tScore
How To Notate Time-Based Anything

Markus Lepper Baltasar Trancón y Widemann

What are tScore and translets ?

tScore is

a text format

(and the corresponding processing software framework)

for denotating arbitrary time-based data.

(translets are a library of parser combinators.)

What are tScore and translets ?

tScore is

a text format

(and the corresponding processing software framework)

for denotating arbitrary time-based data.

(translets are a library of parser combinators.)

Domain experts have developed very different tabular formats for Time

- In very different domains *tabular formats* exist for denotating *time related / diachronic data*
- These formats are in between *text* and *graphics*
- Called "symbolic vs. analog" by GOODMAN
- Tending more to one side or the other
- Eg. railways schedules, cue lists, kinetographics
- Most complex: tradit. sheet music
  = "Common Western Notation" = CWN ⊃ orchestral score
- Non-trivial computer interface

Domain experts have developed very different tabular formats for Time

- In very different domains *tabular formats* exist for denotating *time related / diachronic data*
- These formats are in between *text* and *graphics*
- Called "symbolic vs. analog" by GOODMAN
- Tending more to one side or the other
- Eg. railways schedules, cue lists, kinetographics
- Most complex: tradit. sheet music
  = "Common Western Notation" = CWN ⊃ orchestral score
- Non-trivial computer interface

Domain experts have developed very different tabular formats for Time

- In very different domains *tabular formats* exist for denotating *time related / diachronic data*
- These formats are in between *text* and *graphics*
- Called "symbolic vs. analog" by GOODMAN
- Tending more to one side or the other
- Eg. railways schedules, cue lists, kinetographics
- Most complex: tradit. sheet music
  = "Common Western Notation" = CWN $\supset$ orchestral score
- Non-trivial computer interface

Need of a common notation and a generic infrastructure

A common format $\left\{\begin{matrix}\text{readable}\\ \text{writable}\end{matrix}\right\}$ by $\left\{\begin{matrix}\text{computers}\\ \text{humans}\end{matrix}\right\}$

for denotating arbitrary parameter ranges

in arbitrary time domains

is strongly desired !

- Closely following well-established domain concepts
- Realized by a generic infrastructure

Need of a common notation and a generic infrastructure

A common format $\left\{\begin{matrix}\text{readable}\\ \text{writable}\end{matrix}\right\}$ by $\left\{\begin{matrix}\text{computers}\\ \text{humans}\end{matrix}\right\}$

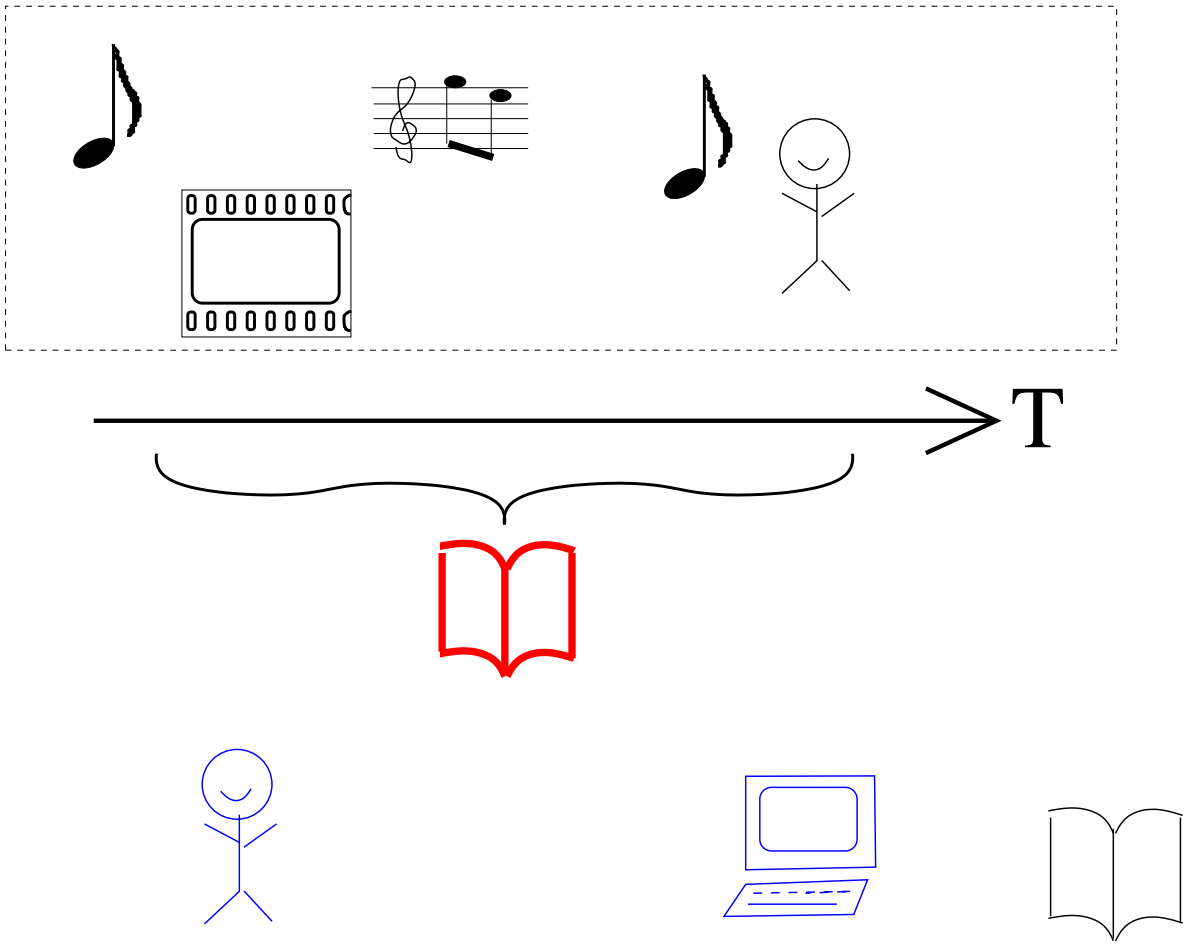
for denotating arbitrary parameter ranges

in arbitrary time domains

is strongly desired !

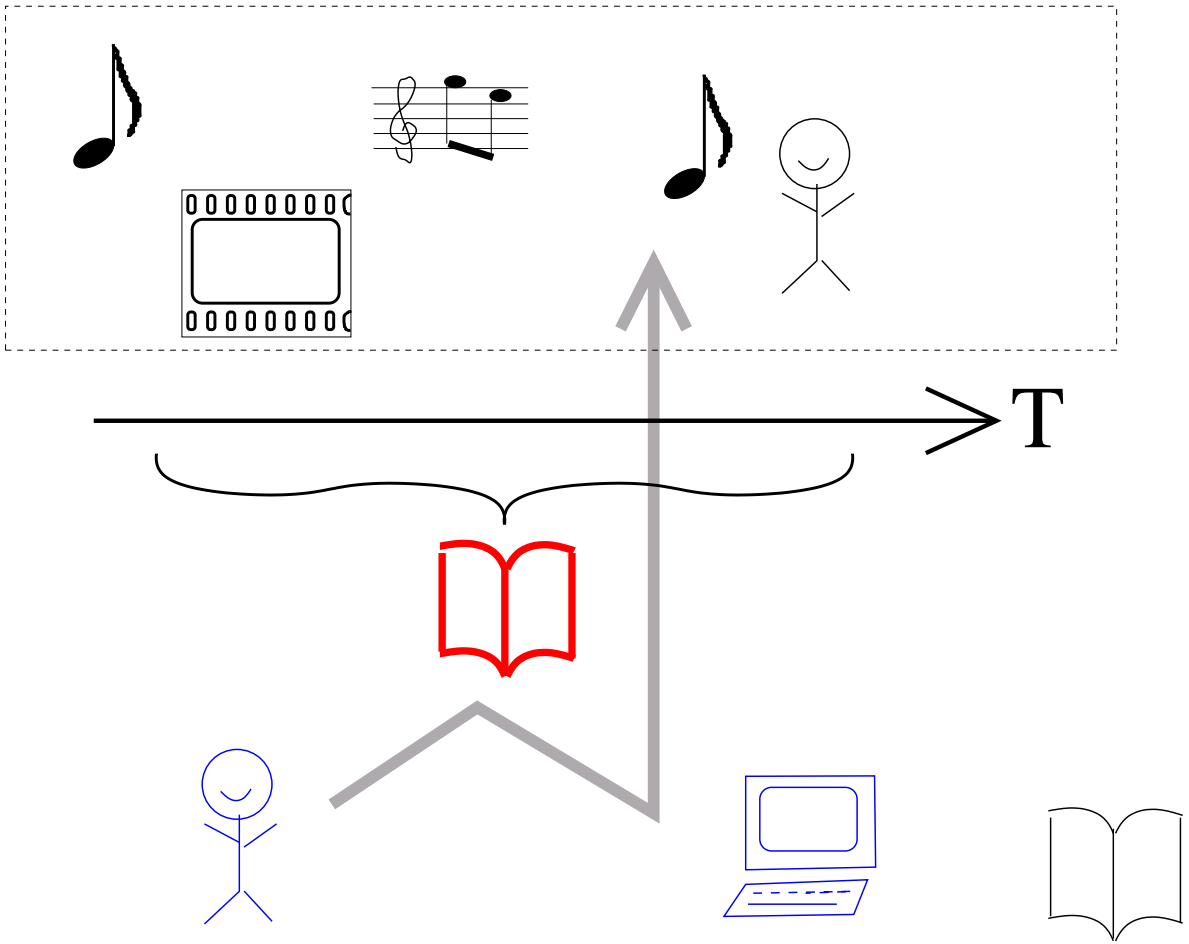
- Closely following well-established domain concepts
- Realized by a generic infrastructure

Use cases in Stage Productions



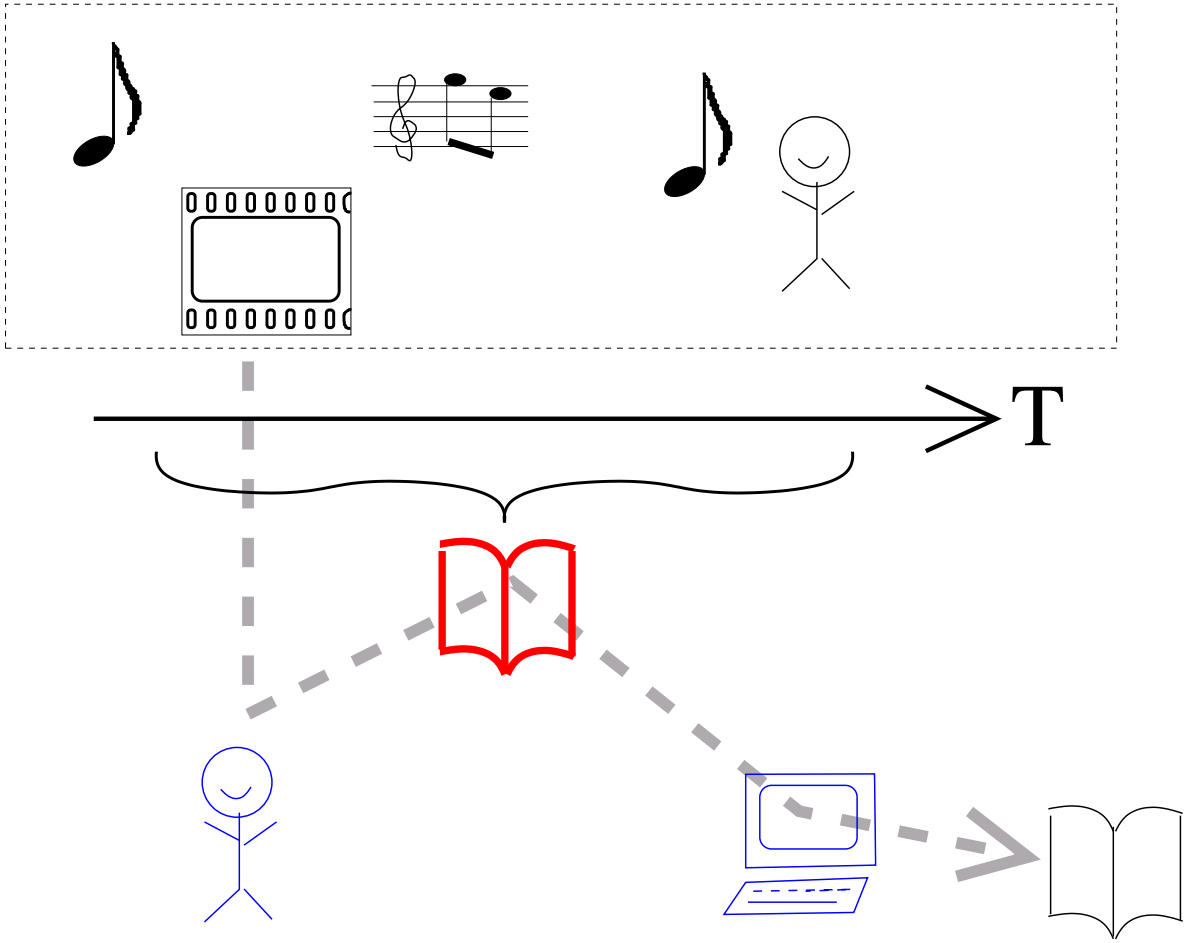
Use cases in Stage Productions

Human writes score and computer executes



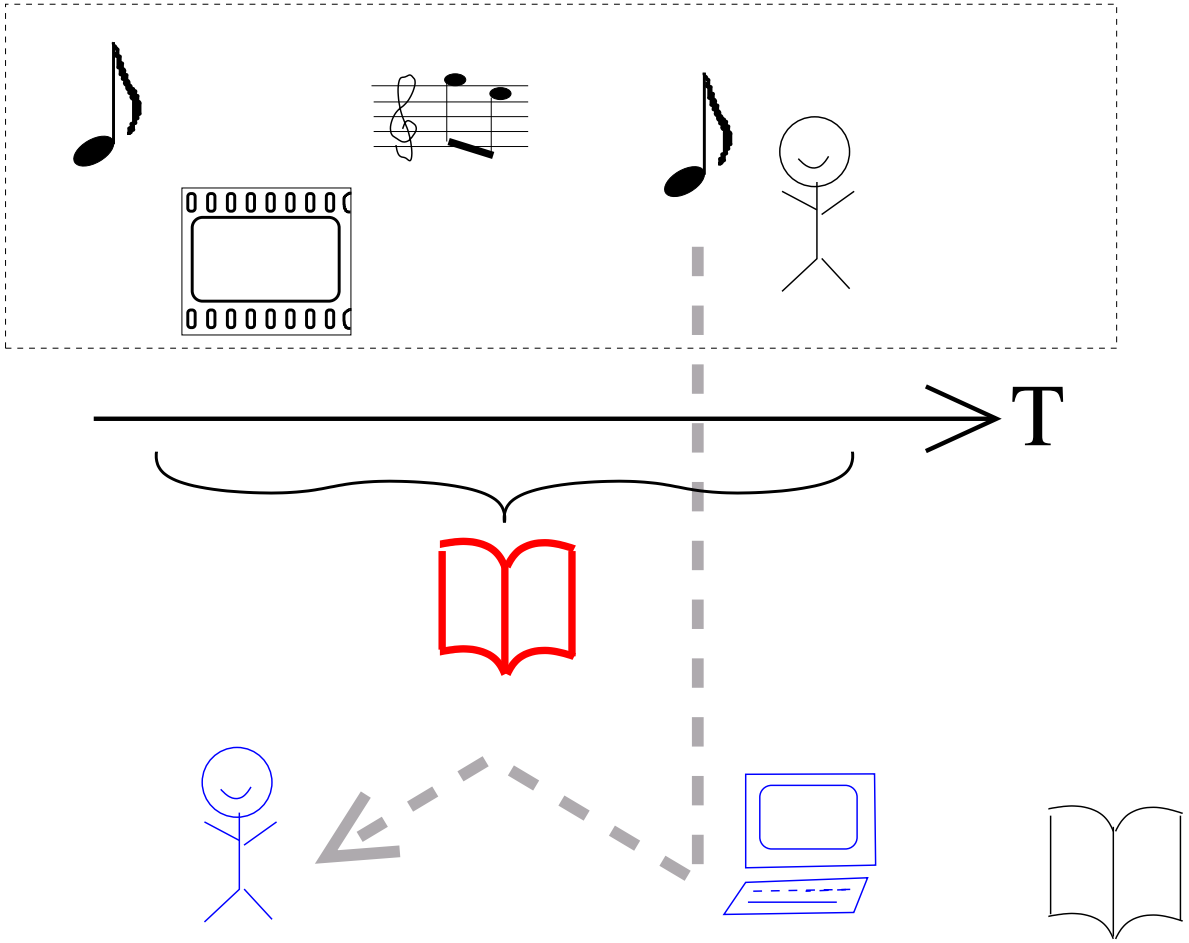
Use cases in Stage Productions

Human writes protocol and computer renders



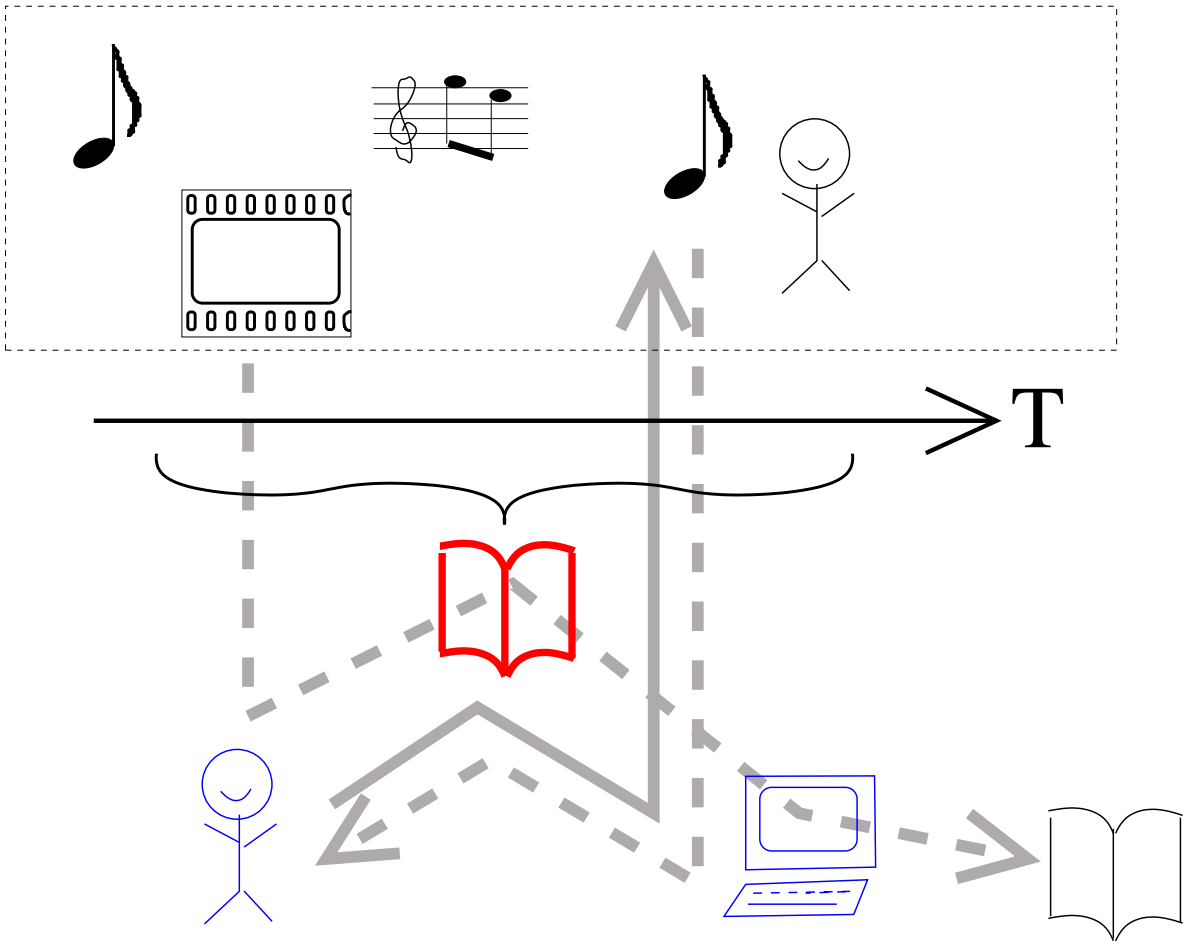
Use cases in Stage Productions

Computer recognizes and writes readable protocol



Use cases in Stage Productions

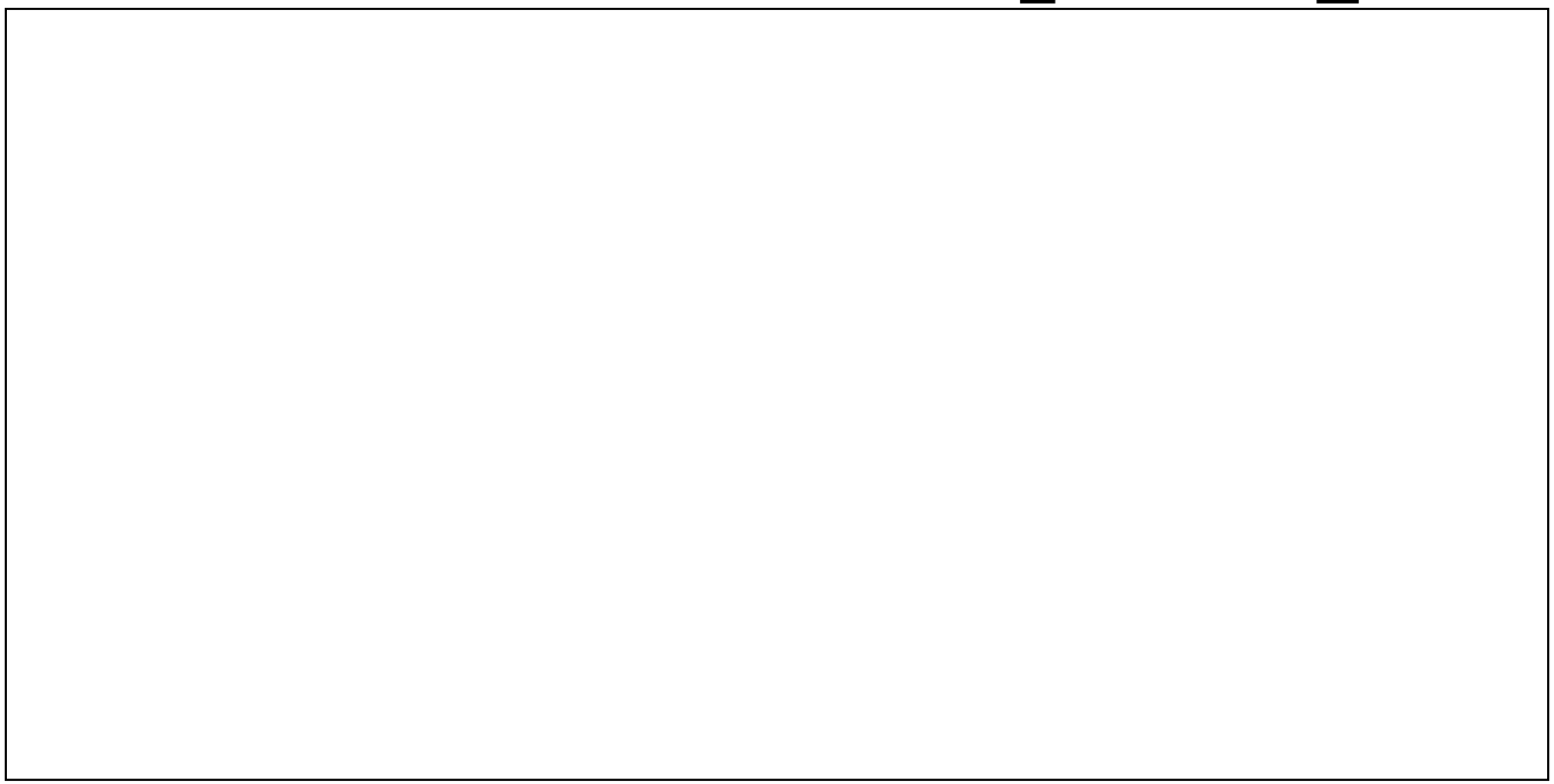
etc.



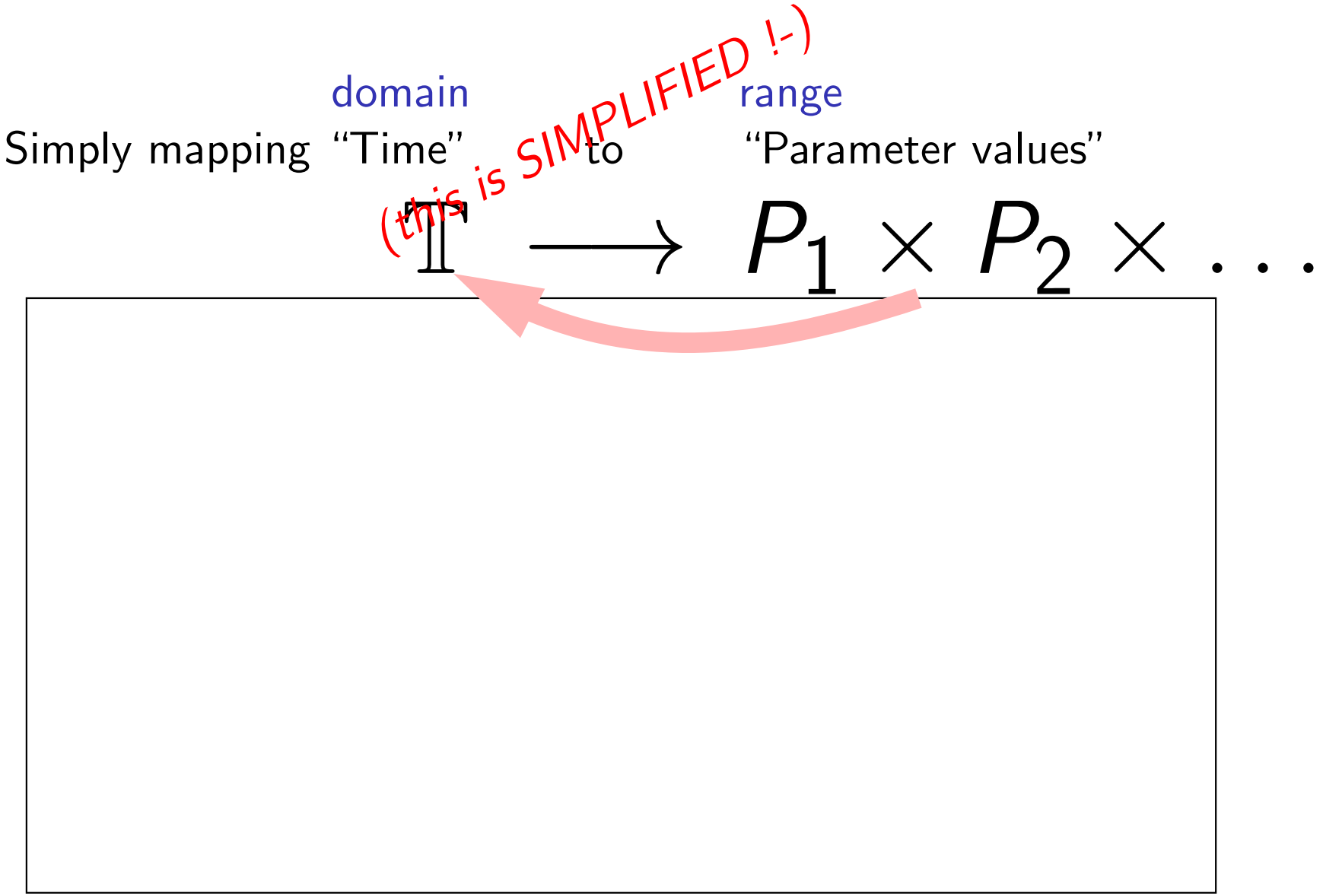
What are the contents of time notation?

Simply mapping “Time” (domain) to “Parameter values” (range)

$$\mathbb{T} \longrightarrow P_1 \times P_2 \times \ldots$$



What are the contents of time notation?



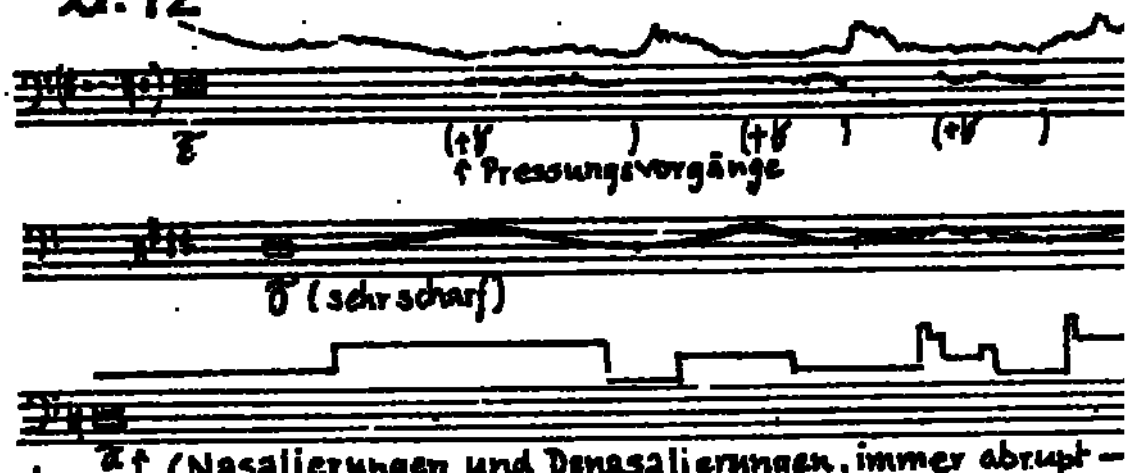
What are the contents of time notation?

Simply mapping "Time" (domain) to "Parameter values" (range)

$$\mathbb{T} \longrightarrow P_1 \times P_2 \times \ldots$$

analog

(pure graphical notation, "Streckennotation")



What are the contents of time notation?

Simply mapping “Time” (domain) to “Parameter values” (range)

$$\mathbb{T} \longrightarrow P_1 \times P_2 \times \ldots$$

analog

```
\notes
\ibl0’ce0\qb0{_e_b}\tbl0\qb0{e}
\ibl0e0\qb0{_g{’_a}}\tbl0\qb0{‘g}
\enotes
```

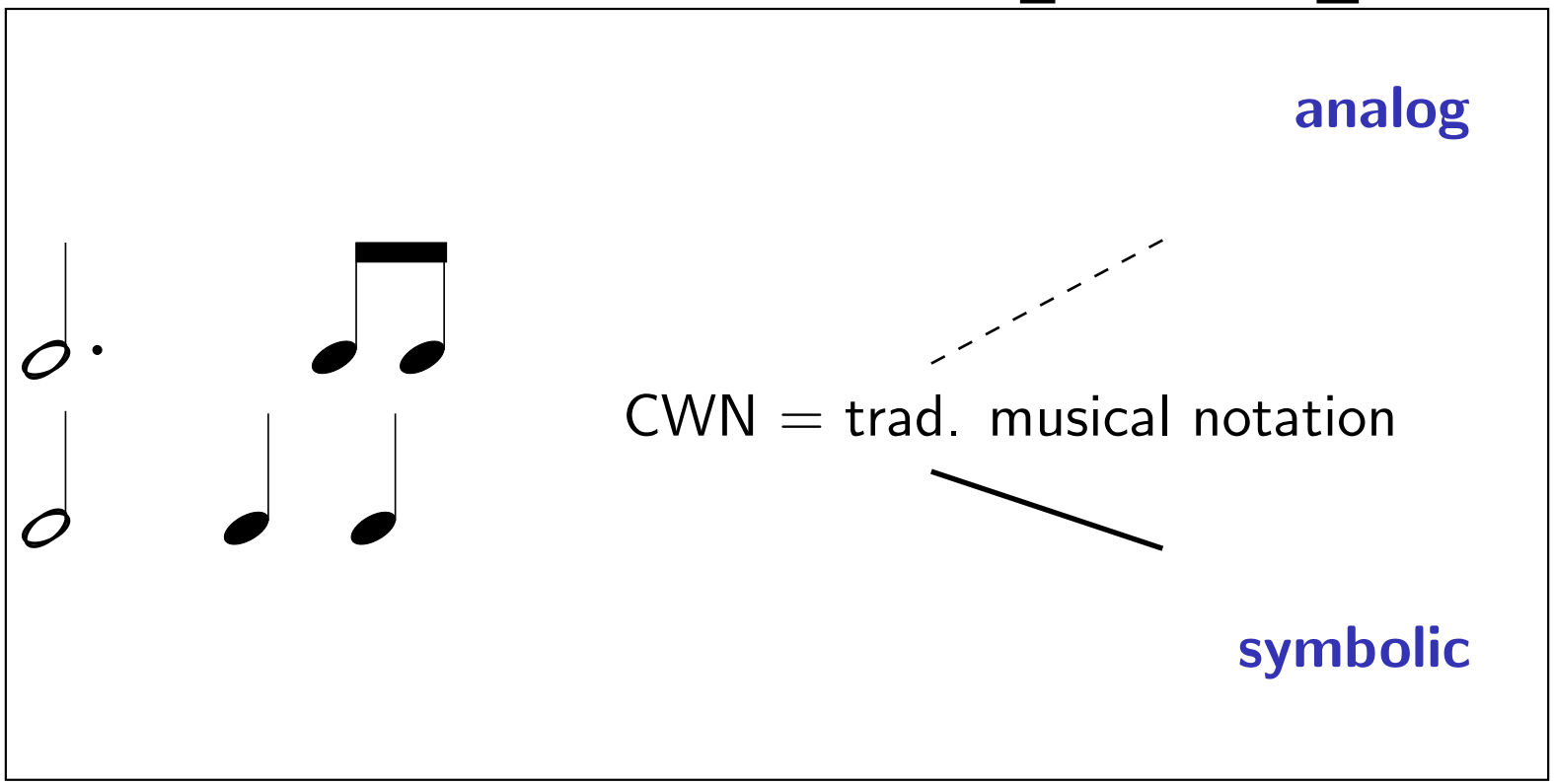
(expressions, as in LilyPond, musixTex, Guido, . . .)

symbolic

What are the contents of time notation?

Simply mapping domain "Time" to range "Parameter values"

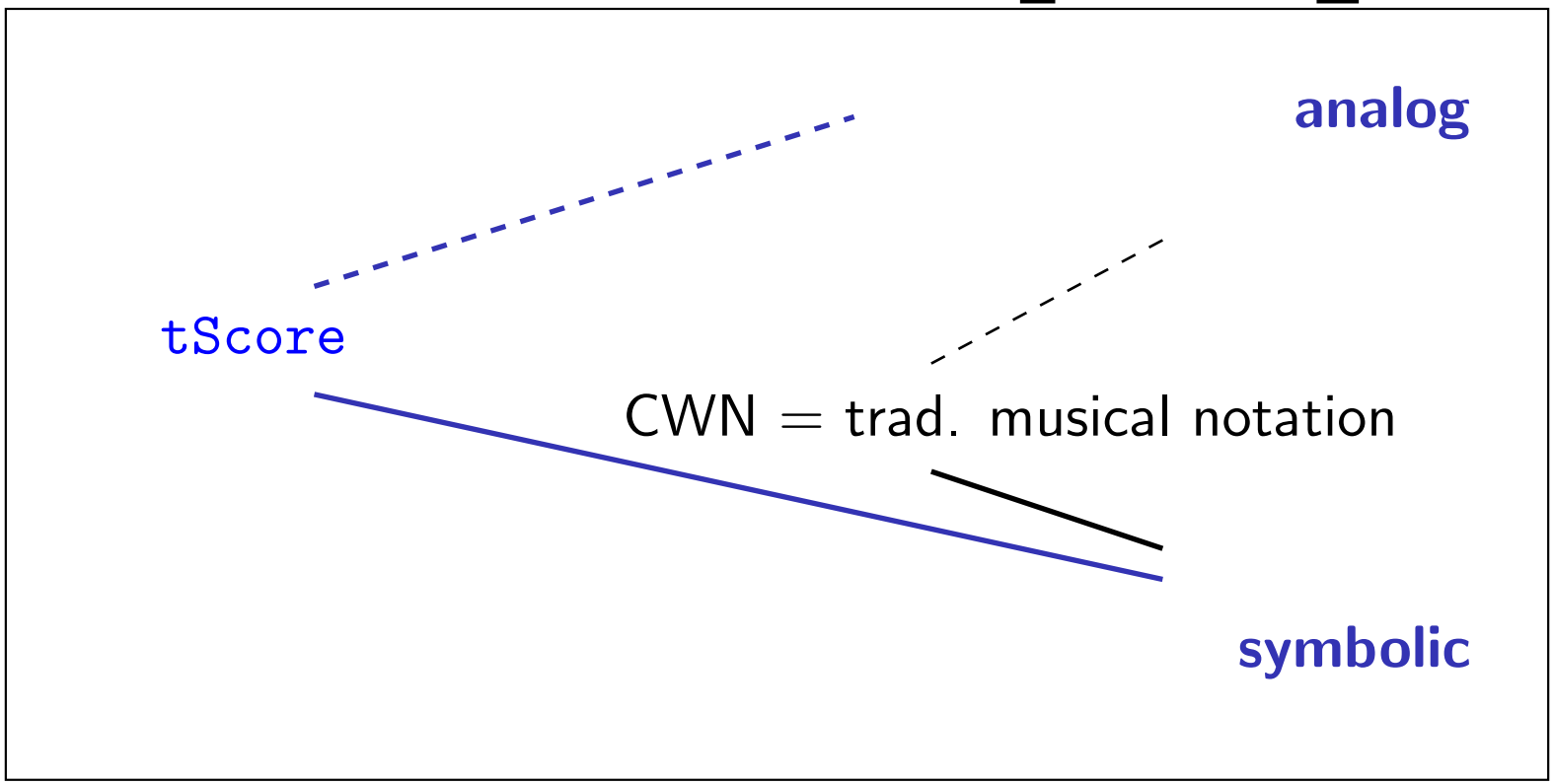
$$\mathbb{T} \longrightarrow P_1 \times P_2 \times \ldots$$



What are the contents of time notation?

Simply mapping “Time” (domain) to “Parameter values” (range)

$$\mathbb{T} \longrightarrow P_1 \times P_2 \times \ldots$$



CWN — Common Western Notation

- ✚ well elaborated, centuries of optimization
- ✚ compact
- ✚ easy readable, “one glance”, *“prima vista”*
- ✚ utmost versatile and adaptable
  (covers Monteverdi to Stockhausen)
- ▬ specialized, restricted range (pitch information)
- ▬ restricted domain (metrical duration values)
- ▬ diversity of variants, partly conflicting (e.g. enharmonics)
- ✚ used in many “foreign” fields (alternative ranges)
- ▬ hardly computer compatible

CWN — Common Western Notation

- ✚ well elaborated, centuries of optimization
- ✚ compact
- ✚ easy readable, “one glance”, *“prima vista”*
- ✚ utmost versatile and adaptable
  (covers Monteverdi to Stockhausen)
- ▬ specialized, restricted range (pitch information)
- ▬ restricted domain (metrical duration values)
- ▬ diversity of variants, partly conflicting (e.g. enharmonics)
- ✚ used in many “foreign” fields (alternative ranges)
- ▬ hardly computer compatible

CWN — Common Western Notation

- ✚ well elaborated, centuries of optimization
- ✚ compact
- ✚ easy readable, “one glance”, *“prima vista”*
- ✚ utmost versatile and adaptable
  (covers Monteverdi to Stockhausen)
- ▬ specialized, restricted range (pitch information)
- ▬ restricted domain (metrical duration values)
- ▬ diversity of variants, partly conflicting (e.g. enharmonics)
- ✚ used in many “foreign” fields (alternative ranges)
- ▬ hardly computer compatible

tScore Design Principles

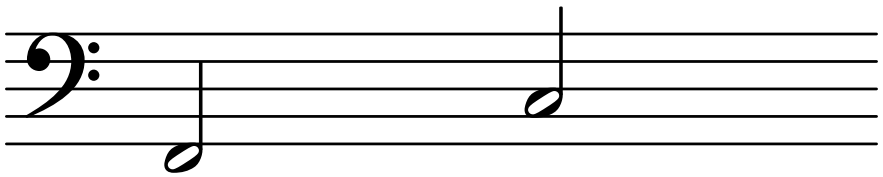
- Keep Principles of Conventional Music Notation
  - time flows from left to right
  - time distribution by division of space
  - synchronicity by super-position / vertical alignment
  - multiple tracks / parameters
- Do NOT keep restrictions
  - do NOT keep idiosyncratic junctims (e.g duration by note head AND stem AND flags AND dots AND ties AND tuplet brackets)
  - do NOT decide for parameter ranges (This is already impossible in "pure" music !-)
  - do NOT decide for one particular domain structure (there is NO standard notion of "time" !!)
  - allow arbitrary tracks with arbitrary parameter ranges. Esp. allow *overloading*
  - make it a *(typewriter) TEXT*

tScore Design Principles

- Keep Principles of Conventional Music Notation
  - time flows from left to right
  - time distribution by division of space
  - synchronicity by super-position / vertical alignment
  - multiple tracks / parameters
- Do NOT keep restrictions
  - do NOT keep idiosyncratic junctims
    (e.g duration by note head AND stem AND flags AND dots AND ties AND tuplet brackets)
  - do NOT decide for parameter ranges (This is already impossible in "pure" music !-)
  - do NOT decide for one particular domain structure
    (there is NO standard notion of "time" !!)
  - allow arbitrary tracks with arbitrary parameter ranges.
    Esp. allow *overloading*
  - make it a *(typewriter) TEXT*

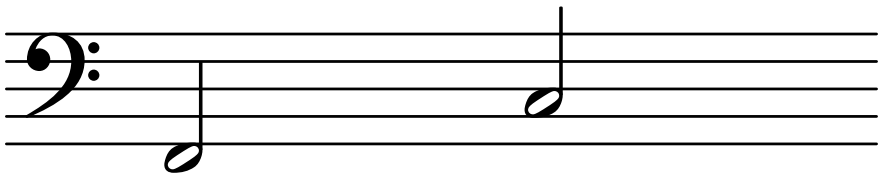
The Notion “Event” is Critical

2



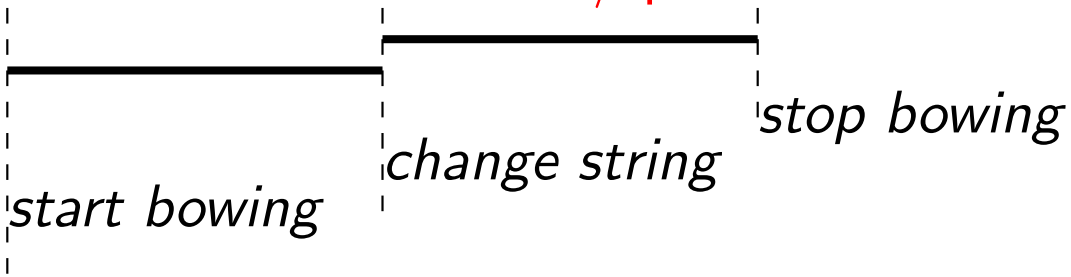
The Notion “Event” is Critical

2



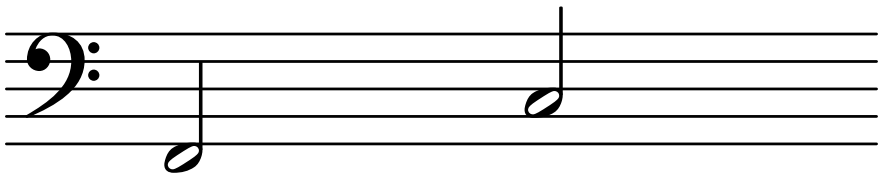
bt: “Any ’event’ must be duration-less / point-like”

3



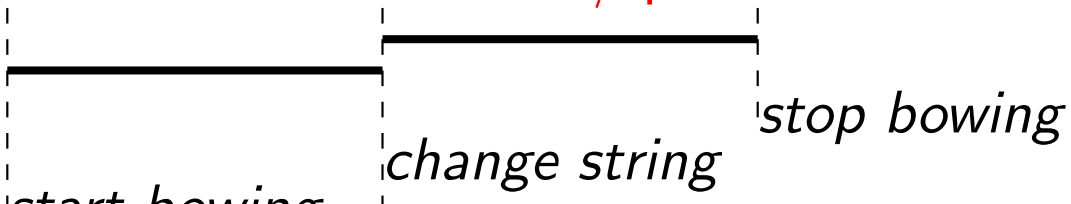
The Notion "Event" is Critical

2

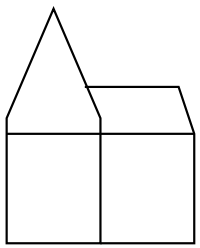


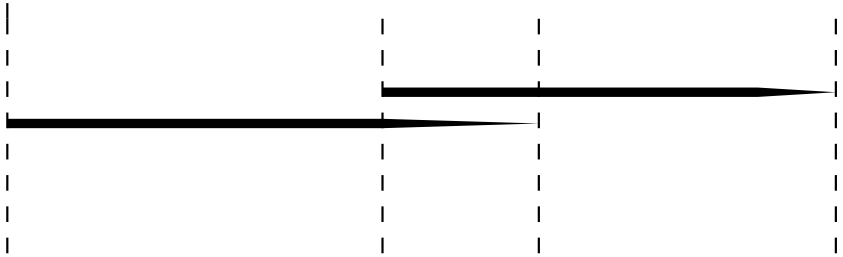
bt: "Any 'event' must be duration-less / point-like"

3



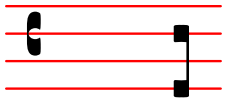
4



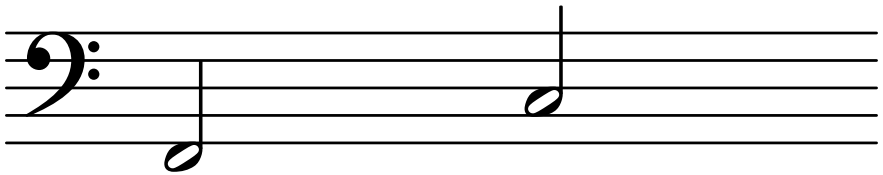


The Notion "Event" is Critical

1

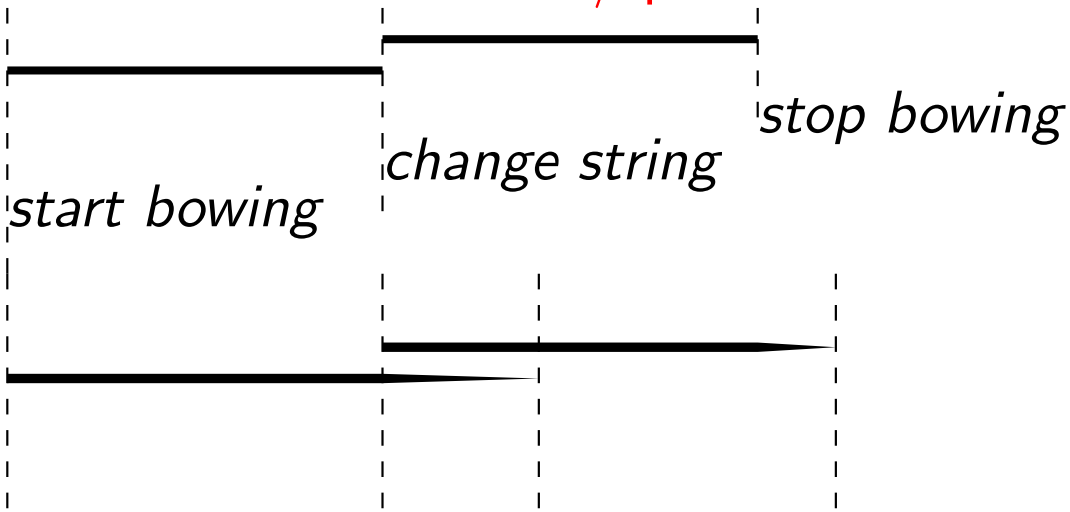


2

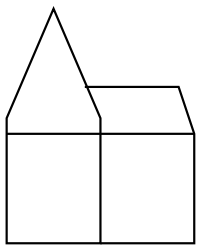


bt: "Any 'event' must be duration-less / point-like"

3



4



The Extent of the Model is Critical

- Substance vs. Accidens ?
- Part of the model OR part of the *representation* (="rendering") ?
- E.g. bar lines, accidentals, articulation signs' positions, . . .

tScore Design Principles – Consequences

- Take all decisions *as late as possible*.
- Take all decisions *explicitly*.

tScore example with standard CWN data

"PARS" separates independent time realms.

```
PARS prima
  T          19          !          !          20

  T          20                                21

PARS seconda ...
EOF
```

tScore example with standard CWN data

"T" lines define flow of time.

```
PARS prima
  T           19          !          !          20

  T           20                                 21

PARS seconda ...
EOF
```

tScore example with standard CWN data

T line entries define division of time.

```
PARS prima
  T          19          !          !          20
  T          20                                21
```

tScore example with standard CWN data

"VOX" main parameter values define **events**

```
PARS prima
  T           19            !           !          20
    VOX sop   a     b       c     (d e) f (g a h)

  T           20                                   21
    VOX sop   a           b       (e   f)
```

tScore example with standard CWN data

. . . and thus further division of time.

```
PARS prima
  T           19            !            !           20
    VOX sop   a      b      c     (d e)f (g a h)

  T           20                                    21
    VOX sop   a            b      (e   f)
```

tScore example with standard CWN data

parameter tracks for further specification of event data.

```
PARS prima
  T           19          !          !          20
    VOX sop   a     b     c    (d e)f  (g a h)
        dyn   f           ff<        >     pp
        art   -     -     >          >     >

  T           20                                21
    VOX sop   a           b     (e   f)
        nota  [           !      clef-vl
                                   ]
        art   (                    ).-
```

tScore example with standard CWN data

parameter tracks separate lexical appearance and meaning.

```
PARS prima
  T            19            !            !            20
    VOX sop    a      b      c     (d e)f (g a h)
        dyn    f             ff<          >      pp
        art    -      -      >            >      >

  T            20                                  21
    VOX sop    a             b      (e   f)
        nota   [             !      clef-vl
                                       ]
        art    (                     ).-
```

tScore example with standard CWN data

Re-usage ("overloading") of lexical entities easily possible.

```
PARS prima
  T            19          !          !          20
    VOX sop    a      b    c    (d e)f (g a h)
        dyn    f           ff<        >     pp
        art    -      -    >          >     >

  T            20                                21
    VOX sop    a           b     (e   f)
        nota   [           !      clef-vl
                                     ]
        art    (                     ).-
```

tScore example with standard CWN data

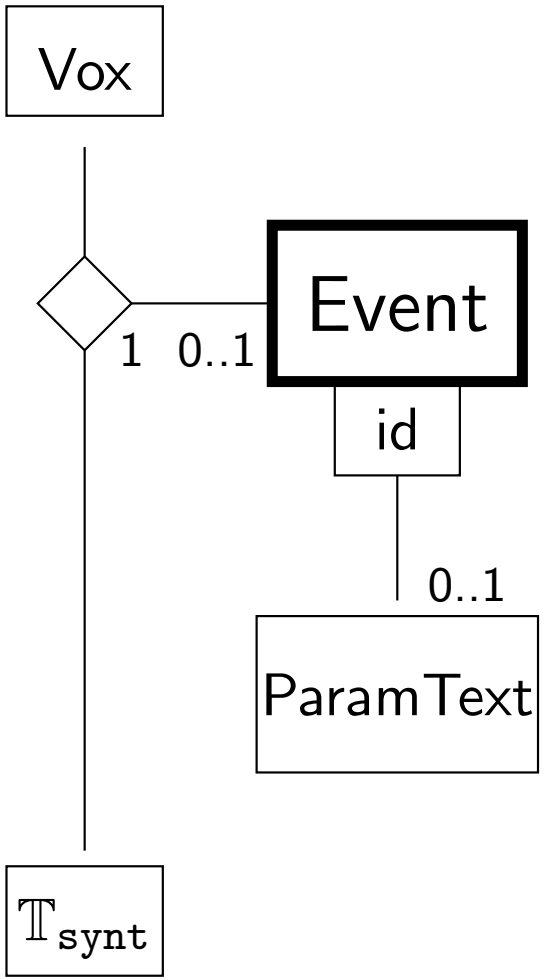
Re-usage ("overloading") of lexical entities easily possible.
(Almost) arbitrary "ASCII ART" permitted.

```
PARS prima
  T            19          !          !          20
    VOX sop    a     b     c    (d e)f (g a h)
        dyn    f           ff<        >     pp
        art    -     -     >          >     >

  T            20                                21
    VOX sop    a           b     (e   f)
        nota   [           !      clef-vl
                                    ]
        art    (                   ).-
```

tScore Data Model

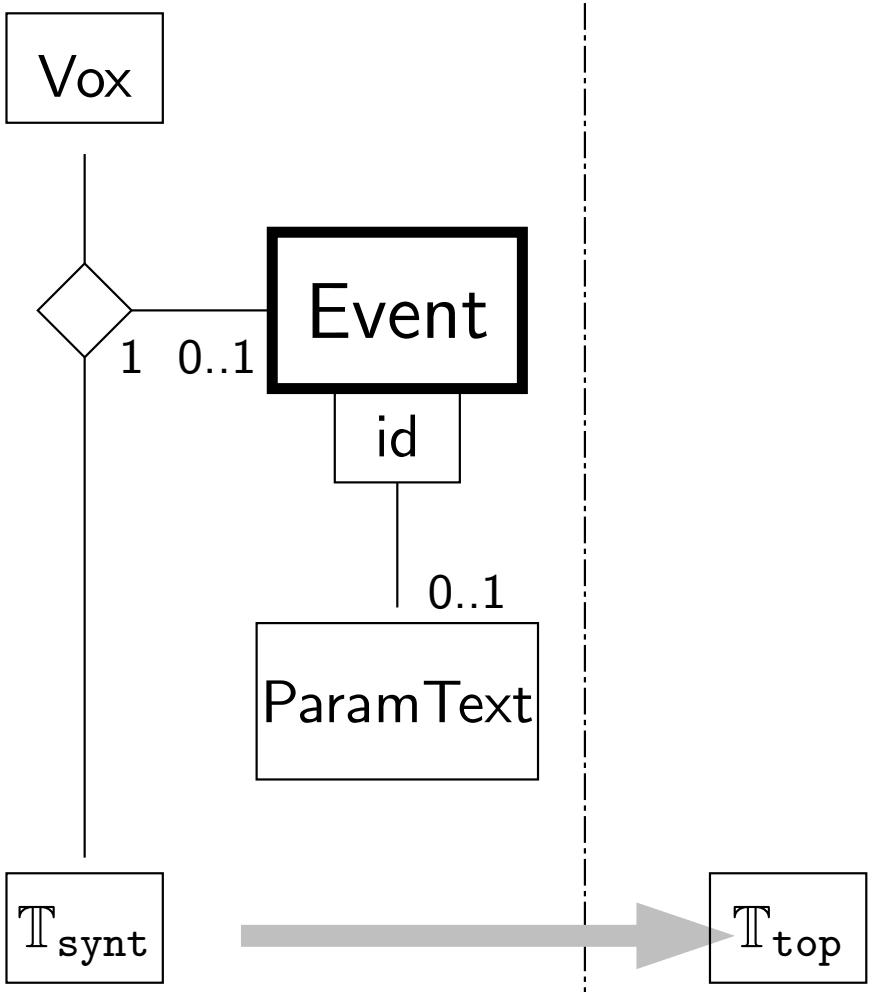
Generic Syntactic Data



$\mathrm{Vox} \times \mathbb{T} \nrightarrow \mathrm{Event}$

tScore Data Model

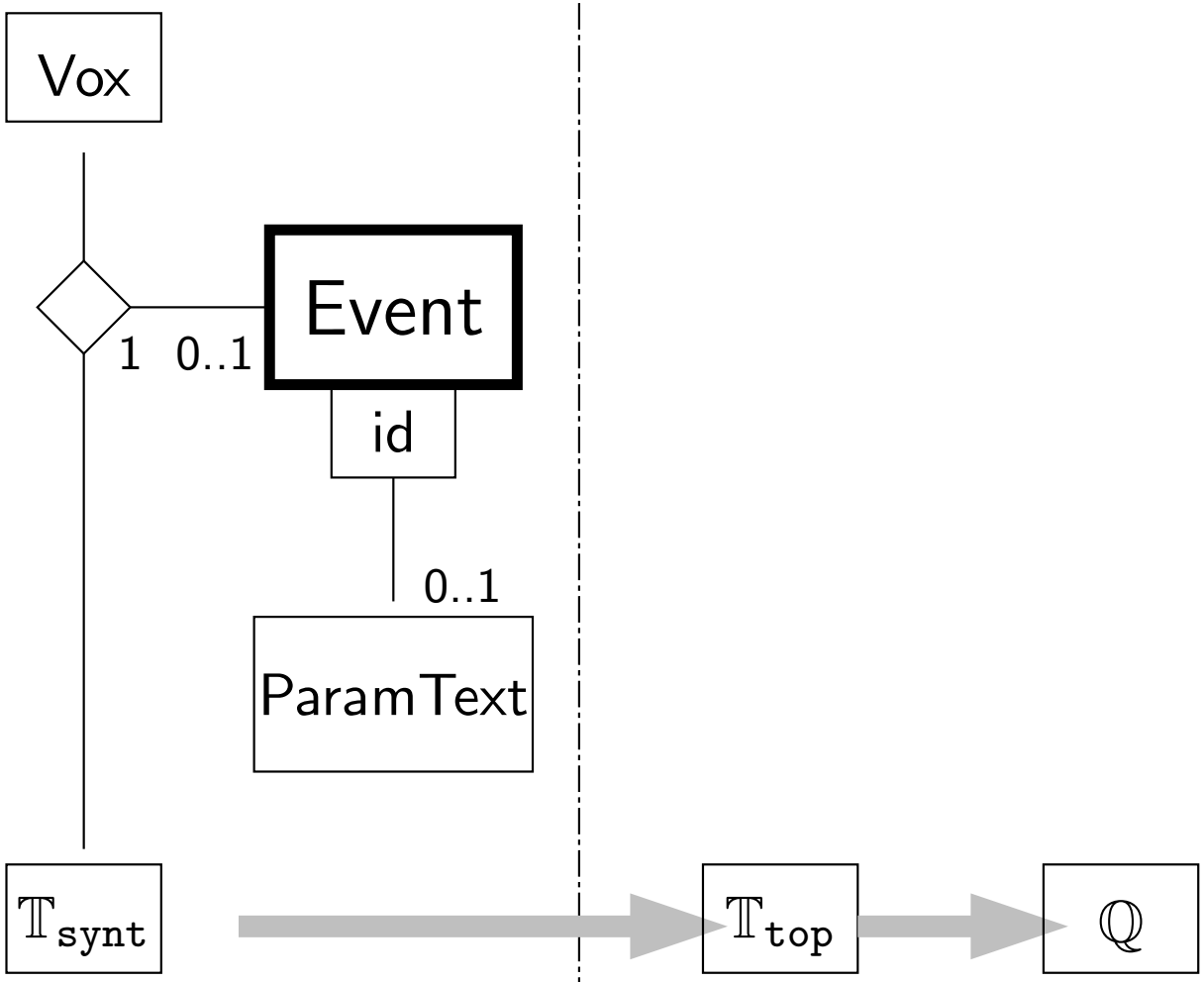
Generic Syntactic Data | *Specific Semantic Interpretation*



$\mathtt{Vox} \times \mathbb{T} \nrightarrow \mathtt{Event}$

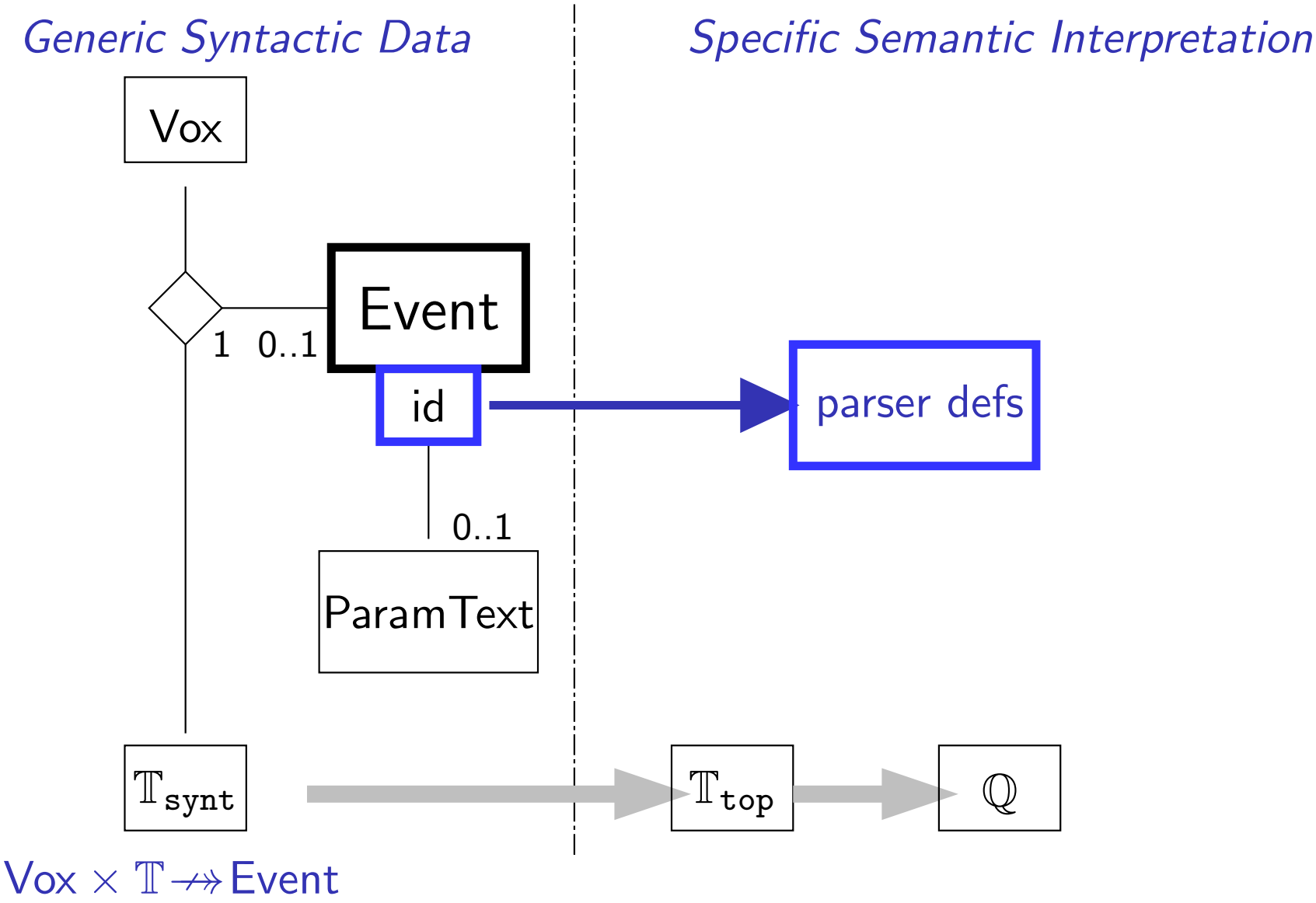
tScore Data Model

Generic Syntactic Data | *Specific Semantic Interpretation*



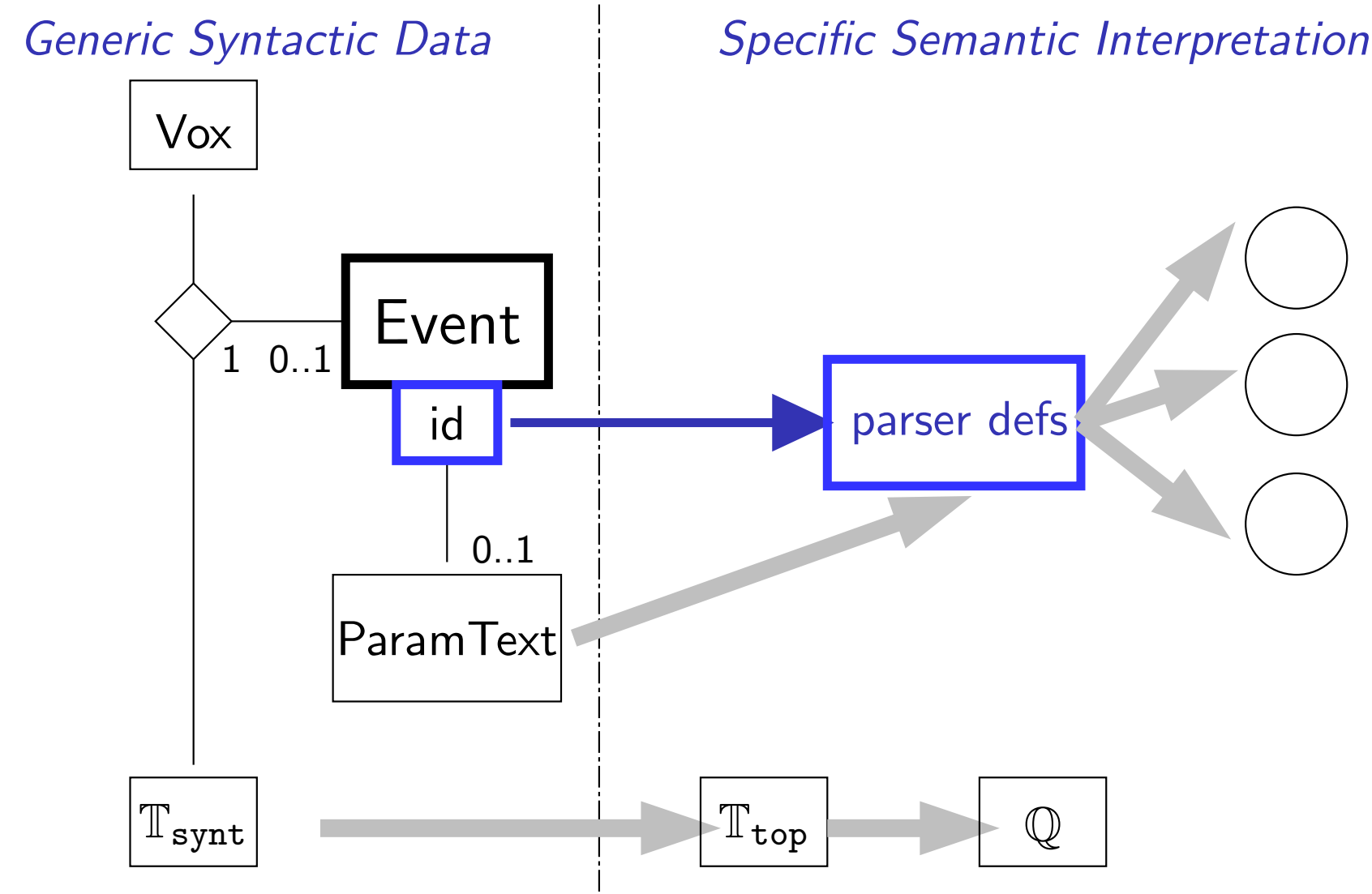
$\mathrm{Vox} \times \mathbb{T} \nrightarrow \mathrm{Event}$

tScore Data Model



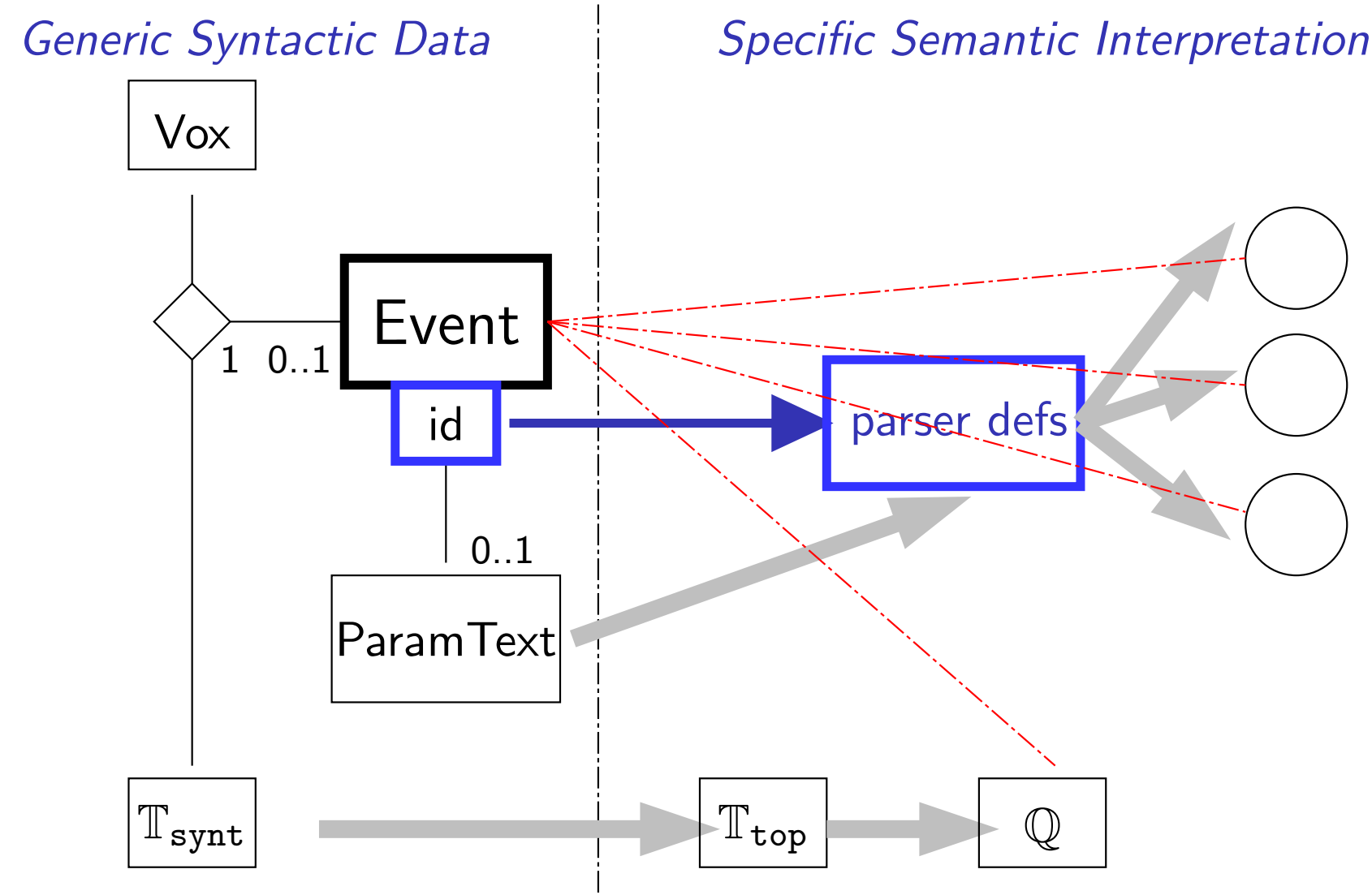
$\mathtt{Vox} \times \mathbb{T} \nrightarrow \mathtt{Event}$

tScore Data Model



$\mathtt{Vox} \times \mathbb{T} \nrightarrow \mathtt{Event}$

tScore Data Model



$\mathtt{Vox} \times \mathbb{T} \nrightarrow \mathtt{Event}$

translets Design Principles

- Users are domain experts, design their data
  (Not "parsing" experts, designing "grammars")
- They want to do programming
  by heavy use of libraries (e.g. "R", etc.)
- $\Longrightarrow$ Easy access to parsing results
- $\Longrightarrow$ Support of non-determinism
- $\Longrightarrow$ Exhaustive error diagnosis (at all stages)
- Special for tScore context:
  small-scale parsing
- "co-algebraic" storing of parsing results,
  = de-coupling of syntax combinators and storages

translets Design Principles

- Users are domain experts, design their data
  (Not "parsing" experts, designing "grammars")
- They want to do programming
  by heavy use of libraries (e.g. "R", etc.)
- $\Longrightarrow$ Easy access to parsing results
- $\Longrightarrow$ Support of non-determinism
- $\Longrightarrow$ Exhaustive error diagnosis (at all stages)
- Special for tScore context:
  small-scale parsing
- "co-algebraic" storing of parsing results,
  = de-coupling of syntax combinators and storages

translets Design Principles

- Users are domain experts, design their data
  (Not "parsing" experts, designing "grammars")
- They want to do programming
  by heavy use of libraries (e.g. "R", etc.)
- $\Longrightarrow$ Easy access to parsing results
- $\Longrightarrow$ Support of non-determinism
- $\Longrightarrow$ Exhaustive error diagnosis (at all stages)
- Special for tScore context:
  small-scale parsing
- "co-algebraic" storing of parsing results,
  = de-coupling of syntax combinators and storages

translets Design Principles

- Users are domain experts, design their data
  (Not "parsing" experts, designing "grammars")
- They want to do programming
  by heavy use of libraries (e.g. "R", etc.)
- $\Longrightarrow$ Easy access to parsing results
- $\Longrightarrow$ Support of non-determinism
- $\Longrightarrow$ Exhaustive error diagnosis (at all stages)
- Special for tScore context:
  small-scale parsing
- "co-algebraic" storing of parsing results,
  = de-coupling of syntax combinators and storages

translets Design Principles

- Users are domain experts, design their data
  (Not "parsing" experts, designing "grammars")
- They want to do programming
  by heavy use of libraries (e.g. "R", etc.)
- $\Longrightarrow$ Easy access to parsing results
- $\Longrightarrow$ Support of non-determinism
- $\Longrightarrow$ Exhaustive error diagnosis (at all stages)
- Special for tScore context:
  small-scale parsing
- "co-algebraic" storing of parsing results,
  = de-coupling of syntax combinators and storages

translets Design Principles

- Users are domain experts, design their data
  (Not "parsing" experts, designing "grammars")
- They want to do programming
  by heavy use of libraries (e.g. "R", etc.)
- $\Longrightarrow$ Easy access to parsing results
- $\Longrightarrow$ Support of non-determinism
- $\Longrightarrow$ Exhaustive error diagnosis (at all stages)
- Special for tScore context:
  small-scale parsing
- "co-algebraic" storing of parsing results,
  = de-coupling of syntax combinators and storages

translets parser combinators

$$\texttt{parse} \;:\; (K \times P \times S \times \{\mathit{ignore}, \mathit{reject}, \mathit{warn}\}) \;\rightarrow\; \mathbb{F}(\texttt{Msg})$$
$$(M, \ldots, M) \;\rightarrow\; (M, \ldots, M)$$

$$\begin{aligned} P \;::=\; & \texttt{CONST}(S) \mid \texttt{CONST}(S, v) \mid \texttt{CAT}(S, \ldots, S) \mid \texttt{REGEX}(S) \\ & \mid \texttt{SEQU}(P, \ldots, P) \mid \texttt{SEQU}(f, P, \ldots, P) \mid \texttt{SEQU}(c, P, \ldots, P) \\ & \mid \texttt{PERM}(P, \ldots, P) \mid \texttt{PERM}(f, P, \ldots, P) \mid \texttt{PERM}(c, P, \ldots, P) \\ & \mid \texttt{STORE}(M, P) \\ & \mid \texttt{ORn}(P, \ldots, P) \mid \texttt{OR1}(P, \ldots, P) \\ & \mid \texttt{PRIORn}(P, \ldots, P) \mid \texttt{PRIOR1}(P, \ldots, P) \\ & \mid \texttt{OPT}(b, v, P) \mid \texttt{STAR}(b, P, P) \mid \texttt{PLUS}(b, P, P) \\ & \mid \texttt{HEAD}(P, P, \ldots) \mid \texttt{FRAME}(P, P, P) \end{aligned}$$

translets parser combinators

$$\texttt{parse} \;:\; (K \times P \times S \times \{\mathit{ignore}, \mathit{reject}, \mathit{warn}\}) \;\rightarrow\; \mathbb{F}(\texttt{Msg})$$
$$(M, \ldots, M) \;\rightarrow\; (M, \ldots, M)$$

$$P \;::=\; \texttt{CONST}(S) \mid \texttt{CONST}(S, v) \mid \texttt{CAT}(S, \ldots, S) \mid \texttt{REGEX}(S)$$
$$\mid \texttt{SEQU}(P, \ldots, P) \mid \texttt{SEQU}(f, P, \ldots, P) \mid \texttt{SEQU}(c, P, \ldots, P)$$
$$\mid \texttt{PERM}(P, \ldots, P) \mid \texttt{PERM}(f, P, \ldots, P) \mid \texttt{PERM}(c, P, \ldots, P)$$
$$\mid \texttt{STORE}(M, P)$$
$$\mid \texttt{ORn}(P, \ldots, P) \mid \texttt{OR1}(P, \ldots, P)$$
$$\mid \texttt{PRIORn}(P, \ldots, P) \mid \texttt{PRIOR1}(P, \ldots, P)$$
$$\mid \texttt{OPT}(b, v, P) \mid \texttt{STAR}(b, P, P) \mid \texttt{PLUS}(b, P, P)$$
$$\mid \texttt{HEAD}(P, P, \ldots) \mid \texttt{FRAME}(P, P, P)$$

translets parser combinators

$$\texttt{parse} \;:\; (K \times P \times S \times \{\mathit{ignore}, \mathit{reject}, \mathit{warn}\}) \;\rightarrow\; \mathbb{F}(\texttt{Msg})$$
$$(M, \ldots, M) \;\rightarrow\; (M, \ldots, M)$$

$$P \;::=\; \texttt{CONST}(S) \mid \texttt{CONST}(S, v) \mid \texttt{CAT}(S, \ldots, S) \mid \texttt{REGEX}(S)$$
$$\mid \texttt{SEQU}(P, \ldots, P) \mid \texttt{SEQU}(f, P, \ldots, P) \mid \texttt{SEQU}(c, P, \ldots, P)$$
$$\mid \texttt{PERM}(P, \ldots, P) \mid \texttt{PERM}(f, P, \ldots, P) \mid \texttt{PERM}(c, P, \ldots, P)$$
$$\mid \texttt{STORE}(M, P)$$
$$\mid \texttt{ORn}(P, \ldots, P) \mid \texttt{OR1}(P, \ldots, P)$$
$$\mid \texttt{PRIORn}(P, \ldots, P) \mid \texttt{PRIOR1}(P, \ldots, P)$$
$$\mid \texttt{OPT}(b, v, P) \mid \texttt{STAR}(b, P, P) \mid \texttt{PLUS}(b, P, P)$$
$$\mid \texttt{HEAD}(P, P, \ldots) \mid \texttt{FRAME}(P, P, P)$$

translets parser combinators

$$\texttt{parse} \;:\; (K \times P \times S \times \{\mathit{ignore}, \mathit{reject}, \mathit{warn}\}) \;\rightarrow\; \mathbb{F}(\texttt{Msg})$$
$$(M, \ldots, M) \;\rightarrow\; (M, \ldots, M)$$

$$P \;::=\; \texttt{CONST}(S) \mid \texttt{CONST}(S, v) \mid \texttt{CAT}(S, \ldots, S) \mid \texttt{REGEX}(S)$$
$$\mid \texttt{SEQU}(P, \ldots, P) \mid \texttt{SEQU}(f, P, \ldots, P) \mid \texttt{SEQU}(c, P, \ldots, P)$$
$$\mid \texttt{PERM}(P, \ldots, P) \mid \texttt{PERM}(f, P, \ldots, P) \mid \texttt{PERM}(c, P, \ldots, P)$$
$$\mid \texttt{STORE}(M, P)$$
$$\mid \texttt{ORn}(P, \ldots, P) \mid \texttt{OR1}(P, \ldots, P)$$
$$\mid \texttt{PRIORn}(P, \ldots, P) \mid \texttt{PRIOR1}(P, \ldots, P)$$
$$\mid \texttt{OPT}(b, v, P) \mid \texttt{STAR}(b, P, P) \mid \texttt{PLUS}(b, P, P)$$
$$\mid \texttt{HEAD}(P, P, \ldots) \mid \texttt{FRAME}(P, P, P)$$

translets parser combinators

$$\texttt{parse} \;:\; (K \times P \times S \times \{\mathit{ignore}, \mathit{reject}, \mathit{warn}\}) \;\rightarrow\; \mathbb{F}(\texttt{Msg})$$
$$(M, \ldots, M) \;\rightarrow\; (M, \ldots, M)$$

$$P \;::=\; \texttt{CONST}(S) \mid \texttt{CONST}(S, v) \mid \texttt{CAT}(S, \ldots, S) \mid \texttt{REGEX}(S)$$
$$\mid \texttt{SEQU}(P, \ldots, P) \mid \texttt{SEQU}(f, P, \ldots, P) \mid \texttt{SEQU}(c, P, \ldots, P)$$
$$\mid \texttt{PERM}(P, \ldots, P) \mid \texttt{PERM}(f, P, \ldots, P) \mid \texttt{PERM}(c, P, \ldots, P)$$
$$\mid \texttt{STORE}(M, P)$$
$$\mid \texttt{ORn}(P, \ldots, P) \mid \texttt{OR1}(P, \ldots, P)$$
$$\mid \texttt{PRIORn}(P, \ldots, P) \mid \texttt{PRIOR1}(P, \ldots, P)$$
$$\mid \texttt{OPT}(b, v, P) \mid \texttt{STAR}(b, P, P) \mid \texttt{PLUS}(b, P, P)$$
$$\mid \texttt{HEAD}(P, P, \ldots) \mid \texttt{FRAME}(P, P, P)$$

translets parser combinators

$$\texttt{parse} \;:\; (K \times P \times S \times \{\mathit{ignore}, \mathit{reject}, \mathit{warn}\}) \;\rightarrow\; \mathbb{F}(\texttt{Msg})$$
$$(M, \ldots, M) \;\rightarrow\; (M, \ldots, M)$$

$$P \;::=\; \texttt{CONST}(S) \mid \texttt{CONST}(S, v) \mid \texttt{CAT}(S, \ldots, S) \mid \texttt{REGEX}(S)$$
$$\mid \texttt{SEQU}(P, \ldots, P) \mid \texttt{SEQU}(f, P, \ldots, P) \mid \texttt{SEQU}(c, P, \ldots, P)$$
$$\mid \texttt{PERM}(P, \ldots, P) \mid \texttt{PERM}(f, P, \ldots, P) \mid \texttt{PERM}(c, P, \ldots, P)$$
$$\mid \texttt{STORE}(M, P)$$
$$\mid \texttt{ORn}(P, \ldots, P) \mid \texttt{OR1}(P, \ldots, P)$$
$$\mid \texttt{PRIORn}(P, \ldots, P) \mid \texttt{PRIOR1}(P, \ldots, P)$$
$$\mid \texttt{OPT}(b, v, P) \mid \texttt{STAR}(b, P, P) \mid \texttt{PLUS}(b, P, P)$$
$$\mid \texttt{HEAD}(P, P, \ldots) \mid \texttt{FRAME}(P, P, P)$$

translets parser combinators

$$\texttt{parse} \;:\; (K \times P \times S \times \{\mathit{ignore}, \mathit{reject}, \mathit{warn}\}) \;\rightarrow\; \mathbb{F}(\texttt{Msg})$$
$$(M, \ldots, M) \;\rightarrow\; (M, \ldots, M)$$

$$P \;::=\; \texttt{CONST}(S) \mid \texttt{CONST}(S, v) \mid \texttt{CAT}(S, \ldots, S) \mid \texttt{REGEX}(S)$$
$$\mid \texttt{SEQU}(P, \ldots, P) \mid \texttt{SEQU}(f, P, \ldots, P) \mid \texttt{SEQU}(c, P, \ldots, P)$$
$$\mid \texttt{PERM}(P, \ldots, P) \mid \texttt{PERM}(f, P, \ldots, P) \mid \texttt{PERM}(c, P, \ldots, P)$$
$$\mid \texttt{STORE}(M, P)$$
$$\mid \texttt{ORn}(P, \ldots, P) \mid \texttt{OR1}(P, \ldots, P)$$
$$\mid \texttt{PRIORn}(P, \ldots, P) \mid \texttt{PRIOR1}(P, \ldots, P)$$
$$\mid \texttt{OPT}(b, v, P) \mid \texttt{STAR}(b, P, P) \mid \texttt{PLUS}(b, P, P)$$
$$\mid \texttt{HEAD}(P, P, \ldots) \mid \texttt{FRAME}(P, P, P)$$

translets parser combinators

$$\texttt{parse} \;:\; (K \times P \times S \times \{\mathit{ignore}, \mathit{reject}, \mathit{warn}\}) \;\rightarrow\; \mathbb{F}(\texttt{Msg})$$
$$(M, \ldots, M) \;\rightarrow\; (M, \ldots, M)$$

$$\begin{aligned} P \;::=\; & \texttt{CONST}(S) \mid \texttt{CONST}(S, v) \mid \texttt{CAT}(S, \ldots, S) \mid \texttt{REGEX}(S) \\ & \mid \texttt{SEQU}(P, \ldots, P) \mid \texttt{SEQU}(f, P, \ldots, P) \mid \texttt{SEQU}(c, P, \ldots, P) \\ & \mid \texttt{PERM}(P, \ldots, P) \mid \texttt{PERM}(f, P, \ldots, P) \mid \texttt{PERM}(c, P, \ldots, P) \\ & \mid \texttt{STORE}(M, P) \\ & \mid \texttt{ORn}(P, \ldots, P) \mid \texttt{OR1}(P, \ldots, P) \\ & \quad \mid \texttt{PRIORn}(P, \ldots, P) \mid \texttt{PRIOR1}(P, \ldots, P) \\ & \mid \texttt{OPT}(b, v, P) \mid \texttt{STAR}(b, P, P) \mid \texttt{PLUS}(b, P, P) \\ & \mid \texttt{HEAD}(P, P, \ldots) \mid \texttt{FRAME}(P, P, P) \end{aligned}$$

translets typing rules

$$\frac{[\![\ \_\ ]\!]^T : P \to \text{Java-types}}{\begin{array}{c}[\![\,\texttt{CONST}(S)\,]\!]^T = [\![\,\texttt{REGEX}(S)\,]\!]^T = \texttt{String} \\ [\![\,\texttt{CAT}(S \to t)\,]\!]^T = [\![\,\texttt{CONST}(S, v : t)\,]\!]^T = t\end{array}}$$

$$\frac{[\![\,p_k\,]\!]^T = t_k \qquad f, c : (t_1, \ldots, t_n) \to t_R}{\begin{array}{c}[\![\,\texttt{SEQU}(p_1, \ldots, p_n)\,]\!]^T = [\![\,\texttt{PERM}(p_1, \ldots, p_n)\,]\!]^T = \texttt{Object} \\ [\![\,\texttt{SEQU}(f, p_1, \ldots, p_n)\,]\!]^T = [\![\,\texttt{PERM}(f, p_1, \ldots, p_n)\,]\!]^T = t_R \\ [\![\,\texttt{SEQU}(c, p_1, \ldots, p_n)\,]\!]^T = [\![\,\texttt{PERM}(c, p_1, \ldots, p_n)\,]\!]^T = t_R\end{array}}$$

$$\frac{M : K \to t_1 \ \cup \ K \leftrightarrow t_1}{[\![\,\texttt{STORE}(M, P_1)\,]\!]^T = t_1}$$

translets typing rules

$$\frac{[\![\,\_\,]\!]^T : P \to \text{Java-types}}{\begin{array}{c}[\![\,\texttt{CONST}(S)\,]\!]^T = [\![\,\texttt{REGEX}(S)\,]\!]^T = \texttt{String} \\ [\![\,\texttt{CAT}(S \to t)\,]\!]^T = [\![\,\texttt{CONST}(S, v : t)\,]\!]^T = t\end{array}}$$

$$\frac{[\![\,p_k\,]\!]^T = t_k \qquad f, c : (t_1, \ldots, t_n) \to t_R}{\begin{array}{c}[\![\,\texttt{SEQU}(p_1, \ldots, p_n)\,]\!]^T = [\![\,\texttt{PERM}(p_1, \ldots, p_n)\,]\!]^T = \texttt{Object} \\ [\![\,\texttt{SEQU}(f, p_1, \ldots, p_n)\,]\!]^T = [\![\,\texttt{PERM}(f, p_1, \ldots, p_n)\,]\!]^T = t_R \\ [\![\,\texttt{SEQU}(c, p_1, \ldots, p_n)\,]\!]^T = [\![\,\texttt{PERM}(c, p_1, \ldots, p_n)\,]\!]^T = t_R\end{array}}$$

$$\frac{M : K \to t_1 \;\cup\; K \leftrightarrow t_1}{[\![\,\texttt{STORE}(M, P_1)\,]\!]^T = t_1}$$

translets typing rules

$$\frac{[\![\ \_\ ]\!]^T : P \to \text{Java-types}}{\begin{array}{c}[\![\ \texttt{CONST}(S)\ ]\!]^T = [\![\ \texttt{REGEX}(S)\ ]\!]^T = \texttt{String} \\ [\![\ \texttt{CAT}(S \to t)\ ]\!]^T = [\![\ \texttt{CONST}(S, v : t)\ ]\!]^T = t\end{array}}$$

$$\frac{[\![\ p_k\ ]\!]^T = t_k \qquad f, c : (t_1, \ldots, t_n) \to t_R}{\begin{array}{c}[\![\ \texttt{SEQU}(p_1, \ldots, p_n)\ ]\!]^T = [\![\ \texttt{PERM}(p_1, \ldots, p_n)\ ]\!]^T = \texttt{Object} \\ [\![\ \texttt{SEQU}(f, p_1, \ldots, p_n)\ ]\!]^T = [\![\ \texttt{PERM}(f, p_1, \ldots, p_n)\ ]\!]^T = t_R \\ [\![\ \texttt{SEQU}(c, p_1, \ldots, p_n)\ ]\!]^T = [\![\ \texttt{PERM}(c, p_1, \ldots, p_n)\ ]\!]^T = t_R\end{array}}$$

$$\frac{M : K \to t_1 \ \cup \ K \leftrightarrow t_1}{[\![\ \texttt{STORE}(M, P_1)\ ]\!]^T = t_1}$$

translets typing rules — II

$$[\![\_]\!]^T : P \rightarrow \text{Java-types}$$
$$[\![p_k]\!]^T = t_k \qquad b : \texttt{boolean}$$

$$[\![\texttt{ORn}(p_1, \ldots, p_n)]\!]^T = [\![\texttt{PRIORn}(p_1, \ldots, p_n)]\!]^T = \texttt{CoTuple}\langle t_1, \ldots, t_n \rangle$$
$$[\![\texttt{OR1}(p_1, \ldots, p_n)]\!]^T = [\![\texttt{PRIOR1}(p_1, \ldots, p_n)]\!]^T = \mathit{upperLimit}(t_1, \ldots, t_n)$$

$$[\![\texttt{OPT}(b, v : t_1, p_1)]\!]^T = t_1 \quad \textit{// Java type does include "null" !}$$
$$[\![\texttt{STAR}(b, p_1, p_2)]\!]^T = [\![\texttt{PLUS}(b, p_1, p_2)]\!]^T = \texttt{List}\langle t_1 \rangle$$

$$[\![\texttt{HEAD}(p_1, p_2, \ldots)]\!]^T = t_1 \qquad [\![\texttt{FRAME}(p_1, p_2, p_3)]\!]^T = t_2$$

translets typing rules — II

$$[\![\_]\!]^T : P \rightarrow \text{Java-types}$$
$$[\![p_k]\!]^T = t_k \qquad b : \texttt{boolean}$$

$$[\![\texttt{ORn}(p_1, \ldots, p_n)]\!]^T = [\![\texttt{PRIORn}(p_1, \ldots, p_n)]\!]^T = \texttt{CoTuple}\langle t_1, \ldots, t_n \rangle$$
$$[\![\texttt{OR1}(p_1, \ldots, p_n)]\!]^T = [\![\texttt{PRIOR1}(p_1, \ldots, p_n)]\!]^T = \mathit{upperLimit}(t_1, \ldots, t_n)$$

$$[\![\texttt{OPT}(b, v : t_1, p_1)]\!]^T = t_1 \quad \textit{// Java type does include "null" !}$$
$$[\![\texttt{STAR}(b, p_1, p_2)]\!]^T = [\![\texttt{PLUS}(b, p_1, p_2)]\!]^T = \texttt{List}\langle t_1 \rangle$$

$$[\![\texttt{HEAD}(p_1, p_2, \ldots)]\!]^T = t_1 \qquad [\![\texttt{FRAME}(p_1, p_2, p_3)]\!]^T = t_2$$

translets typing rules — II

$$[\![\_]\!]^T : P \rightarrow \text{Java-types}$$
$$[\![p_k]\!]^T = t_k \qquad b : \texttt{boolean}$$

$$[\![\texttt{ORn}(p_1, \ldots, p_n)]\!]^T = [\![\texttt{PRIORn}(p_1, \ldots, p_n)]\!]^T = \texttt{CoTuple}\langle t_1, \ldots, t_n \rangle$$
$$[\![\texttt{OR1}(p_1, \ldots, p_n)]\!]^T = [\![\texttt{PRIOR1}(p_1, \ldots, p_n)]\!]^T = \mathit{upperLimit}(t_1, \ldots, t_n)$$

$$[\![\texttt{OPT}(b, v : t_1, p_1)]\!]^T = t_1 \quad \textit{// Java type does include "null" !}$$
$$[\![\texttt{STAR}(b, p_1, p_2)]\!]^T = [\![\texttt{PLUS}(b, p_1, p_2)]\!]^T = \texttt{List}\langle t_1 \rangle$$

$$[\![\texttt{HEAD}(p_1, p_2, \ldots)]\!]^T = t_1 \qquad [\![\texttt{FRAME}(p_1, p_2, p_3)]\!]^T = t_2$$

Reaction on ambigue parsing result / non-determinism:

- silently ignore / select solution
- select solution and emit warning
- reject

Error messaging:

- iff there are completely covered regExp:
  (=error due to ambiguity or to superfluous rest)
  - display only these complete covers as an input/regExp relation
  - display superfluous rest input
- otherwise: display all partially covered regExp:
  - display matched prefix, as an input/regExp relation
  - display the legal continuations / expectations
  - display superfluous rest input

Error messaging:

Collect mappings between input and parser expressions as parsing threads.

Collect expected continuations after thread's death.

```
kdf_vii.tscore:6:34 No parsing result for "C*2uX"
The grammar rule for this input is parser "all" =
OR1(SEQU(CAT("I", "II", "III", "IV")?, CAT("D", "C")?,
         PERM((">")?, ("u")?, ("r")?,
              OR1(SEQU( "/", REGEX("[2-9]")),
                  SEQU(  "*", REGEX("[2-9]")))
              )
         ),
    SEQU( REGEX("[1-9]"), PERM(( ">")?, ("u")?, ("r")?)),
    "X",
     "%"
   )
After having parsed ...
                                        "C"
                              "*"   "2"
                         "u"
... there could follow input according to ...
  PERM( ">"?, "r"?)
There are unparsable rest characters
  "X"  at position 5
```

tScore Prototypical Applications – I

- **FormplanFuge**
- **MovingForms**
- **MaWiRic** "Ricercar" — Visual interpretation of a poem of T.S.Eliot by Mathias Wittekopf.
- **MaWiCM** "Clock Music" — Visual interpretation of a poem of Hans Arp, dto.
- **MIWorat** Representation and automated processing of manually extracted intermediate musical analysis results.
- **Graphart – Scharen**

tScore Prototypical Applications – II

- **CWN** Common Western Notation
  Transformations to lilypond / MusicXML / "sig" synthesis / etc.
  E.g. `fuga_a_3.cwn`.
- **CWN** Automated Statistic Analysis
  E.g. J.S.Bach "Kunst der Fuge"

More info and contact at . . .

http://www.bandm.eu/music
post@markuslepper.eu
baltasar@trancon.de