)
  - Temps leibnizien, relationnel
  - Exemples :
    - partition : couches temporelles
    - relation partition / performance
    - co-construction lors de la performance

912 STMS – CNRS – INRIA équipe MUTANT



Pluralité des temps

Construire un temps partagé

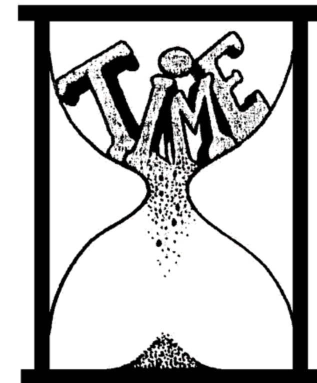


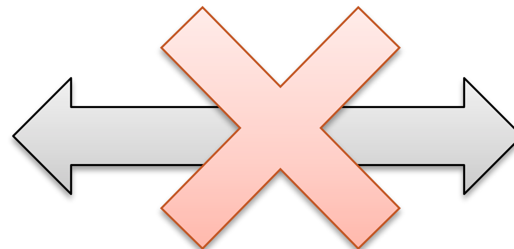
Jean-Louis Giavitto / IRCAM UMR 9912 STMS – CNRS – INRIA équipe MUTANT

Construire un temps partagé

tribu de la grande clepsydre

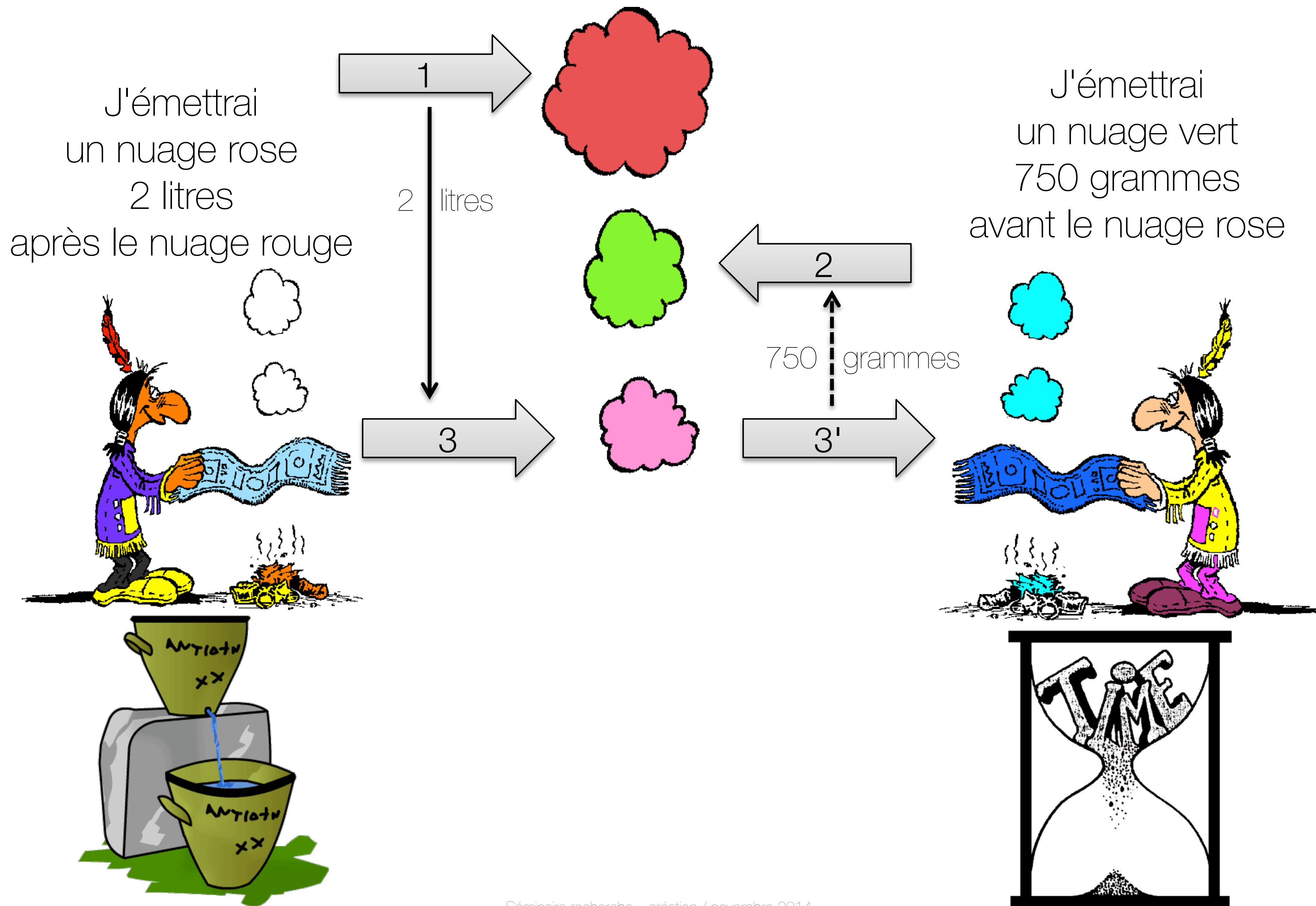
tribu du sable qui coule





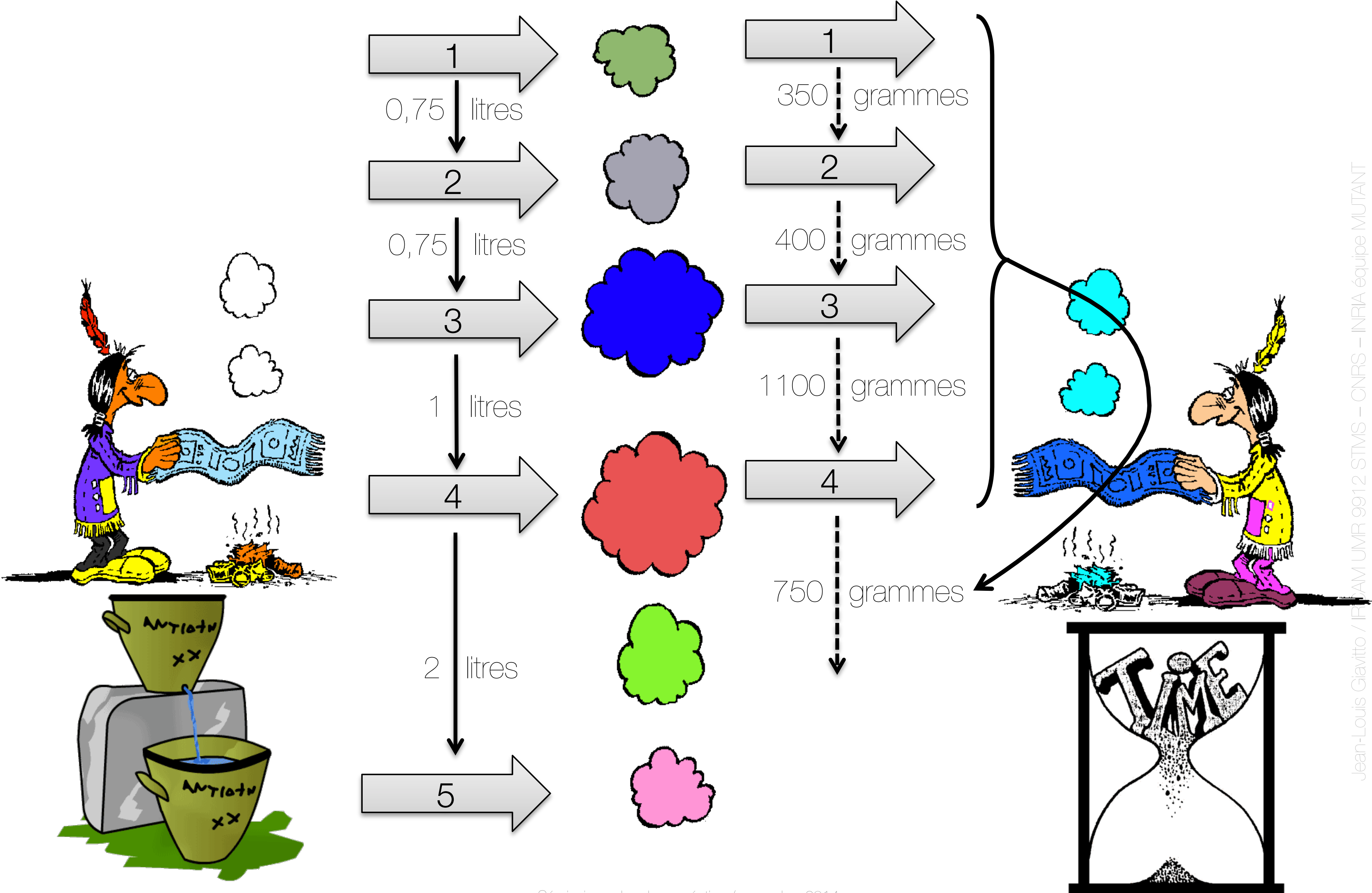
Jean-Louis Giavitto / IRCAM UMR 9912 STMS – CNRS – INRIA équipe MUTANT

Construire un temps partagé



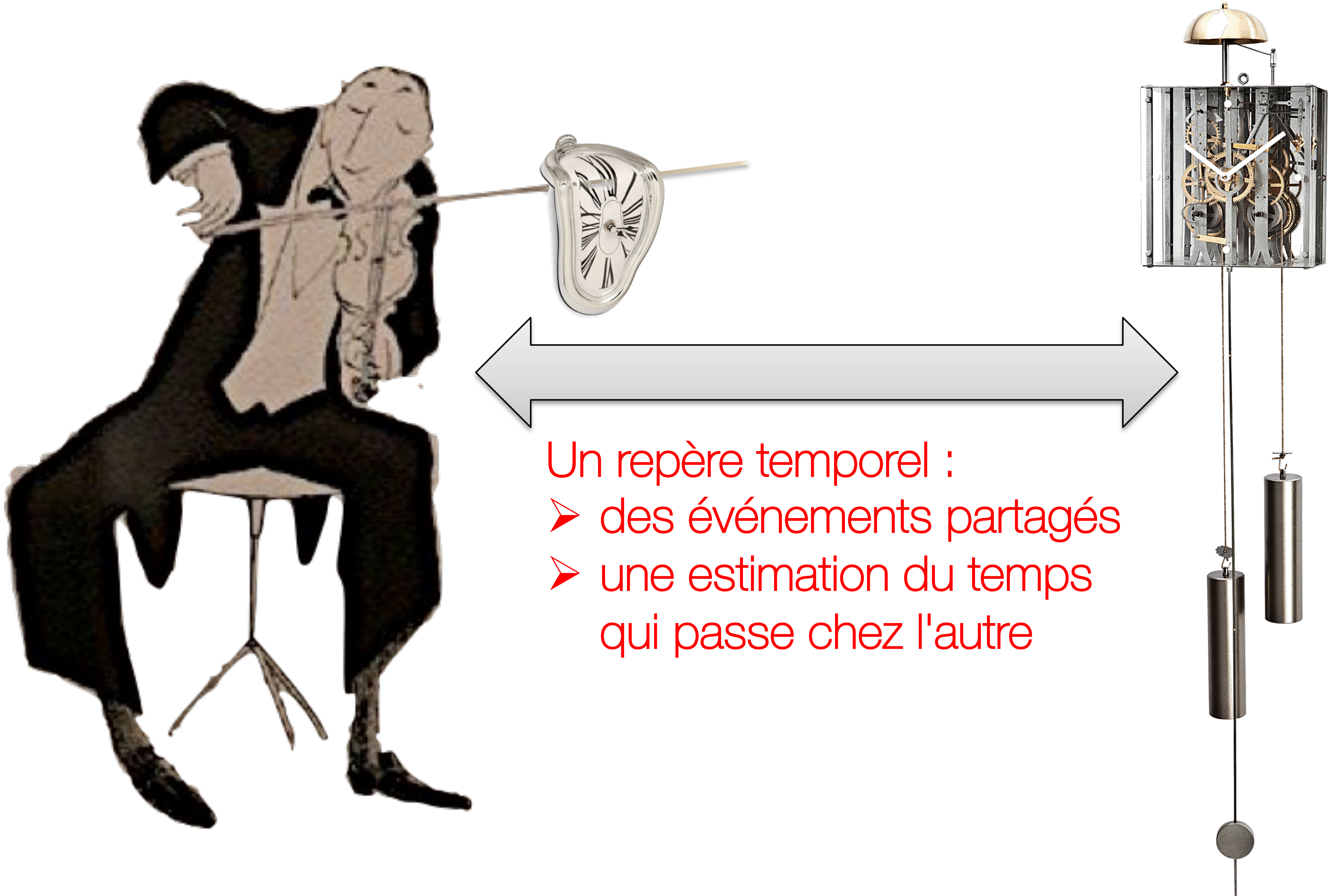
Jean-Louis Giavitto / IRCAM UMR 9912 STMS – CNRS – INRIA équipe MUTANT

Construire un temps partagé



Jean-Louis Giavitto / IRCAM UMR 9912 STMS – CNRS – INRIA équipe MUTANT

Aligner les repères temporels de chacun



Jean-Louis Giavitto / IRCAM UMR 9912 STMS – CNRS – INRIA équipe MUTANT

Exemple :

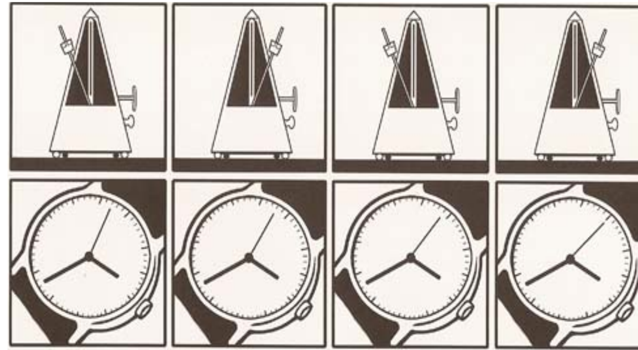
La déformation de la partition à l'interprétation

Qu'est ce qui reste invariant ?

- l'ordre des événements
- leur durée relative
- l'organisation concurrente des objets musicaux (accords, phrases…) est maintenu en respectant des points de synchronisation :
  - attaque/fin de note
  - début/fin de mesure
  - apogée d'une dynamique
  - changement de timbre

  i.e. événement partagés entre timelines
- propriétés qualitative de premier ordre (plus court/plus long) mais aussi de second ordre (plus lent/plus vite)
- souvent exprimable en terme de tempo relatif

Jean-Louis Giavitto / IRCAM UMR 9912 STMS – CNRS – INRIA équipe MUTANT



2. COORDINATION DES TIMELINES

Un système de coordonnées hybrides

position in beats in the score

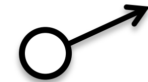
4
3
2
1
0

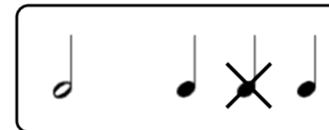
e1 60
e2 75
e4 55

0
1.8
2.5
3.6

time in seconds = $NOW

- événement musical
- tempo





detected events + estimated tempo

$A_NOW

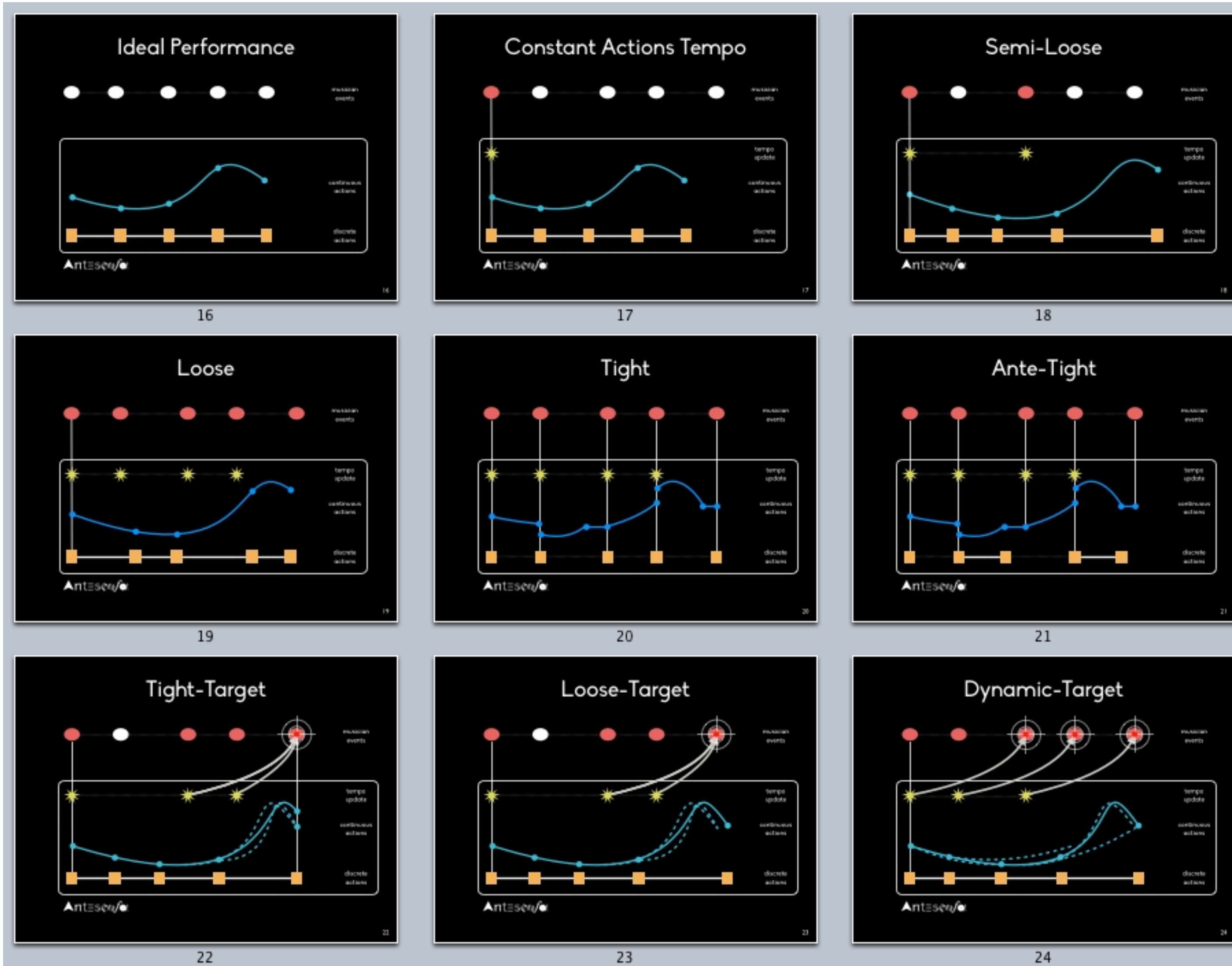
$T_NOW

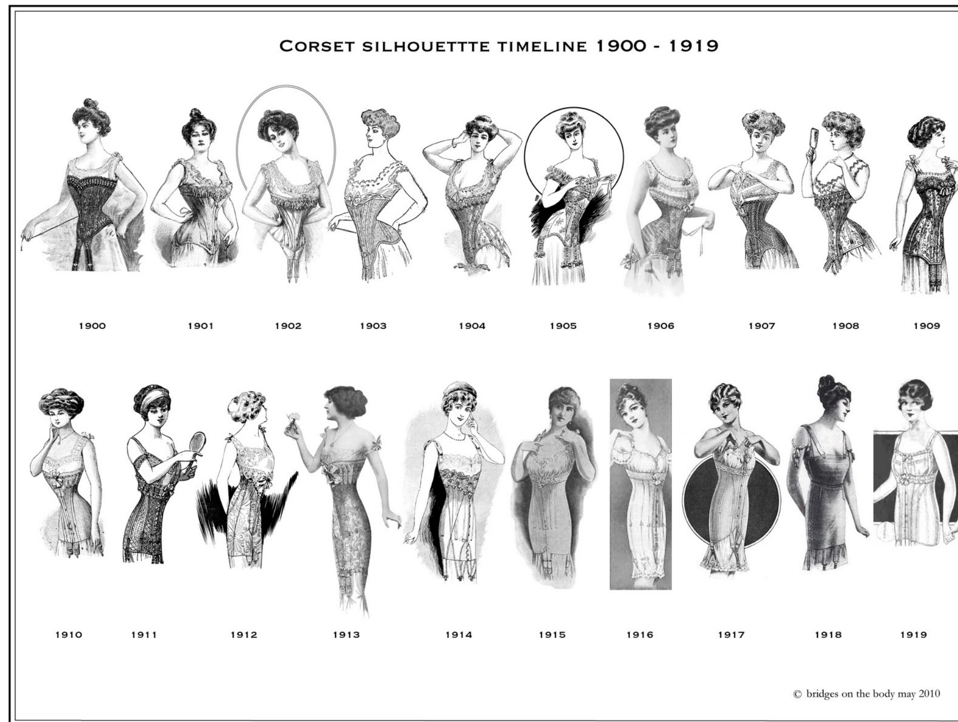
$L_NOW

$NOW

Jean-Louis Giavitto / IRCAM UMR 9912 STMS – CNRS – INRIA équipe MUTANT

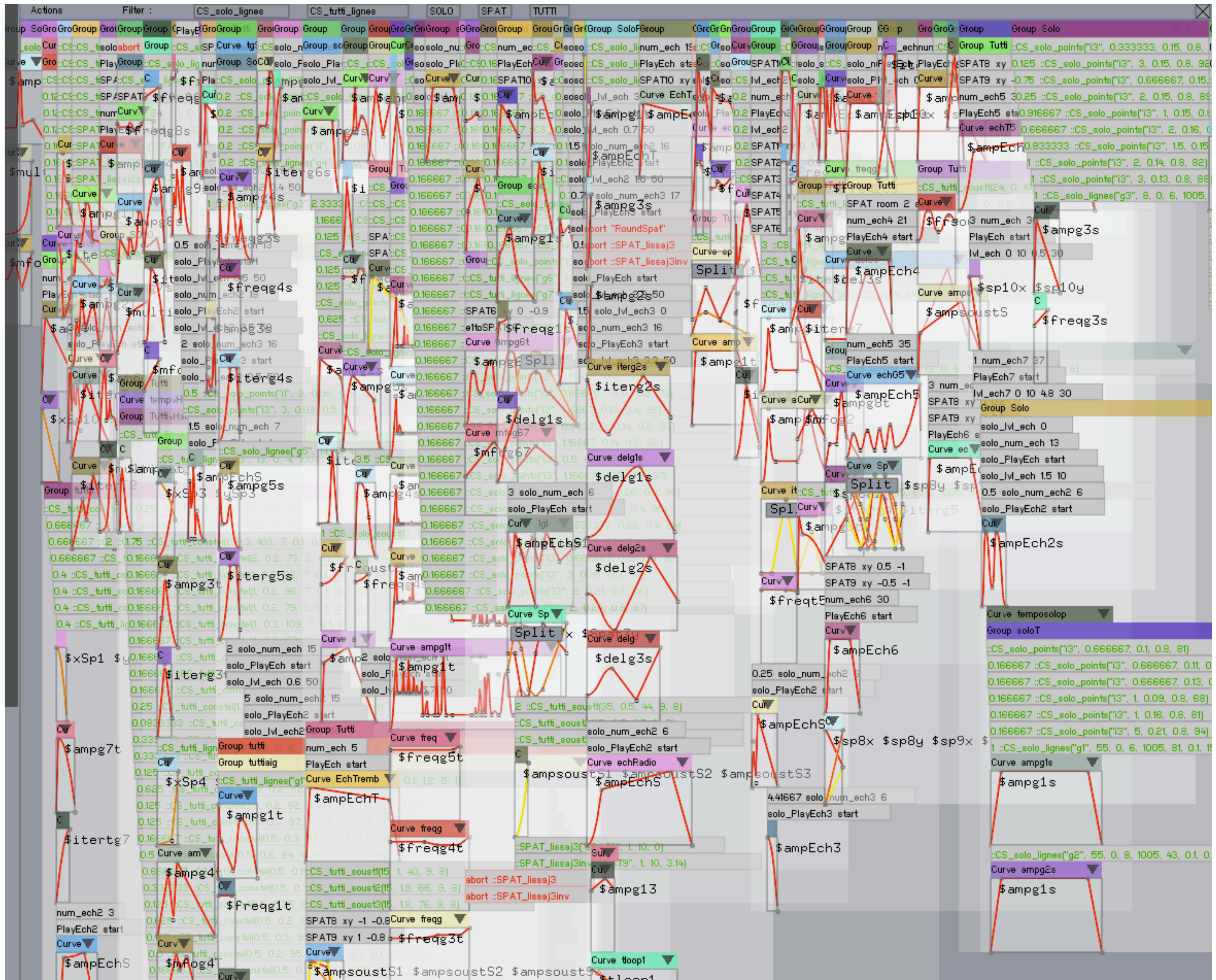
Stratégies de Synchronisation





3. CONSTRUIRE UNE TIMELINE

Nachleben, Julia Blondeau (8'30)

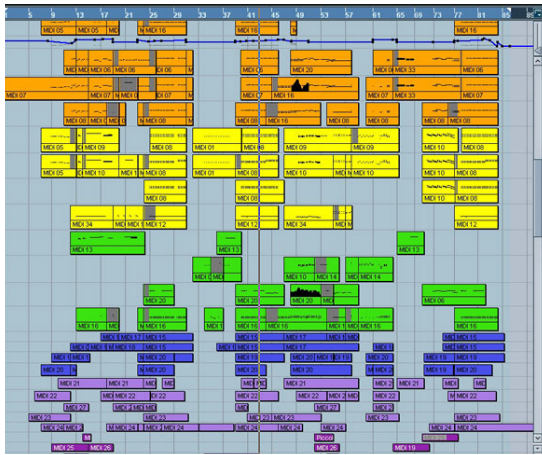


Jean-Louis Giavitto / IRCAM UMR 9912 STMS – CNRS – INRIA équipe MUTANT

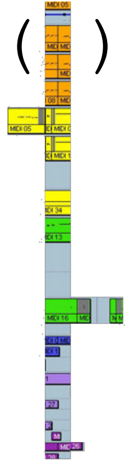
Construire une« timeline »

- Passage d'une définition *extensionelle* (qui liste les événements et les actions) à une définition intensionelle (programmatique)
- Structure de contrôle
  - Group
  - Loop (répétition)
  - Forall (répétition dépendant d'une structure de données)
  - If (déclenchement logique localisé temporellement)
  - Curve (action continue)
  - Processus
  - Whenever (déclenchement logique délocalisé temporellement)
  - Patterns (déclenchement logique complexe)
- Structure de données
  - Données scalaires
  - Map (dictionnaire)
  - Tab (vecteur)
  - NIM (fonction interpolée)

Extension vs. intension



`repeat( )`



Pourquoi passer d'une
description extensionelle
à une description intensionelle
de la timeline ?

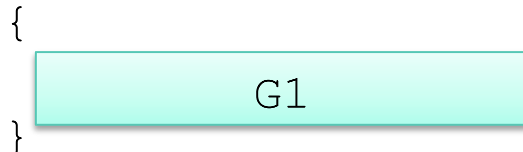
- \+ concis
- \+ expressif
  - réactif
  - paramétré par l'environnement extérieur
- \+ générique (bibliothèque, module)
- \+ idiosyncrasisme
  offrir à un compositeur la possibilité de définir un langage reflétant ses constructions temporelles *propres* (geste musical)

Jean-Louis Giavitto / IRCAM UMR 9912 STMS – CNRS – INRIA équipe MUTANT

STRUCTURES DE CONTRÔLE
(CURVE, PROCESSUS, . . . , PATTERNS)

Jean-Louis Giavitto / IRCAM UMR 9912 STMS – CNRS – INRIA équipe MUTANT

Group

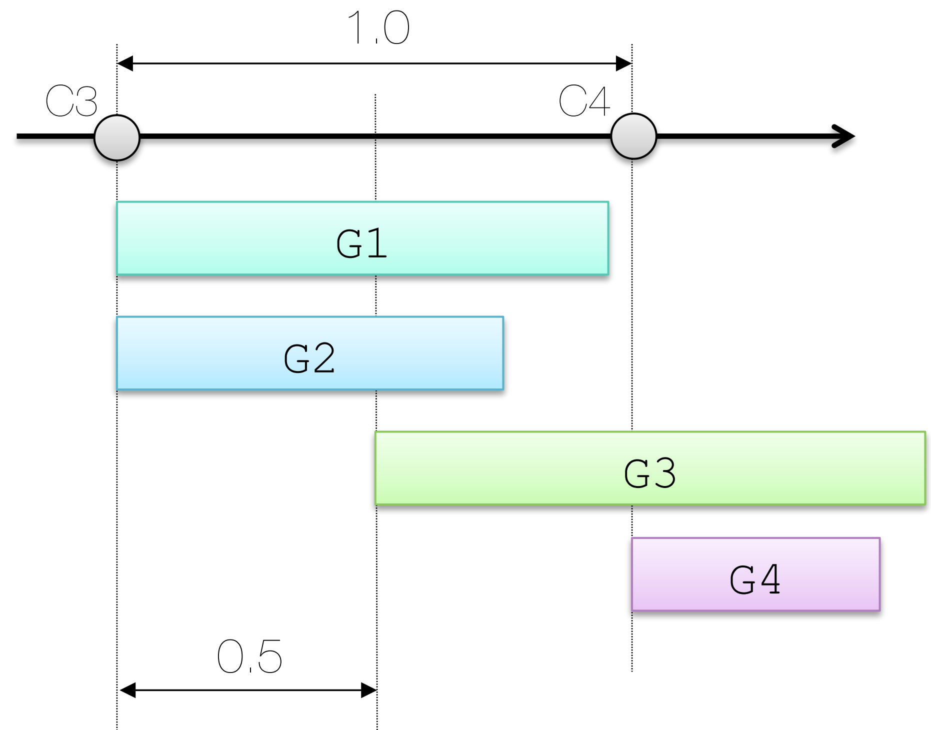
```
Note C3 1.0
    Group G1
    {
```



```
    }
    Group G2
    {
```

```
    }
    0.5 Group G3
    {
```

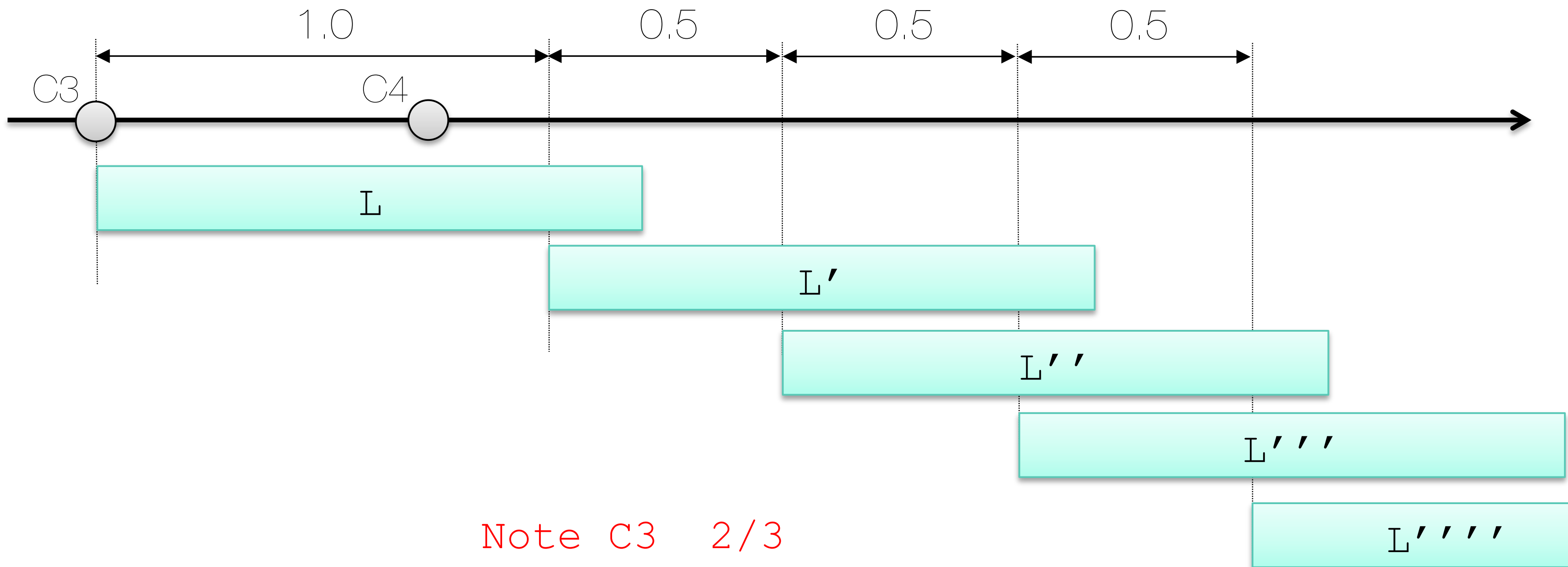
```
    }
    0.5 Group G4
    {
```

```
    }
Note C4 1.5
…
```



Jean-Louis Giavitto / IRCAM UMR 9912 STMS – CNRS – INRIA équipe MUTANT

Loop

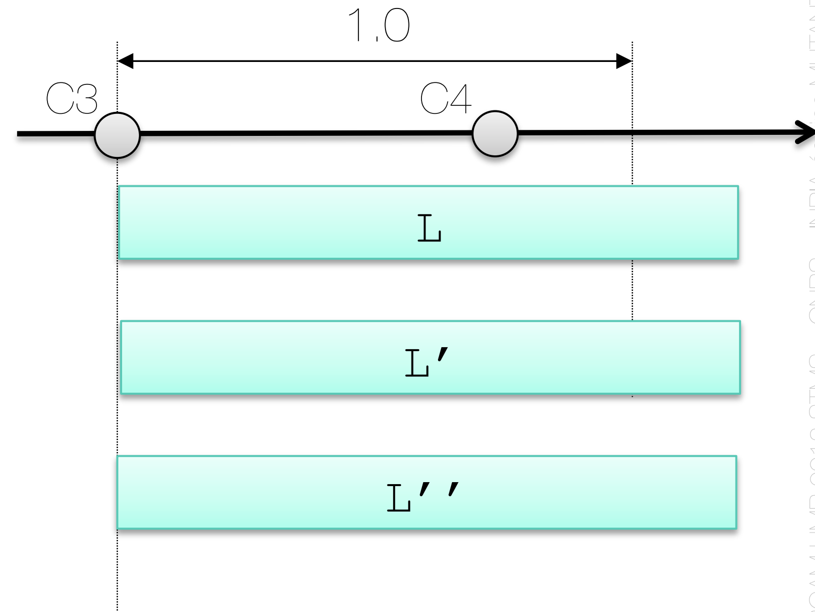



```
Note C3  2/3
    $periode := 1.0
    Loop $periode
    {
       $periode := 0.5
           … L …
    }
Note C4 1.5
…
```

Jean-Louis Giavitto / IRCAM UMR 9… STMS – CNRS – INRIA équipe MUTANT

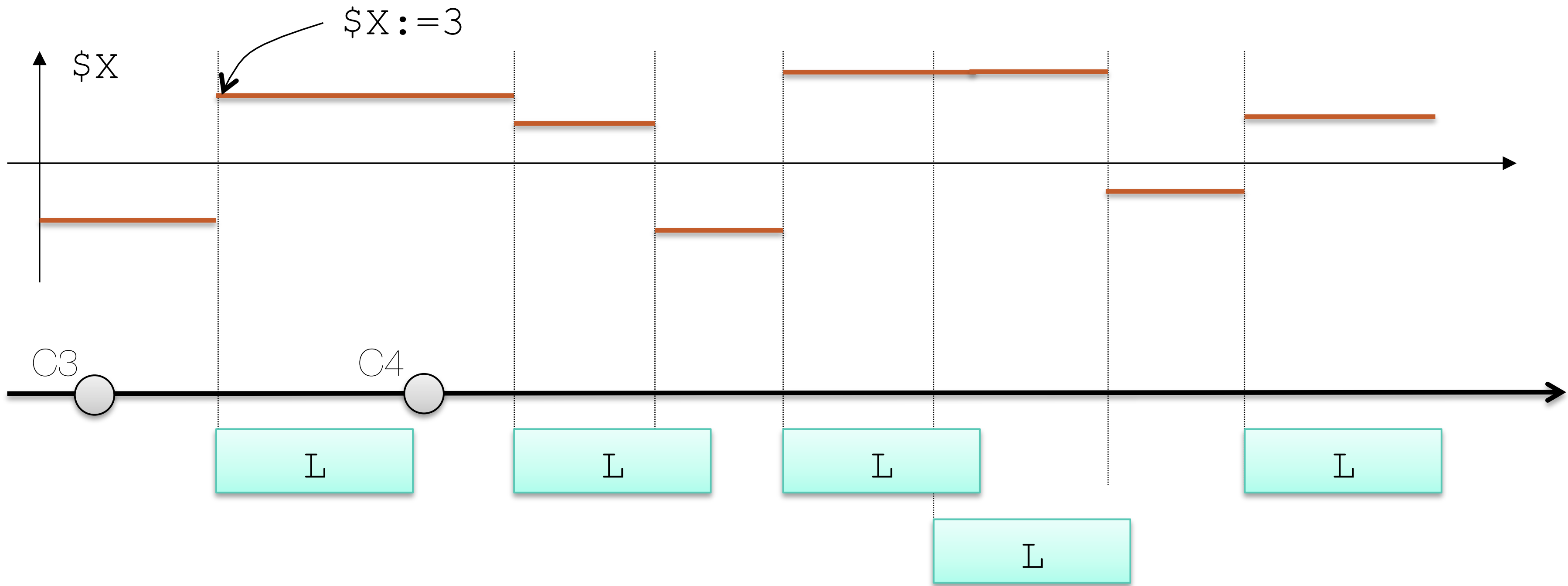
ForAll

```
Note C3  2/3
    ForAll $x in TAB[1, 2, 3] do
    {
      L
    }
Note C4 1.5
…
```



Jean-Louis Giavitto / IRCAM UMR 9912 STMS – CNRS – INRIA équipe MUTANT

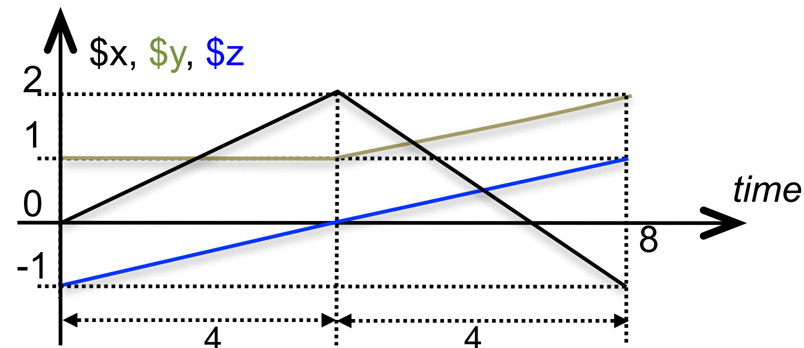
Whenever



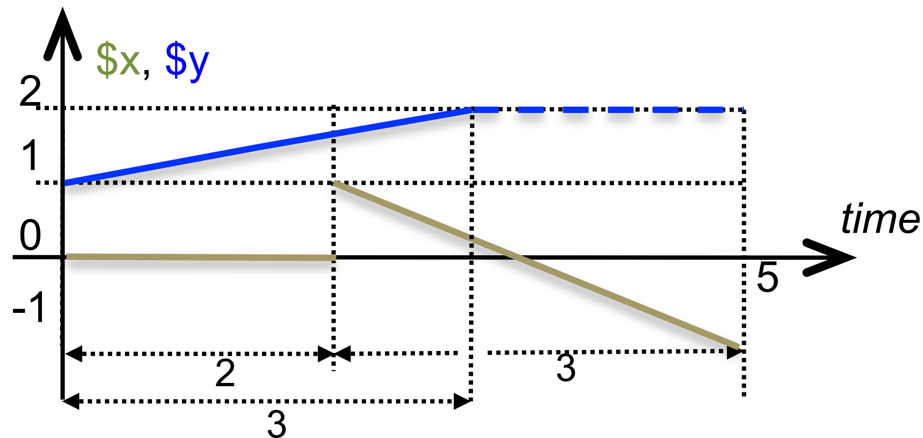

```
Note C3  2/3
    Whenever ($X > 0)
    {
       L
    }
Note C4 1.5
…
```

Curve

```
curve C
{  même durée, mêmes breakpoints
   $x, $y, $z
   {
          {  0, 1, -1 } @linear
        4 {  2, 1,  0 } @linear
        4 { -1, 2,  1 }
   }
}
```

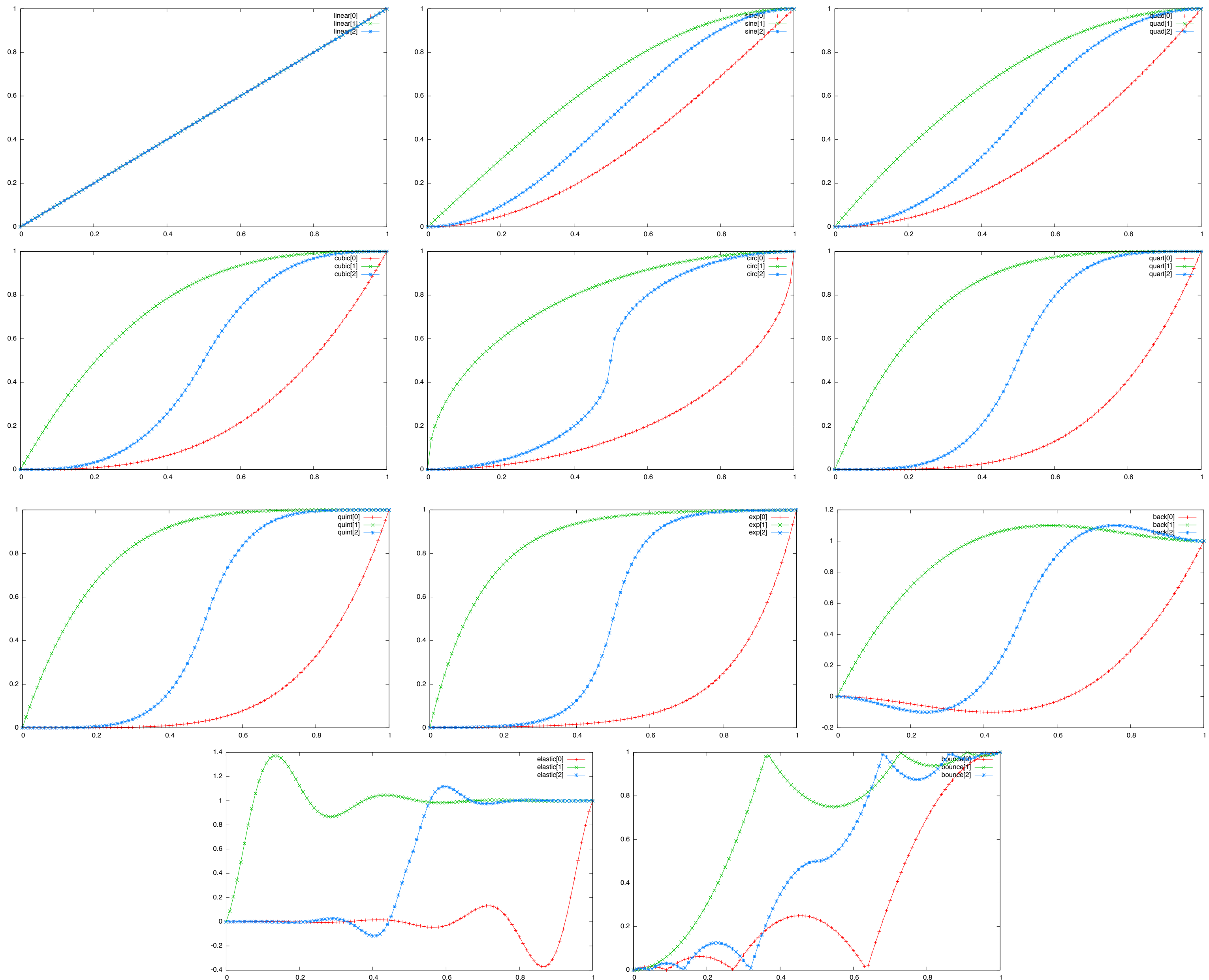



```
curve C
{  même durée, différents breakpoints
   $x
   {
          { 0  } @constant
      2   { 1  } @linear
      3   { -1 }
   }
   $y
   {
          { 1 } @linear
        3 { 2 }
   }
}
```



Jean-Louis Giavitto / IRCAM UMR 9912 STMS – CNRS – INRIA équipe MUTANT

NIM: type d'interpolation



Arrêter une action composée

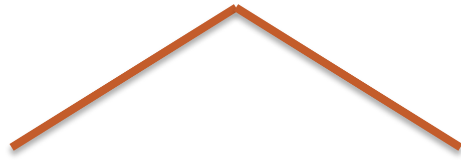
- until / while
  - Loop { … } until ($x > 33)
- duration
  - Loop { .. } during [3#]
  - Loop { .. } during [3s]
  - Loop { .. } during [3]
- abort
  - par nom ou par instance
  - récursif ou non
- @abort

Jean-Louis Giavitto / IRCAM UMR 9912 STMS – CNRS – INRIA équipe MUTANT

Exemple : le fade d'un processus contrôlé par une curve

```
Loop L 10
{
      Curve
      @action = csound $x
      @abort =
            {
                  Curve
                  @action csound $y
                { $y { { $x } 0.5 { 0 } } }
            }
     {
       $x { {0} 5 {1} 5 {0} }
     }
}
```

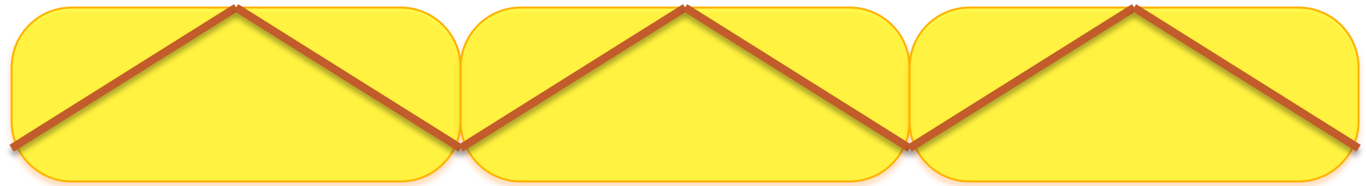
Exemple : le fade d'un processus contrôlé par une curve



```
Loop L 10
{
      Curve
      @action = csound $x
      @abort =
            {
                  Curve
                  @action csound $y
                { $y {  { $x } 0.5 "linear" { 0 } } }
            }
     {
        $x {  {0} 5 {1} 5 {0}  }
     }
}
```

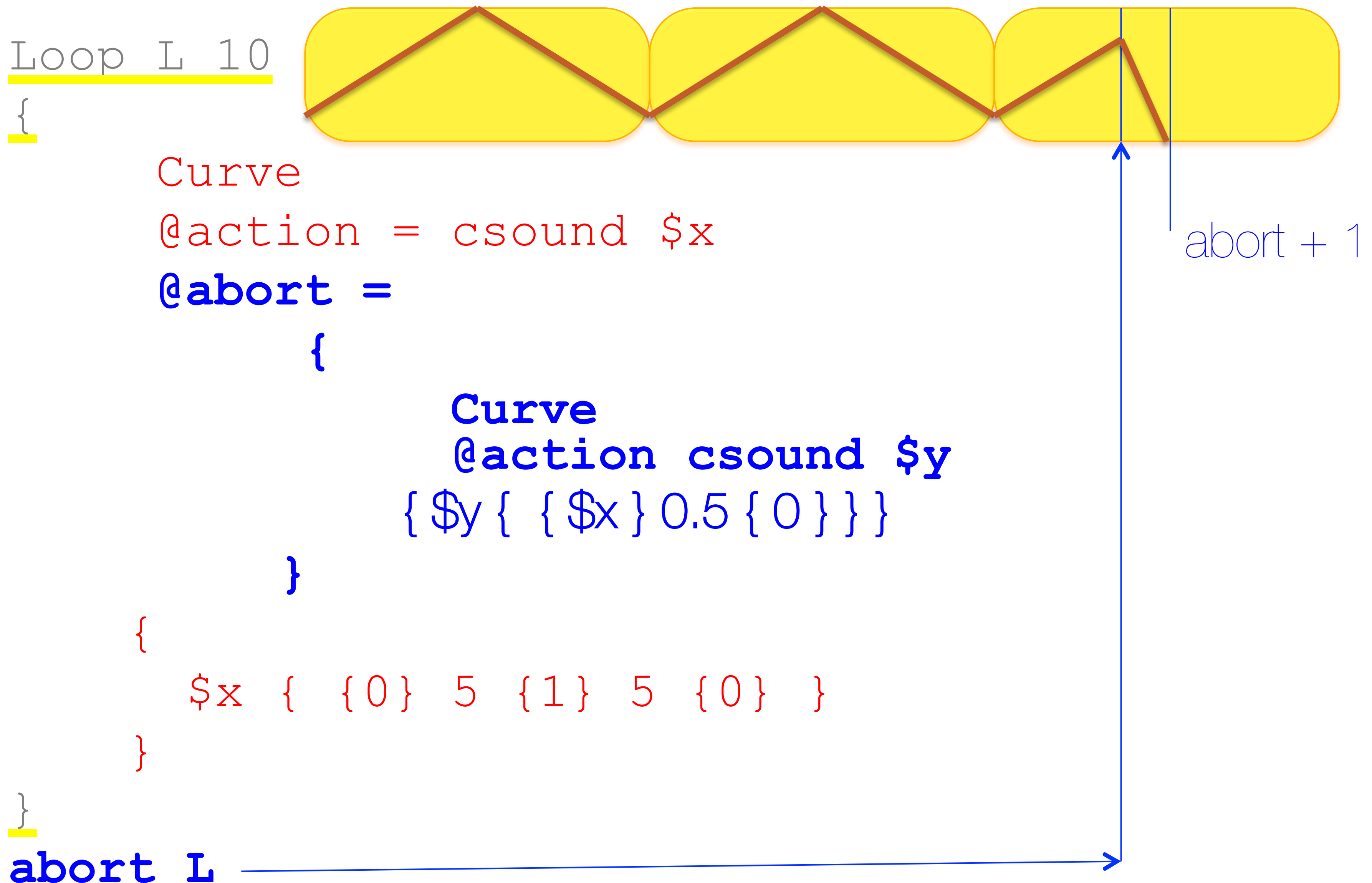
Jean-Louis Giavitto / IRCAM UMR 9912 STMS – CNRS – INRIA équipe MUTANT

Exemple : le fade d'un processus contrôlé par une curve



```
Loop L 10
{
      Curve
      @action = csound $x
      @abort =
            {
                  Curve
                  @action csound $y
                {$y{ {$x}0.5{0}}}
            }
      {
        $x { {0} 5 {1} 5 {0} }
      }
}
```

Exemple : le fade d'un processus contrôlé par une curve



Processus

1. ce sont des valeurs (comme les fonctions)
2. qu'on peut appeler et le résultat est la création d'une instance de groupe
3. l'appel est soit une expression, soit une action
4. les processus peuvent être récursifs
5. et d'ordre sup

ces points distinguent un processus d'une macro

Processus

```
@Proc_def ::Tic($x) {
   $x print TIC
}
@proc_def ::Toc($x) {
   $x print TOC
}
@proc_def ::Clock($p, $q) {
   :: $p(1)
   :: $q(2)
   3 ::Clock($q, $p)
}

Clock(::Tic, ::Toc)
```

généricité: on peut abstraire sur toute expression

appel calculé (décision, contrôle dynamique)

appel récursif (avec une fonction ça bouclerait)

passage en paramètre (un proc est une valeur comme les autres)

Jean-Louis Giavitto / IRCAM UMR 9912 STMS – CNRS – INRIA équipe MUTANT

Processus comme agent

```
@global $incremente, $decremente  ; canaux de communication

@Proc_def ::P($id)
{
      @local $state

      whenever ($incremente = $id)
      {
            $state := $state + 1
      }

      whenever ($decremente = $id)
      {
            $state ;: $state - 1
      }
}

...
$Jose := ::P("José")  ; instanciation de l'agent "José"
…
$s := $Jose.$state    ; accès à la variable locale de l'instance

$incremente := "José" ; envoi du message « incrémente » à José
$decremente := "José" ; envoi du message « decrémente » à José
```

Jean-Louis Giavitto / IRCAM UMR 9912 STMS – CNRS – INRIA équipe MUTANT

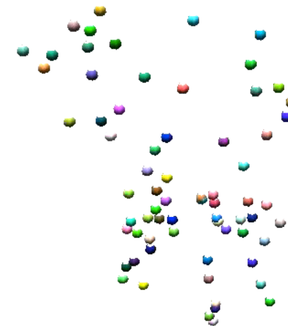
Processus comme agent: Boids

```
@global $go

@Proc_def ::Boids($id)
{
        @local $pos,

        whenever ($go)
        {
                @local $sum, $n
                forall $p in ::Boids {
                        $sum := $p.$pos
                        $n := $n+1
                }
                …
        }
        …
}

$go := true
```



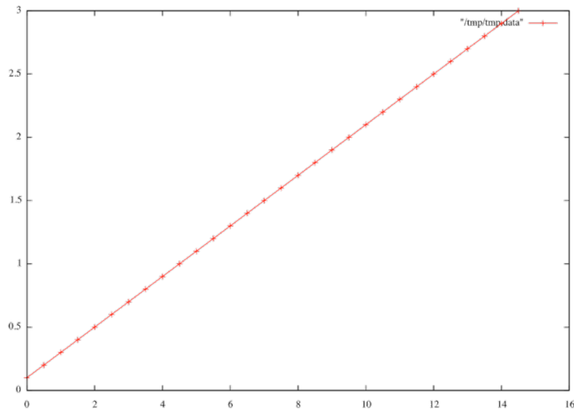
Processus et tempo

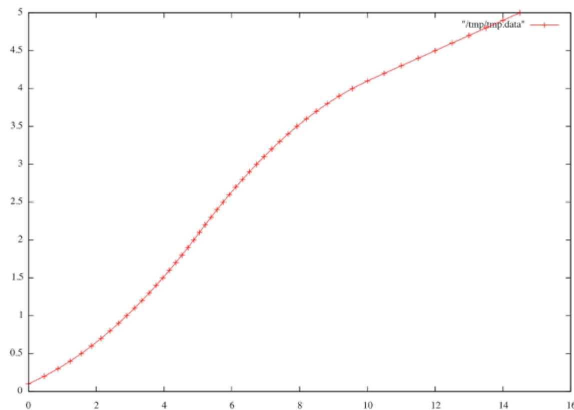
- Un processus hérite du tempo de l'endroit où il est appelé

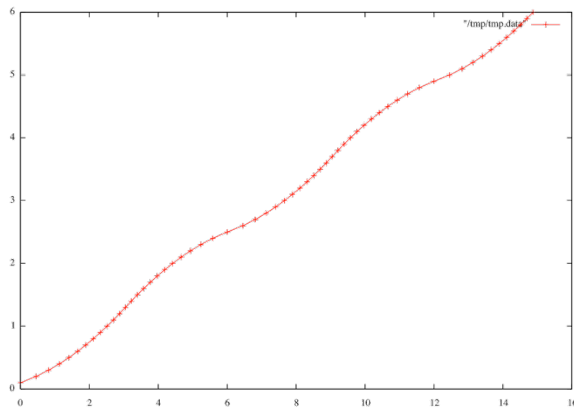
```
Group G1 @tempo := 60
{ Clock(::Tic, ::Toc) }

Group G2 @tempo := 120
{ Clock(::Tic, ::Toc) }
```

Exemple d'empilement de tempi








```
@proc_def ::Trace()
{
        @local $x
        $x := 0
        Loop L 0.5
        {
            $x := $x + 0.1
            plot $NOW " " $x "\n"
        } during [15]
}

$trace1 := ::Trace()

Curve C1 @grain 0.05s
{   $t1 { {60} 5 {180} 5 {60} } }

Group G1 @tempo := $t1
{
    $trace2 := ::Trace()

    Curve C2 @grain 0.05s
    { $t2 {   {60} 3 {180} 3 {60} 3 {180}
              3 {60} 3 {180} 3 {60} }
    }

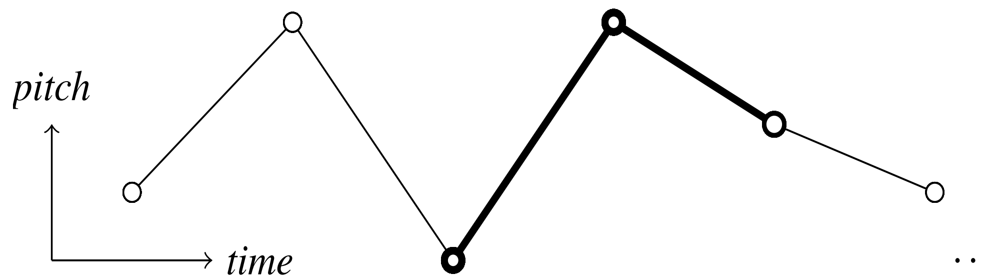
    Group G3 @tempo := $t2
    {
        $trace3 := ::Trace()
    }
}
```

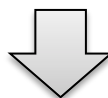
Jean-Louis Giavitto / IRCAM UMR 9912 STMS – CNRS – INRIA équipe MUTANT

PATTERN

Jean-Louis Giavitto / IRCAM UMR 9912 STMS – CNRS – INRIA équipe MUTANT

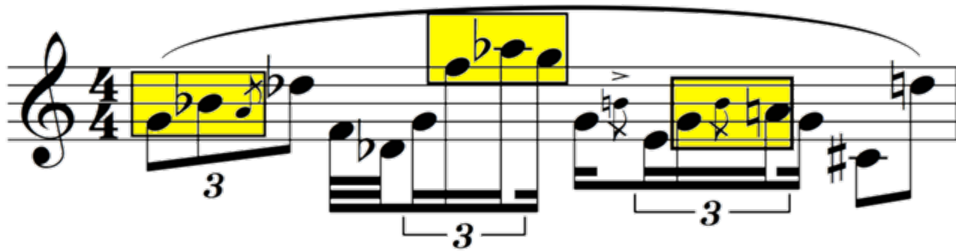
Neume

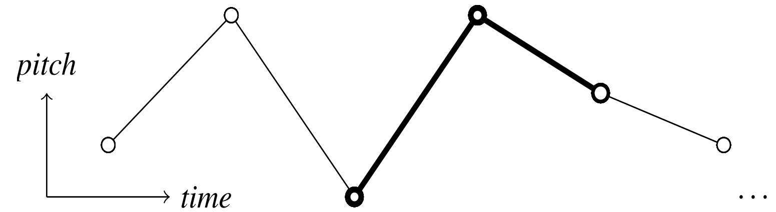




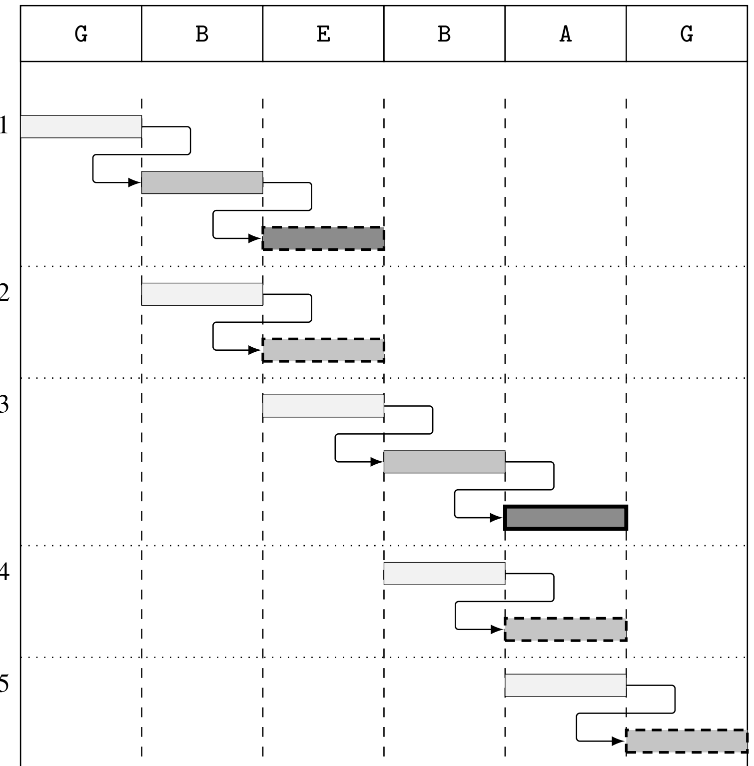
Jean-Louis Giavitto / IRCAM UMR 9912 STMS – CNRS – INRIA équipe MUTANT

Matching a temporal pattern






```
1  whenever ($PITCH) {
2    @local $x
3    $x := $PITCH
4    whenever ($PITCH > $x) {
5      @local $y
6      $y := $PITCH
7      whenever ($PITCH<$y & $PITCH>$x) {
8        @local $z
9        $z := $PITCH
10       a
11     } during [1#]
12   } during[1#]
13 }
```



Jean-Louis Giavitto / IRCAM UMR 9912 STMS – CNRS – INRIA équipe MUTANT

THE PATTERN LANGUAGE: STATE & EVENT

Jean-Louis Giavitto / IRCAM UMR 9912 STMS – CNRS – INRIA équipe MUTANT

NOTE: checking an instantaneous property on $PITCH

```
Pattern P
{
    @local $x , $y , $z
    NOTE $x
    NOTE $y where $x < $y
    NOTE $z where ($y > $z) & ($z > $x)
}
…
whenever P
{ print  "I just saw a P" }
```

Event: checking an instantaneous property

```
Pattern P
{
    @local $x , $y , $z
    Event $PITCH value $x
    Event $PITCH value $y where $x < $y
    Event $PITCH value $z where ($y > $z) & ($z > $x)
}
…
whenever P
{ print  "I just saw a P" }
```

"temporal scope": specifying the succesion

```
Pattern P
{
    @local $x , $y , $z
    Event $PITCH value $x
    Event $PITCH value $y where $x < $y
    Event $PITCH value $z where ($y > $z) & ($z > $x)
}
…
whenever P
{ print  "I just saw a P" }
```

```
@pattern twice
{
  @local $v
  Event $V value $v
  Before [3] Event $V value $v
}
```

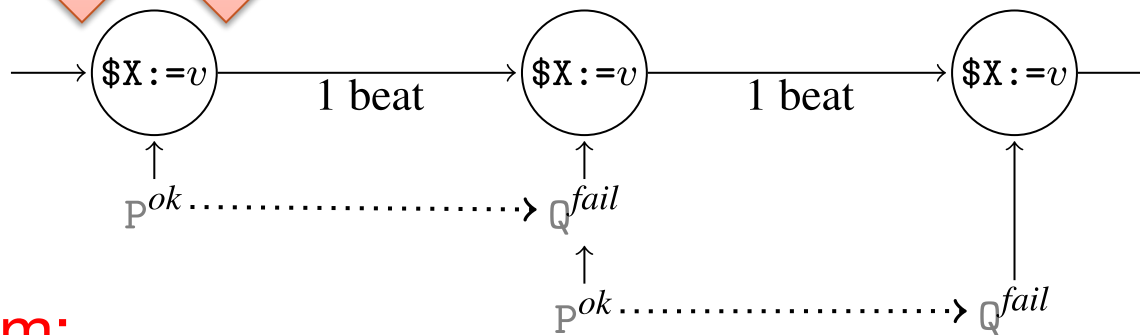
```
before [3#]
before [3s]
```

Jean-Louis Giavitto / IRCAM UMR 9912 STMS – CNRS – INRIA équipe MUTANT

State: a property that lasts

variable `$X` takes the value v at least for 2 beats

```
@local [illegible]ta[illegible] $stop, $w
P: Eve[illegible]CH value v at $start
Q: Even[illegible]CH value $w at $stop where ($stop - $start) ≥ 2
```



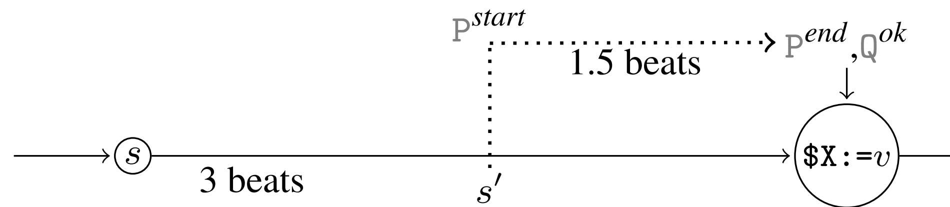
Problem:
they can be an unbounded number of events in the interval

```
State $X where ($X == v) during 2
```

Jean-Louis Giavitto / IRCAM UMR 9912 STMS – CNRS – INRIA équipe MUTANT

Lasting properties do not start everywhere

```
State $X where true during[1.5]
Event $X where ($X == v)
```



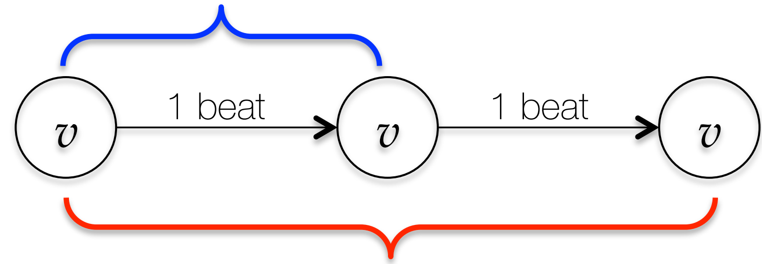
- Continuous time but only a discrete number of events
- Implementation require either
  - a sampling of continuous time (and the start of a potential match at each sampled instant)
  - or the access of all past states (*i.e.* an unbounded memory)

Jean-Louis Giavitto / IRCAM UMR 9912 STMS – CNRS – INRIA équipe MUTANT

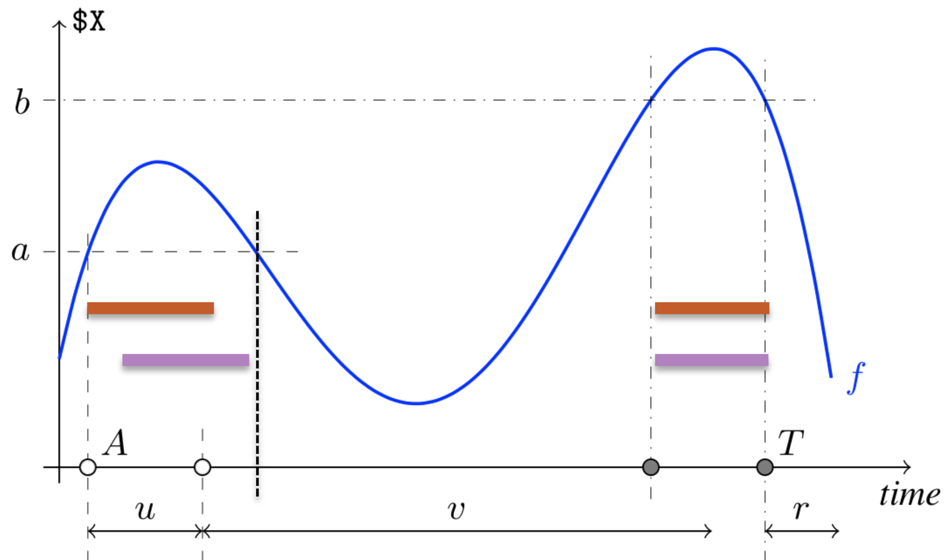
Which match ?

- Earliest match

```
@pattern TwiceIn3B {
  @local $v
  Event $V value $v
  Before[3] Event $V value $v
}
```

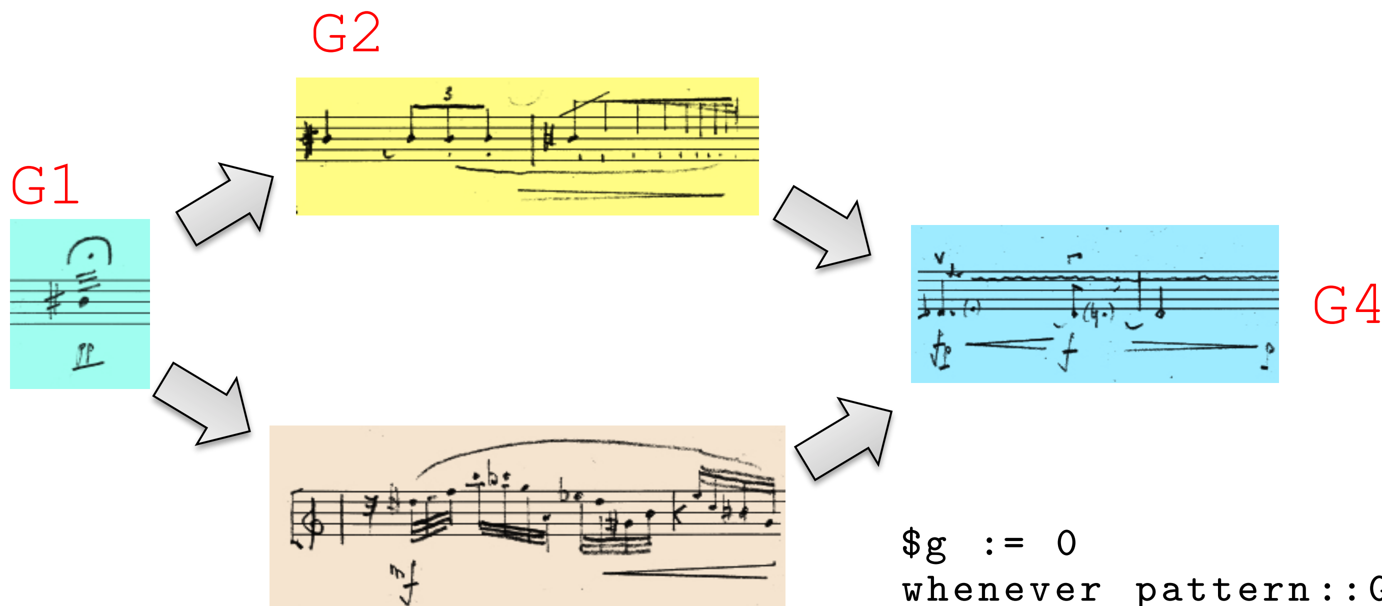


- Refractory period



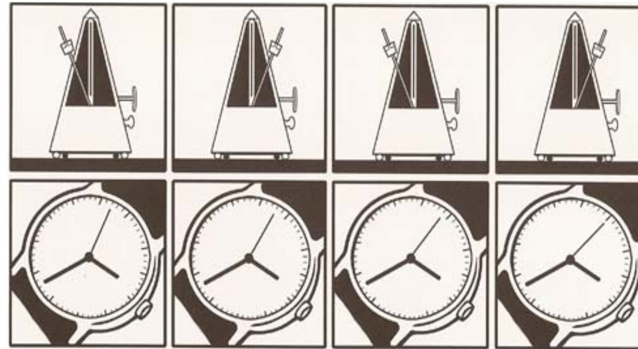
Jean-Louis Giavitto / IRCAM UMR 9912 STMS – CNRS – INRIA équipe MUTANT

Composing and Chaining Patterns




```
$g  := 0
whenever pattern::G1 { $g := 1 }
whenever pattern::G2 { $g := 2 }
whenever pattern::G3 { $g := 3 }
whenever pattern::G4 { $g := 4 }

@pattern Gseq {
  Event $g value 1
  Event $g where ($g==2)||($g==3)
  Event $g value 4
}
...
whenever pattern::Gseq { ... }
```



STRUCTURES DE DONNÉES
(MAP, TAB, NIM)

Jean-Louis Giavitto / IRCAM UMR 9912 STMS – CNRS – INRIA équipe MUTANT

data-ification

donnée temporelles →données en mémoire
données en mémoire → calcul → données en mémoire
données en mémoire → données temporelles

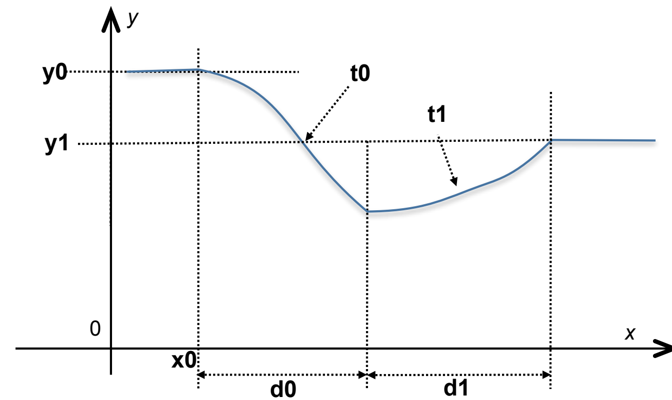
- données temporelles
  = une suite dans le temps de données "instantanées"
- donnée en mémoire
  = structure de données
- *recorder* : donnée temporelles →données en mémoire
- *interpreter* : données en mémoire → données en mémoire
- *player* : données en mémoire → données temporelles

Jean-Louis Giavitto / IRCAM UMR 9912 STMS – CNRS – INRIA équipe MUTANT

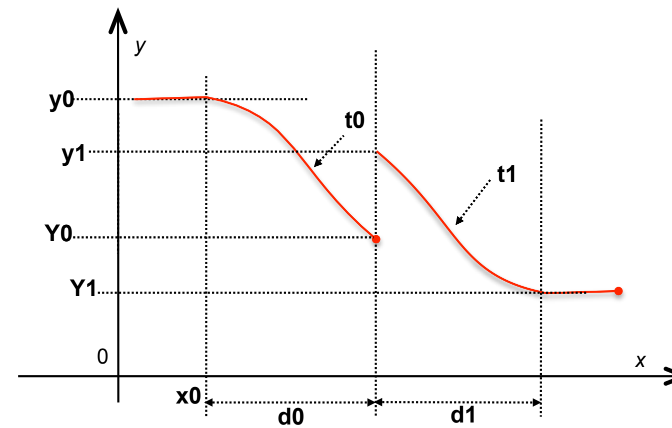
Exemple : les nim

- Curve / Nim

```
NIM { x₀ y₀, d₁ y₁ "cubic"
            , d₂ y₂
            , d₃ y₃ "bounce"
            , ...
            , dₙ yₙ "typeₙ"
}
```




```
NIM { x₀, y₀ d₀ Y₀ "cubic"
        , y₁ d₁ Y₁
        , y₂ d₂ Y₂ "bounce"
        , ...
}
```



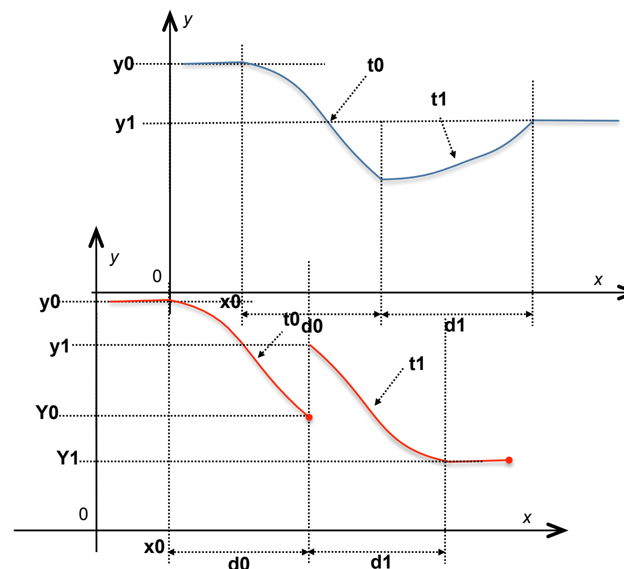
Jean-Louis Giavitto / IRCAM UMR 9912 STMS – CNRS – INRIA équipe MUTANT

Exemple : les nim

- Une NIM est une fonction comme une autre
- Calculer avec les nim:
  - demander la valeur de la NIM en un point donné
  - rajouter un breakpoint en tête
  - rajouter un breakpoint à la fin
  - "jouer" la NIM dans le temps

```
NIM { x_0 y_0, d_1 y_1 "cubic"
             , d_2 y_2
             , d_3 y_3 "bounce"
             , ...
             , d_n y_n "type_n"
    }

NIM { x_0, y_0 d_0 Y_0 "cubic"
         , y_1 d_1 Y_1
         , y_2 d_2 Y_2 "bounce"
         , ...
    }
```



NIM vectorielle

```
NIM{ [-1, 0] [0, 10], [2, 3] [1, 20] ["cubic", "linear"] }
```

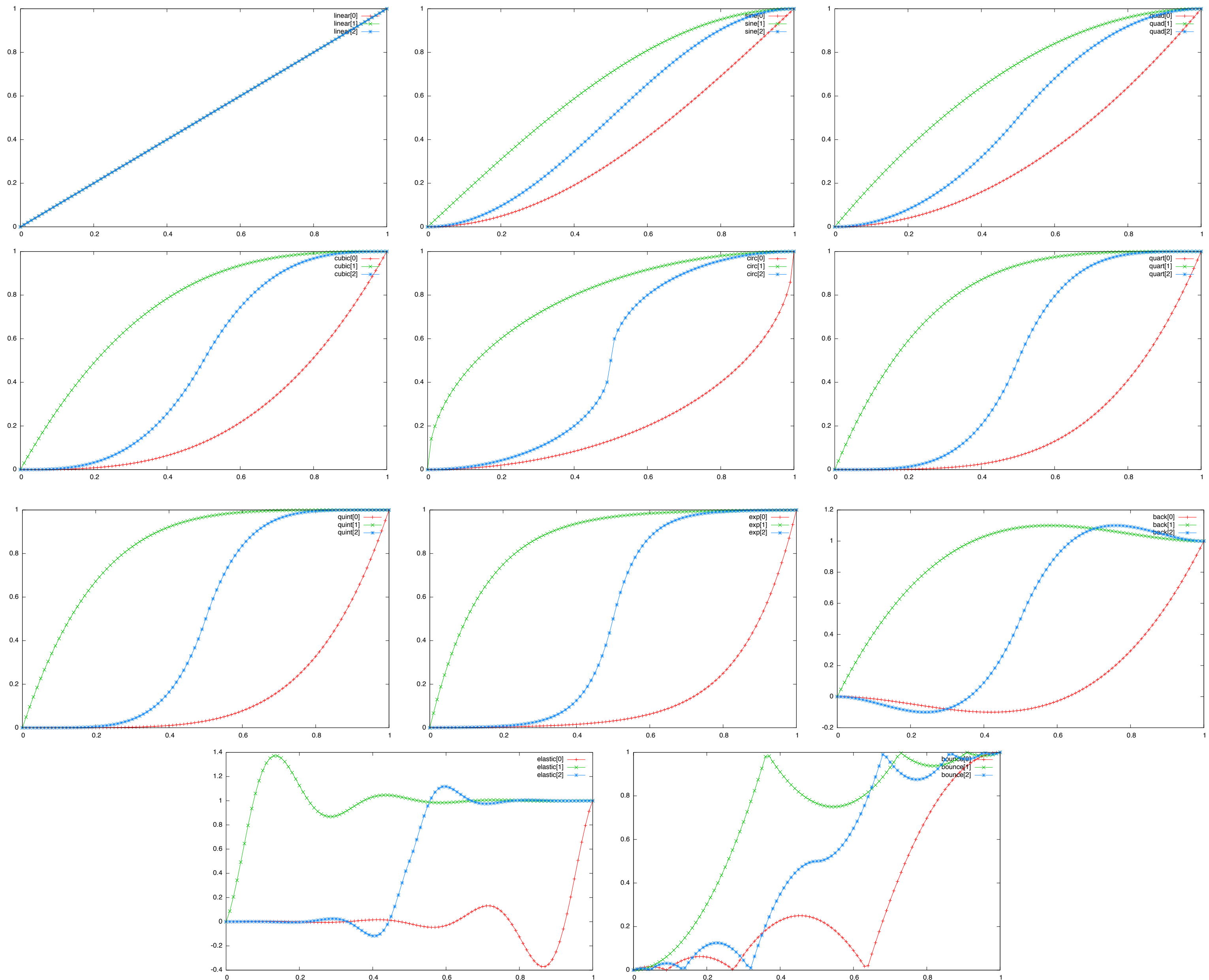
$$\vec{f}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} f_1(x_1) \\ f_2(x_2) \end{pmatrix}$$

```
NIM{ 0, 0, [1, 2] 10 }
```

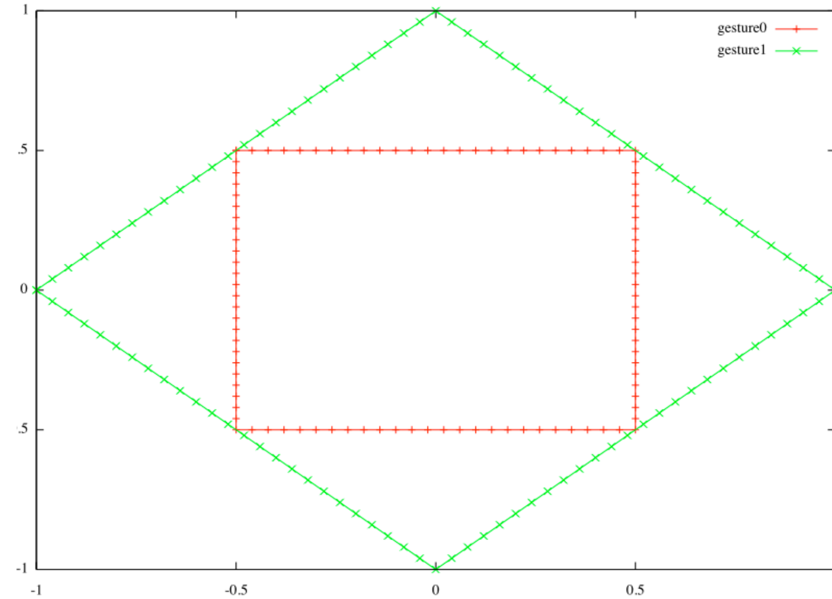
$$\vec{f} = \begin{pmatrix} f_1 \\ f_2 \end{pmatrix} \quad \text{where } f_1(x) = \begin{cases} 0, & \text{if } x < 0 \\ 10, & \text{if } x > 1 \\ 10x, & \text{elsewhere} \end{cases} \quad \text{and } f_2(x) = \begin{cases} 0, & \text{if } x < 0 \\ 10, & \text{if } x > 2 \\ 5x, & \text{elsewhere} \end{cases} .$$

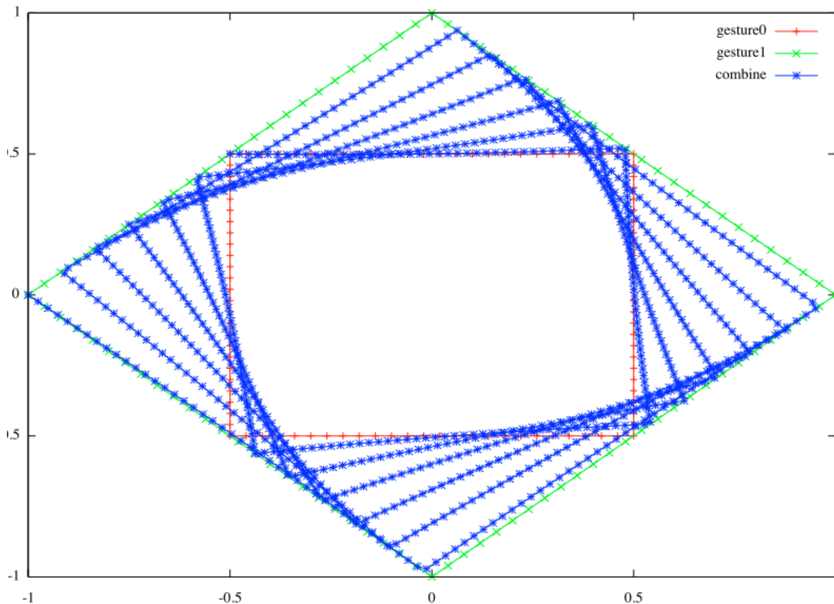
$$f(t) = \begin{bmatrix} f_1(t) \\ f_2(t) \end{bmatrix} \qquad f\begin{bmatrix} t_1 \\ t_2 \end{bmatrix} = \begin{bmatrix} f_1(t_1) \\ f_2(t_2) \end{bmatrix}$$

NIM: type d'interpolation



Interpolation d'un mouvement






```
$gesture0 := NIM { 0 [-0.5, 0.5], 1 TAB[0.5, 0.5]
                              , 1 TAB[0.5, -0.5]
                              , 1 TAB[-0.5, -0.5]
                              , 1 TAB[-0.5, 0.5] }

$gesture1 := NIM { 0 [-1, 0], 1 TAB[0, 1]
                          , 1 TAB[1, 0]
                          , 1 TAB[0, -1]
                          , 1 TAB[-1, 0] }

@fun_def @combine($t, $p) {
        (1 - $p) * $gesture0($t)
   +         $p * $gesture1($t)
}
```

```
Curve
@grain = 0.01
@action = {
   $tmp := @combine($pos % 4, $pos/24)
   $x := $tmp[0]
   $y := $tmp[1]
}
{  $pos {    {0} 10 {24} } }
```


Jean-Louis Giavitto / IRCAM UMR 9912 STMS – CNRS – INRIA équipe MUTANT

Fonction, dictionnaire et fonctions interpolées

- ```
  @fun_def fact($x) {
        ($x == 0 ? 0 : $x * @fact($x – 1))
  }
  @fact(3)
  ```

- ```
  $dico := MAP { (clé, valeur) }
  $dico("José")
  ```

- ```
  $courbe := NIM { … }
  $courbe(1.0)
  ```
  ```

Fonctions prédéfinies et utilisateurs @fun_def

- Valeur de premier ordre
  @fun_def apply($f, $x) { $f($x) }
- Curryfiée
  @fun_def plus($a, $b) { $a + $b }
  $incr := @plus(1)
- Récursive
- Dynamiquement typées
- Librairie de ~160 fonctions prédéfinies
- Pas d'argument optionnels
  ou de nombre d'arguments variables

Librairie

approx, asin, atan, between, bounded_integrate_inv, bounded_integrate, car, cdr, ceil, clear, concat, cons, copy, cosh, cos, count, dim, domain, empty, exp, explode, find, flatten, floor, gnuplot, gshift_map, history_map, history_tab, history_map_date, history_tab_date, insert, insert, integrate, iota, is_bool, is_defined, is_fct, is_float, is_function, is_integer_indexed, is_interpolatedmap, is_int, is_list, is_map, is_numeric, is_prefix, is_string, is_subsequence, is_subsequence, is_suffix, is_symbol, is_undef, is_vector, lace, listify, log10, log2, log, make_duration_map, make_score_map, map, map_compose, map_concat, map_normalize, map_reverse, mapval, max_key, max_val, max, member, merge, min_key, min_val, min, normalize, occurs, permute, plot, pow, push_back, push_front, rand_int, random, rand, reduce, range, remove, remove_duplicate, replace, reshape, resize, reverse, rnd_bernouilli, rnd_binomial, rnd_exponential, rnd_gamma, rnd_geometric, rnd_normal, rnd_uniform_int, rnd_uniform_float, rotate, round, rplot, scan, scramble, select_map, shape, shift_map, sinh, sin, size, sort, sort, sputter, sqrt, stutter, system, tan

VECTEUR
(TAB)

Jean-Louis Giavitto / IRCAM UMR 9912 STMS – CNRS – INRIA équipe MUTANT

Tableau

- Vecteur indexé de 0 à n

- Vecteur hétérogène:
  [0, true, [3.14159, “une chaîne”, @fact], ::Trace()]

- Définition en compréhension (APL, Mathematica, series in CL)
  [*exp* | $it in *source*, *cond*]
  [@rand_int(1) | (10)]
  [2*$it | $it in $tab]
  [$A[$i] + $B[$i] | $i in @size($A), $A[i] > 0]

- Extension implicite
  “quand il faut et que ce n’est pas ambigu”:
  $A + 1, 2*$A, $A + $B, …

Vecteur de vecteurs

```
[ 0 | (100) ]          ; builds a vector of 100 elements, all zeros
[ @random() | (10) ] ; build a vector of ten random numbers
[ $i | $i in 40, $i % 2 == 0 ] ; lists the even numbers from 0 to 40
[ $i | $i in 40 : 2]            ; same as previous
[ 2*$i | $i in (20) ]           ; same as previous

; equivalent to ($s + $t) assuming arguments of the same size
[ $s[$i] + $t[$i] | $i in @size($t) ]

; transpose of a matrix $m
[ [$m[$j, $i] | $j in @size($m)] | $i in @size($m[0])]

; scalar product of two vectors $s and $t
@reduce(@+, $s * $t)

; matrice*vector product
[ @reduce(@+, $m[$i] * $v) | $i in @size($m) ]

; squaring a matrix $m, i.e. $m * $m
[ [ @reduce(@+, $m[$i] * $m[$j]) | $i in @size($m[$j]) ]
 | $j in @size($m) ]
```

Fonctions prédéfinies sur les tableaux

- ~50 fonctions dans la bibliothèque
  - @permute
  - @sort
  - @lace, @stutter, @sputter … à la SuperCollider
  - Operateur: @map, @reduce (fold), @scan, *etc.*

Un exemple: Processus Markoviens

```
// On définit autant de processus que d'action à lancer,
// autrement dit, que de sommet dans le graphe.
//
// Les arguments du processus sont
//  $x : le numéro de l'étape (le i-ème processus visité).
//  $d : la durée de la visite

@proc_def ::A($x, $d) {
    Loop 1.0 { print "step" $x "visit A at" $NOW "for" $d "beats" }
}

@proc_def ::B($x, $d) {
    Loop 1.0 { print "step" $x "visit B at" $NOW "for" $d "beats" }
}

@proc_def ::C($x, $d) {
    Loop 1.0 { print "step" $x "visit C at" $NOW "for" $d "beats" }
}

@proc_def ::D($x, $d) {
    Loop 1.0 { print "step" $x "visit D at" $NOW "for" $d "beats" }
}
```

```
// On donne pour chaque sommet, les proba de transitions vers
// chaque autre sommet (y compris soit-même).
// On suppose que A correspond à l'index 0, B à 1, etc.
//  La somme des éléments de chaque vecteur doit faire 1.0

$proba_A := tab[ 0.0,  0.2,  0.4,  0.4 ]
$proba_B := tab[ 0.3,  0.1,  0.4,  0.3 ]
$proba_C := tab[ 0.8,  0.2,  0.1,  0.1 ]
$proba_D := tab[ 0.0,  0.5,  0.2,  0.3 ]

// Le temps où l'on reste dans un sommet dépend du prochain
// sommet qui sera visité. Ainsi, quand on est dans A,
//  - on reste 5 temps dans A si on visite B ensuite
//  - on reste 3 temps dans A si on visite C ensuite
//  - on reste 4 temps dans A si on visite D ensuite
// etc.

$waitingA := tab[ 0,  5,  3,  4 ]
$waitingB := tab[ 5,  3,  4,  2 ]
$waitingC := tab[ 3,  4,  2,  2 ]
$waitingD := tab[ 0,  2,  3,  4 ]
```

Jean-Louis Giavitto / IRCAM UMR 9912 STMS – CNRS – INRIA équipe MUTANT

Partie générique

```
// A chaque instant, l'état de la navigation
// dans  le graphe est décrite par les valeurs
// de 3 variables :
// -  $proc
//    processus courant en cours d'exécution
// -  $proc_index
//    son index dans le tableau $actions
// -  $step
//    nème étape de la navigation

@global $proc, $proc_index, $step

// On calcule $pX comme la proba cumulée des
// transition sortant de X parce que cela permet
// simplement de déterminer quelle transition
// prendre en tirant un nombre entre 0 et 1.

$pA := @scan(@+, $proba_A);
$pB := @scan(@+, $proba_B);
$pC := @scan(@+, $proba_C);
$pD := @scan(@+, $proba_D);

// La matrice des transitions cumulées du graphe
$proba := tab[$pA, $pB, $pC, $pD]

// La matrice des temps à passer dans chaque
// sommet en fonction du prochain sommet à
// visiter
$waiting := tab[$waitingA, $waitingB, $waitingC,
$waitingD]

// Un vecteur qui relie un index et une action
$actions := tab[::A, ::B, ::C, ::D]
```

Partie générique (2)

```
@fun_def @sup($cmp, $x) { $x > $cmp }

// Le processus qui navigue dans le graphe
//
// c'est juste un processus récursif.
// Il met à jour l'état de la visite, tue l'action
// courante du sommet visité et lance l'action du
// prochain sommet visité.
// L'argument $id est juste le numéro de l'étape

@proc_def ::run($id)
{
  @local $next_index, $delay

  $step := $step + 1
  $cmp := @random()
  $next_index := @find_index($proba[$proc_index],
                             @sup($cmp))
  $delay := $waiting[$proc_index, $next_index]

  // On récupère le proc à lancer à partir de son index
  $P := $actions[$next_index]

  // On tue le processus courant
  abort $proc

  print "    "

  // On lance le prochain processus
  $proc_index := $next_index
  $proc := $P($step, $delay)

  // Et on se relance après une attente de $delay
  $delay ::run($id + 1)
}

// --- Lancement --------------

// il faut amorcer la pompe.
// On le fait en lançant le processus A
$proc_index := 0
$proc := ::A(0, 0)
$step := 1

::run(1)  // lance la navigation dans le graphe
```

Une trace

```
antescofo~ - Score loaded succesfully with 0 events and 20 actions.
antescofo~ - Sequence playback... .
print: step 0 visit A at 0. for 0 beats
print:
print: step 2 visit D at 0. for 4 beats
print: step 2 visit D at 1. for 4 beats
print: step 2 visit D at 2. for 4 beats
print: step 2 visit D at 3. for 4 beats
print:
print: step 3 visit C at 4. for 3 beats
print: step 3 visit C at 5. for 3 beats
print: step 3 visit C at 6. for 3 beats
print:
print: step 4 visit B at 7. for 4 beats
print: step 4 visit B at 8. for 4 beats
print: step 4 visit B at 9. for 4 beats
print: step 4 visit B at 10. for 4 beats
print:
print: step 5 visit D at 11. for 2 beats
print: step 5 visit D at 12. for 2 beats
print:
print: step 6 visit B at 13. for 2 beats
print: step 6 visit B at 14. for 2 beats
print:
print: step 7 visit D at 15. for 2 beats
print: step 7 visit D at 16. for 2 beats
print:
print: step 8 visit B at 17. for 2 beats
print: step 8 visit B at 18. for 2 beats
print:
print: step 9 visit A at 19. for 5 beats
print: step 9 visit A at 20. for 5 beats
print: step 9 visit A at 21. for 5 beats
print: step 9 visit A at 22. for 5 beats
print: step 9 visit A at 23. for 5 beats
print:
print: step 10 visit C at 24. for 3 beats
print: step 10 visit C at 25. for 3 beats
antescofo~ - Stopped!
```

LA SUITE ?

Directions de travail (on a besoin de vous)

- Documentation (tutorial, ref, how-to, FAQ, partitions en vraie grandeur…)
- Bétonner et simplifier la syntaxe
- Notation et mécanismes expressifs et lisibles (comment simplifier les grandes partitions)
  - Référer symboliquement aux événements partagés entre deux cadres temporels
  - Permettre les tableaux dans les Curves (comme pour les NIM)
  - Dictionnaire en compréhension
  - D'autres fonctions d'estimation de tempo (et sur tout type d'événement)
  - Tempo symbolique
  - …
- Librairies spécialisées, plugin
- Nouvelles structure de données (ex.: buffer audio)
- Standalone
- Marier pattern et machine d'écoute
- …