W3C Music Notation CG

Working Processes Update

Outline of Process

- Issues raised in MNX GitHub repository by any community member
- Open discussion from community welcomed on any issue
- Co-chairs review issues regularly
- Co-chairs identify issues for Active Review
- Issues in Active Review are intended to be the focus of community discussion
- Once consensus is reached, issue is either closed with no action, or a pull request is created to address the issue
- Issues with pending pull requests have the label PR Review
- Once pull request has been reviewed, pull request is merged and issue is closed

Milestones

- Issues are added to a milestone after review by the co-chairs
- Currently three milestones defined:
  - V1 – targeted for implementation in the first version of the MNX specification
  - V next – targeted for implementation in the following version of the MNX specification
  - Uncommitted – reviewed by the co-chairs, but not currently targeted for any specific release
- Only co-chairs can determine milestones

How to get involved

- Join the Music Notation Community Group on the W3C web site
- Agree to the W3C Contributor License Agreement
- Register for an account on GitHub
- Ideally, add your GitHub username to the Contributors page on the MN CG wiki
- We recommend you “Watch” the MNX repository on GitHub to be notified by email of discussion on issues (set up a rule to filter email to go into a specific folder)
- Raise new issues or add your comments to existing ones

Current status

- 40 open issues
- 2 issues open in V1 milestone
- 1 issue open in V next milestone
- 1 issue open in Uncommitted
- 36 issues yet to be assigned a milestone

MNX-Generic

(or, for Prince fans, *The Encoding Formerly Known As GMNX*)

An encoding standard for music notation instances

Joe Berkovitz
co-chair, W3C Music Notation Group

What's MNX-Generic?

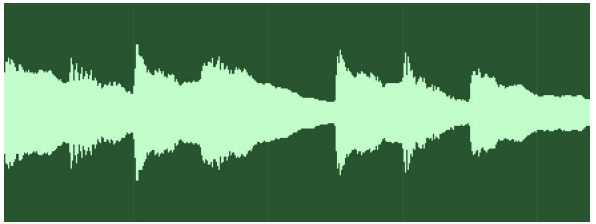
- A low-level, literal encoding format for instances of scores
- 3 kinds of instances:
  - Graphics
  - Performance audio
  - Performance data ("MIDI-like")
- Connects time (audio) to space (graphics) or semantics
- No dynamic layout or interpretation: instances are static
- Not limited to specific definition of "music notation" for a culture or genre
- Does not encode the "meaning" of notational symbols
  - But… can refer to contents of a semantic document!

Why use MNX-Generic?

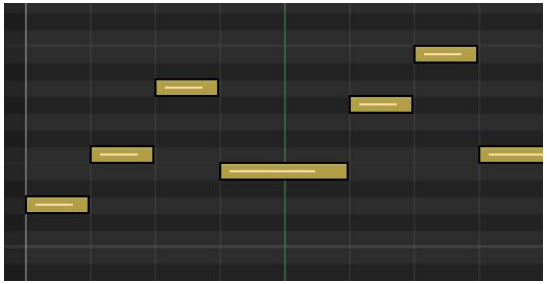
- Presentation of any notated music, as long as it's static
- Use cases that don't involve score modification or reflowing:
  - Score viewers and players with fixed layout
  - Music learning/practice
  - Play-along applications
  - Performance assessment
  - Music with highly customized or unique graphics (analysis, appreciation)
  - Analysis and exploration
  - Archival copies of rendered score

Instance Types

Graphics

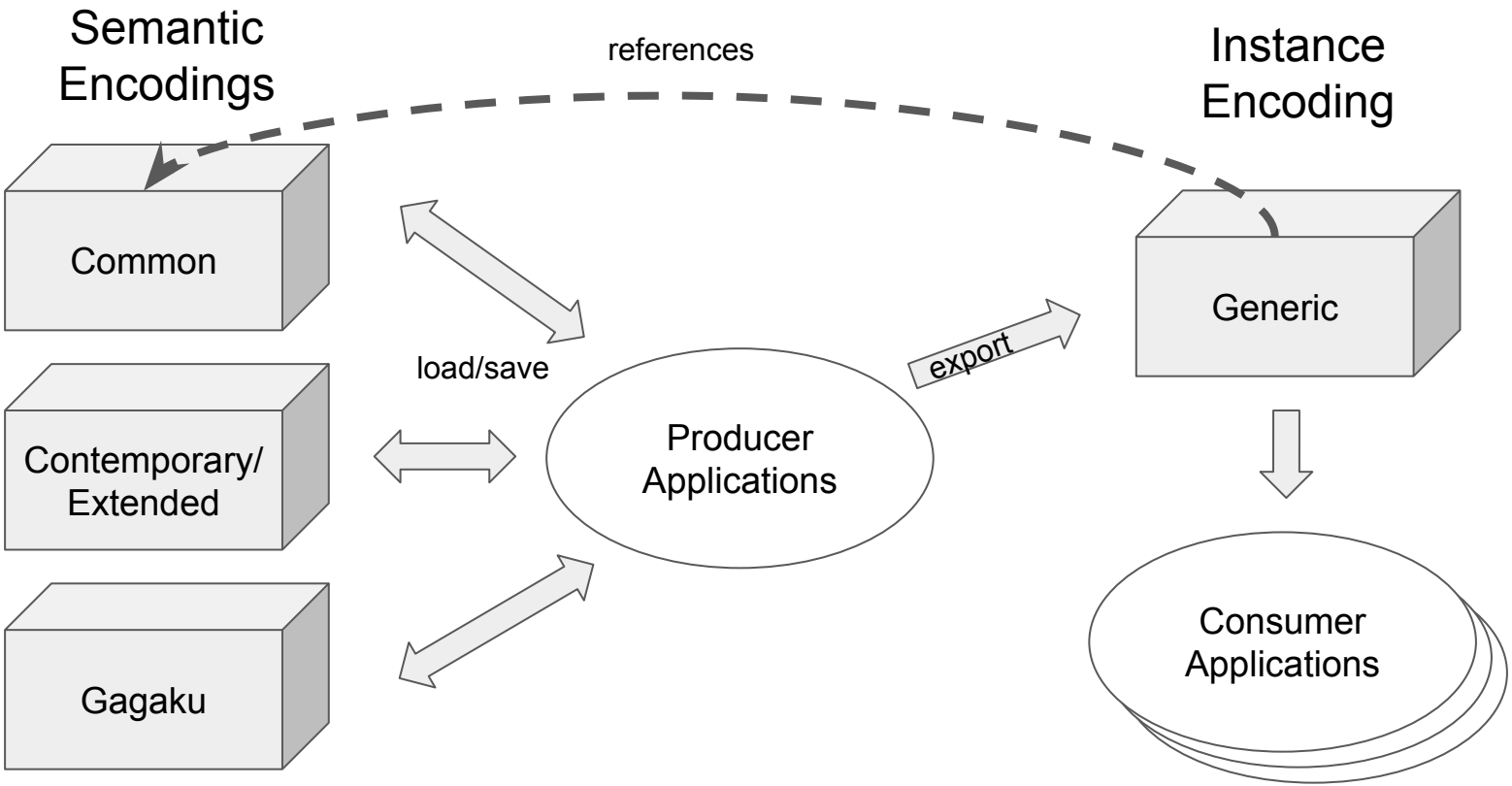


Audio Media

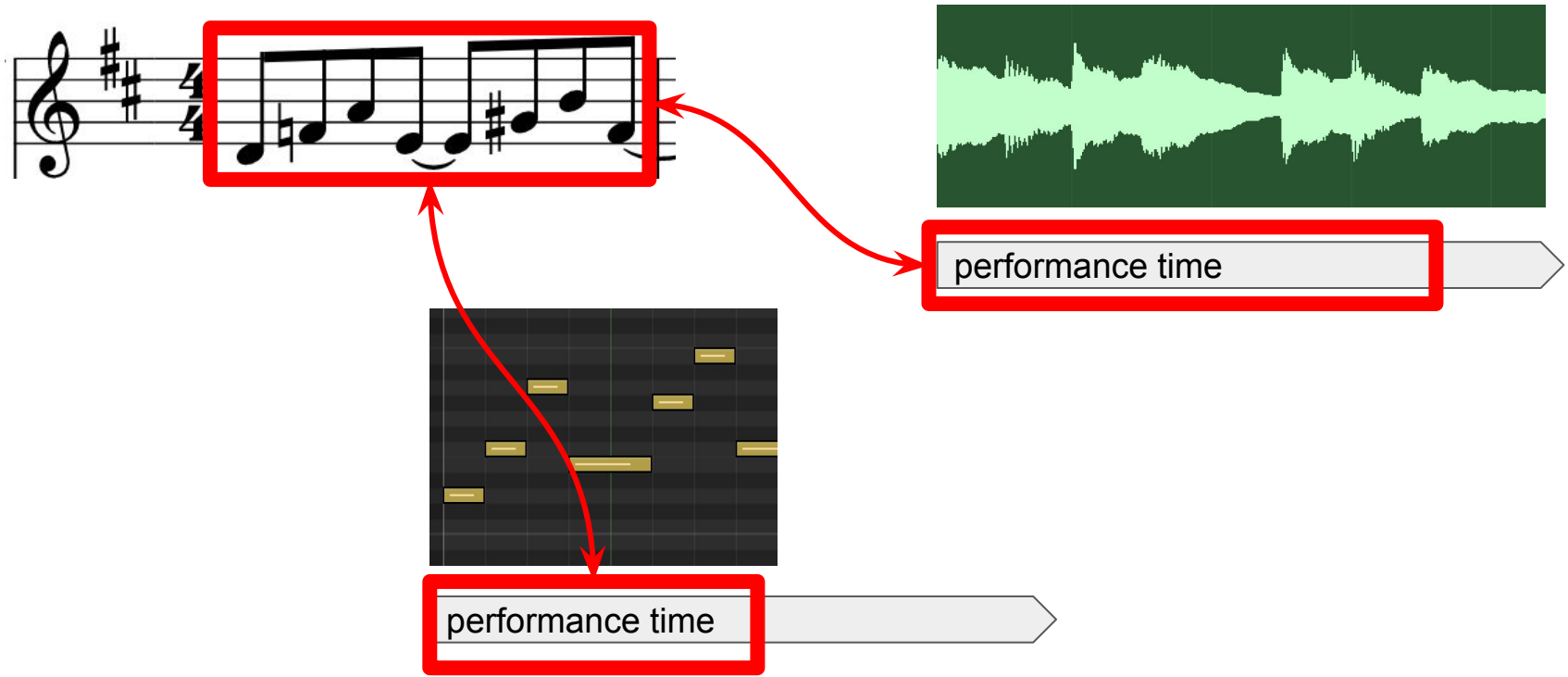


Performance Data

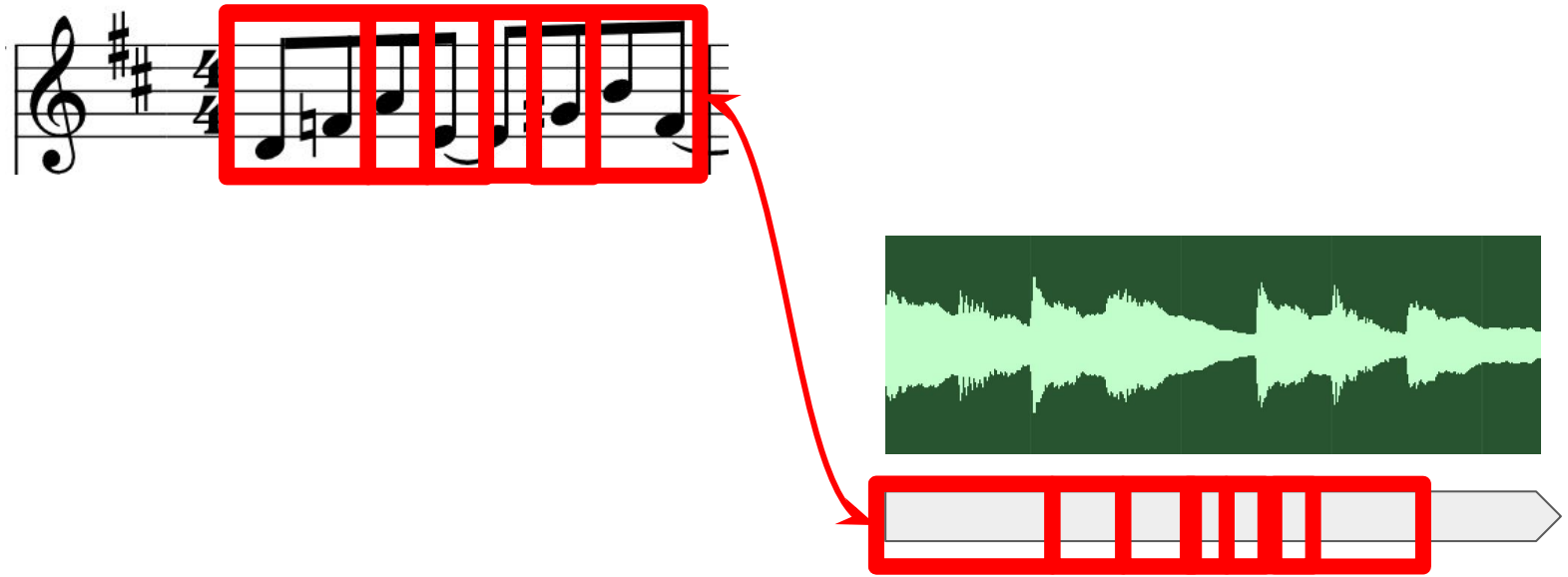
An MNX Encoding Ecosystem



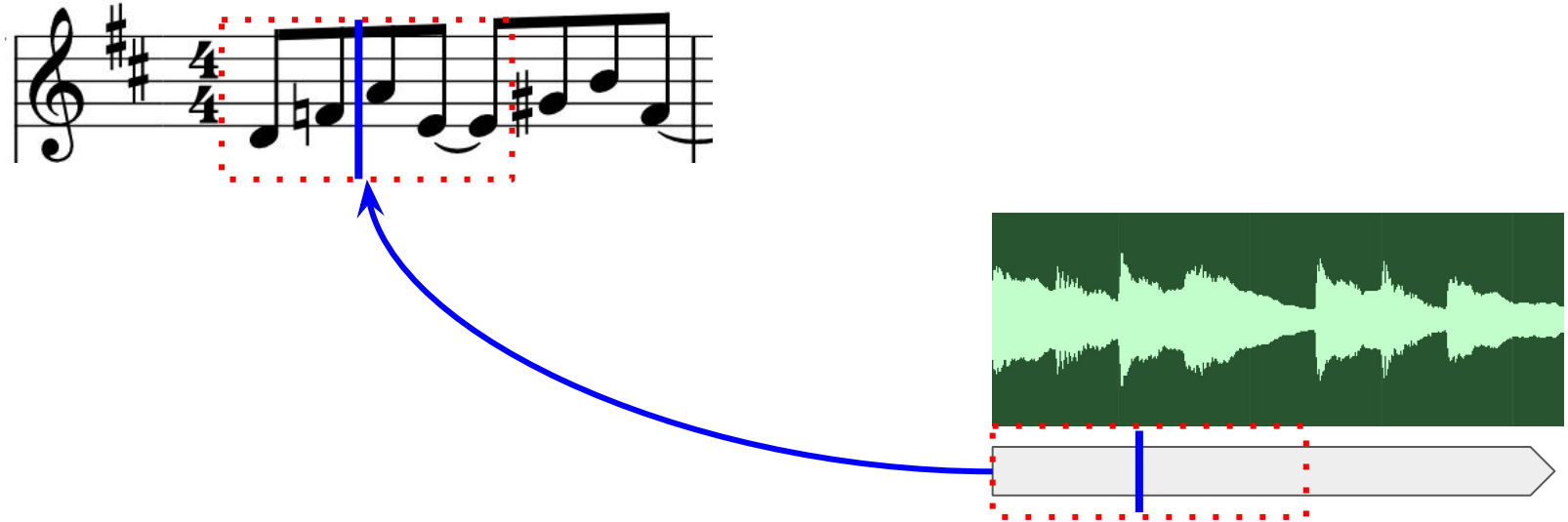
Discrete space/time mappings: *Regions*



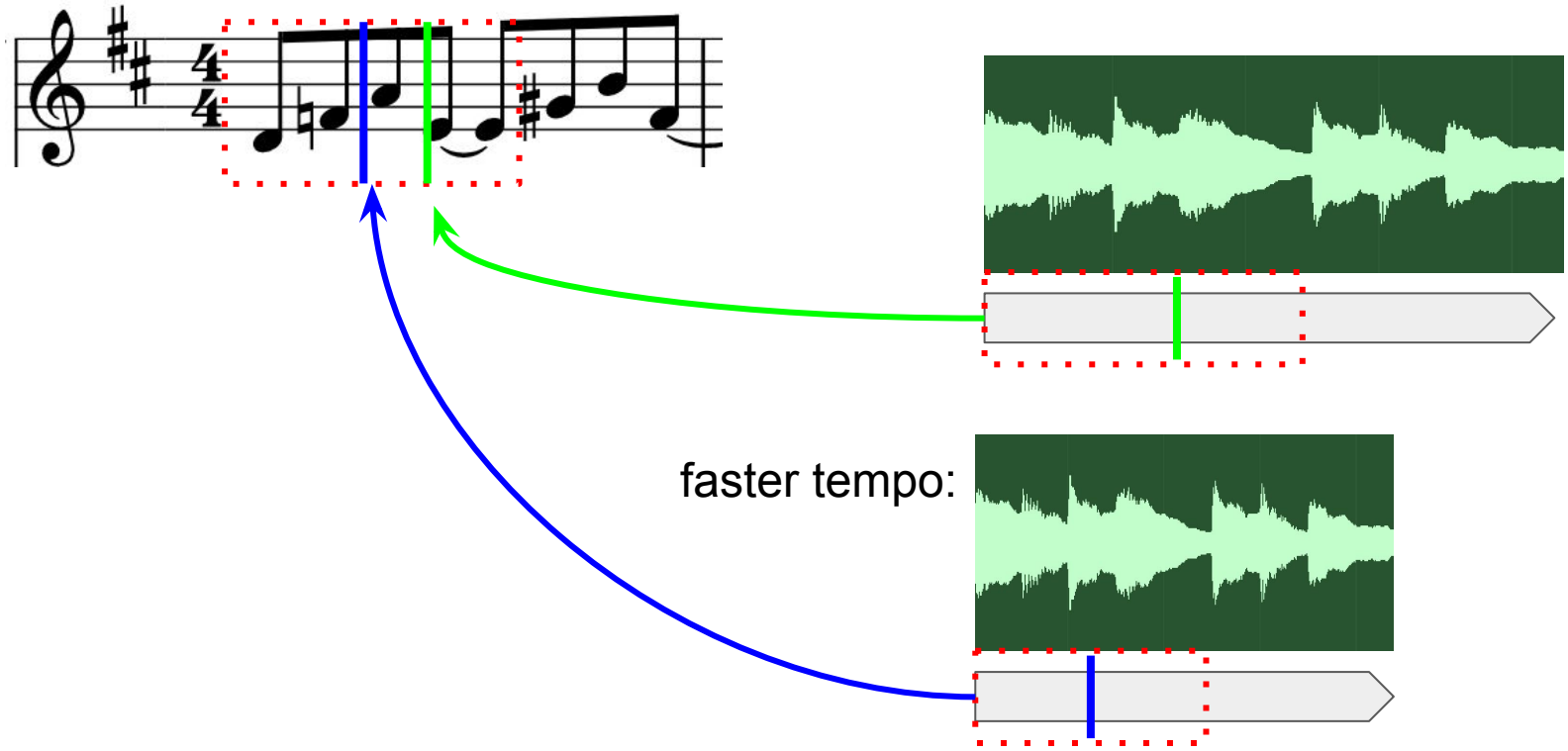
Regions can be arbitrarily fine-grained



Continuous space/time mappings: *Cursors*

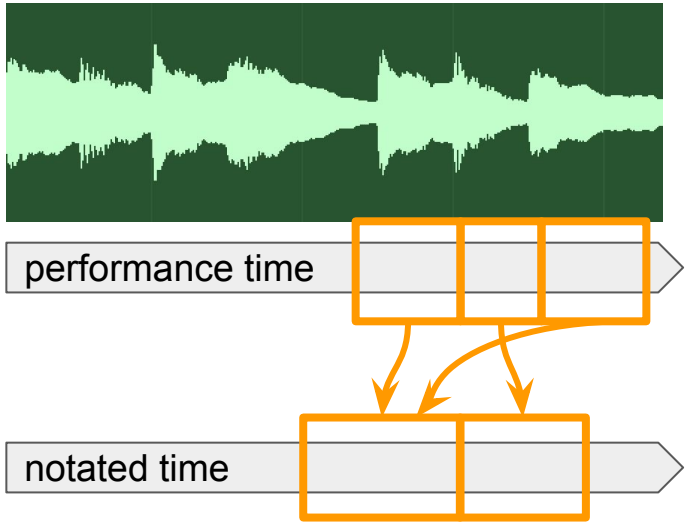


Multiple performances with different mappings...



Notated time: An abstract time axis

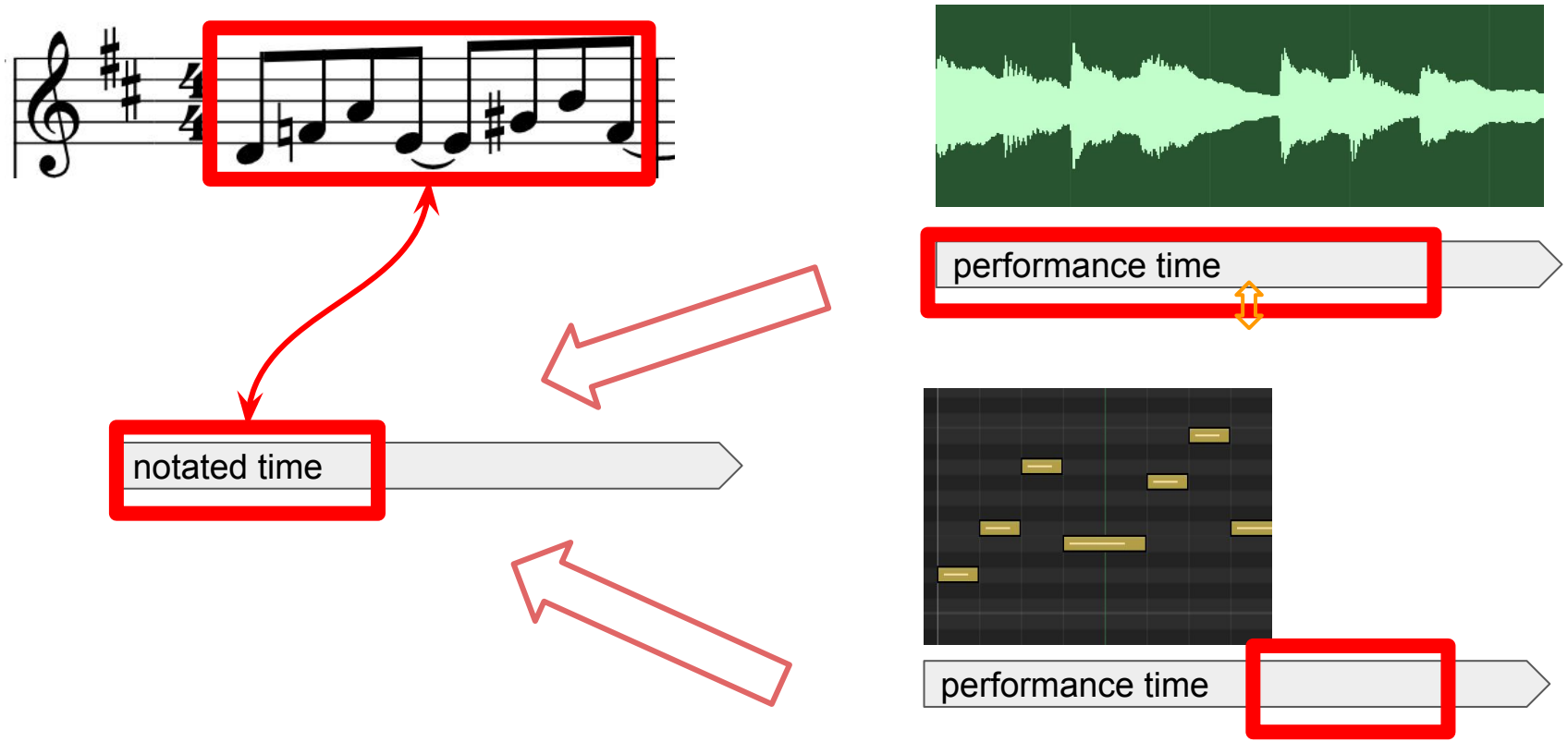
Rule 1: *Equal notated times refer to the same place in the score.*
Rule 2: *Greater notated times occur after smaller notated times in the score.*



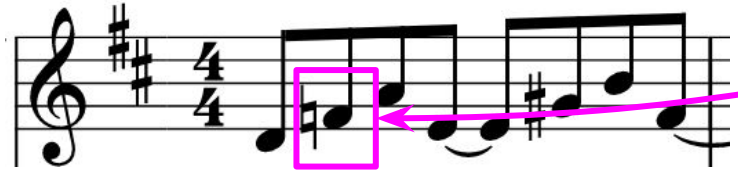
Why have notated time?

- Multiple performances can share the same notated time axis.
- Only one set of mappings from notated time to graphics and semantics is needed.
- Notated time is a *semantic time dimension*, so it lets us to map performances directly to semantics with no graphics needed.

Mapping performance to "notated time"

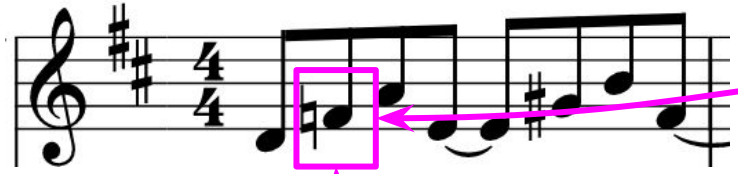


Linking Events, Graphics and Semantics

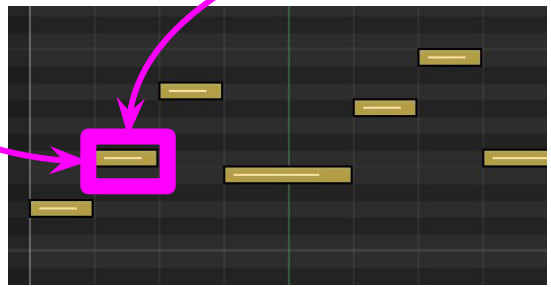


```
...
<event value="/8">
  <note pitch="F4"/>
</event>
<event value="/8">
  <note pitch="A4"/>
</event>
...
```

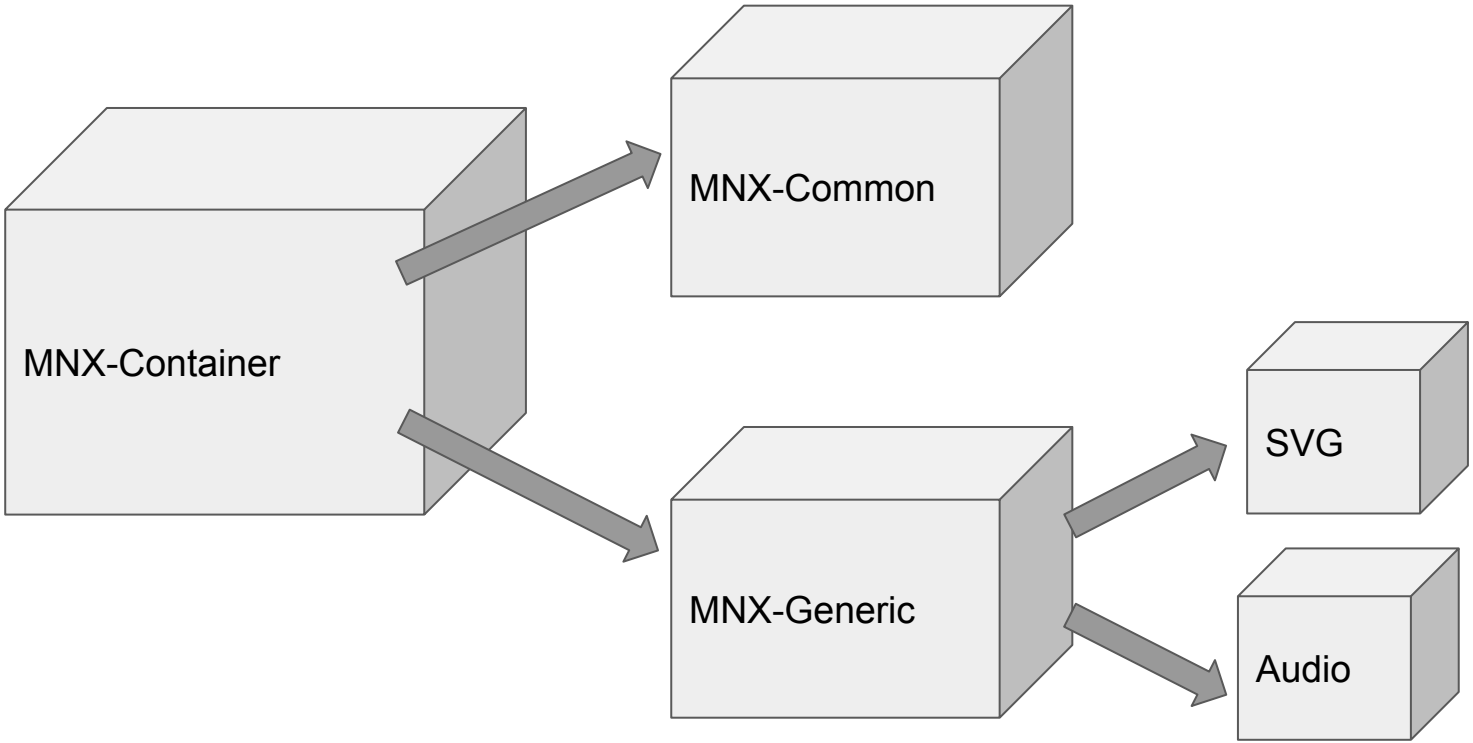
Linking Events, Graphics and Semantics



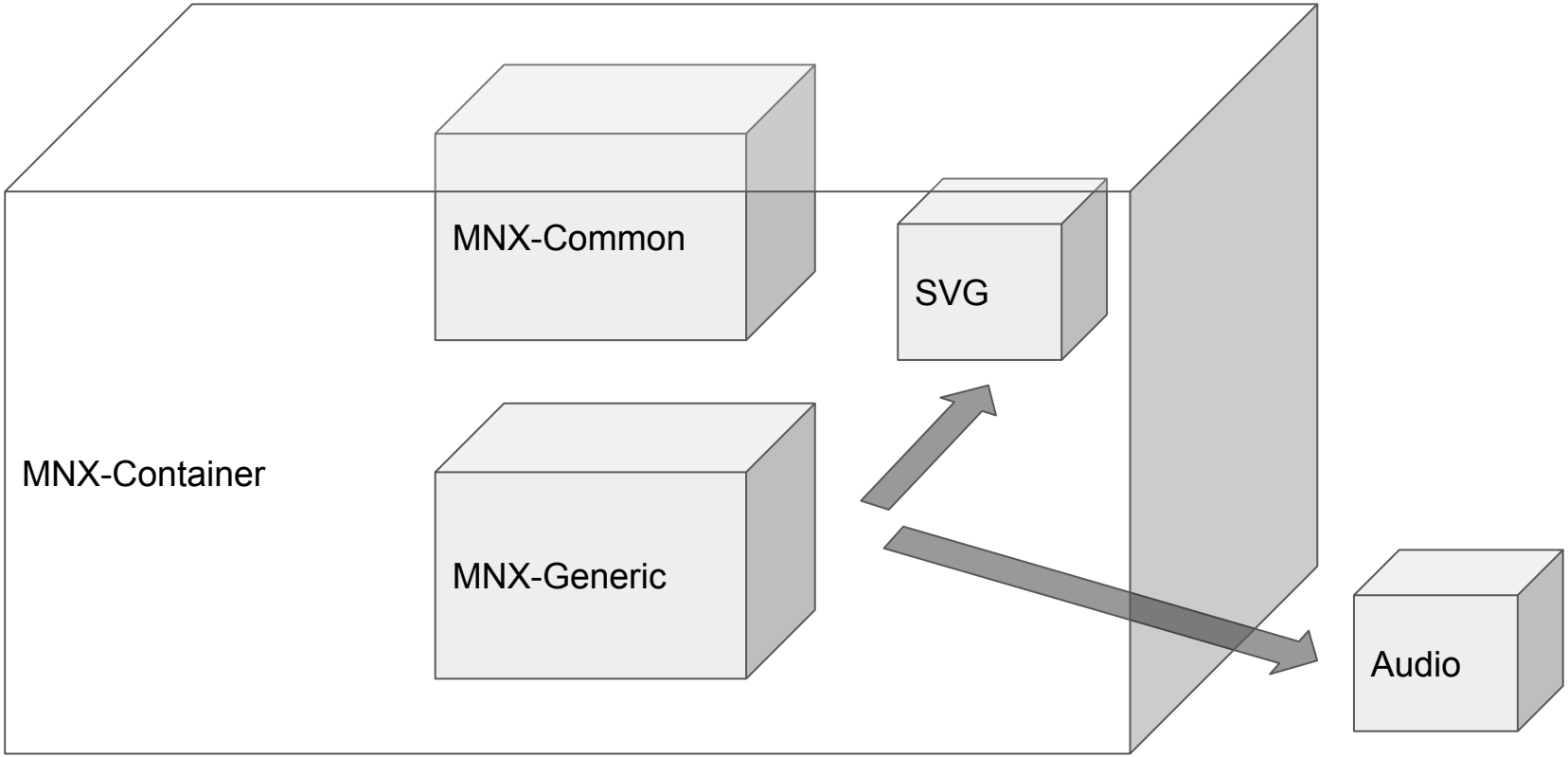
```
...
<event value="/8">
  <note pitch="F4"/>
</event>
<event value="/8">
  <note pitch="A4"/>
</event>
...
```



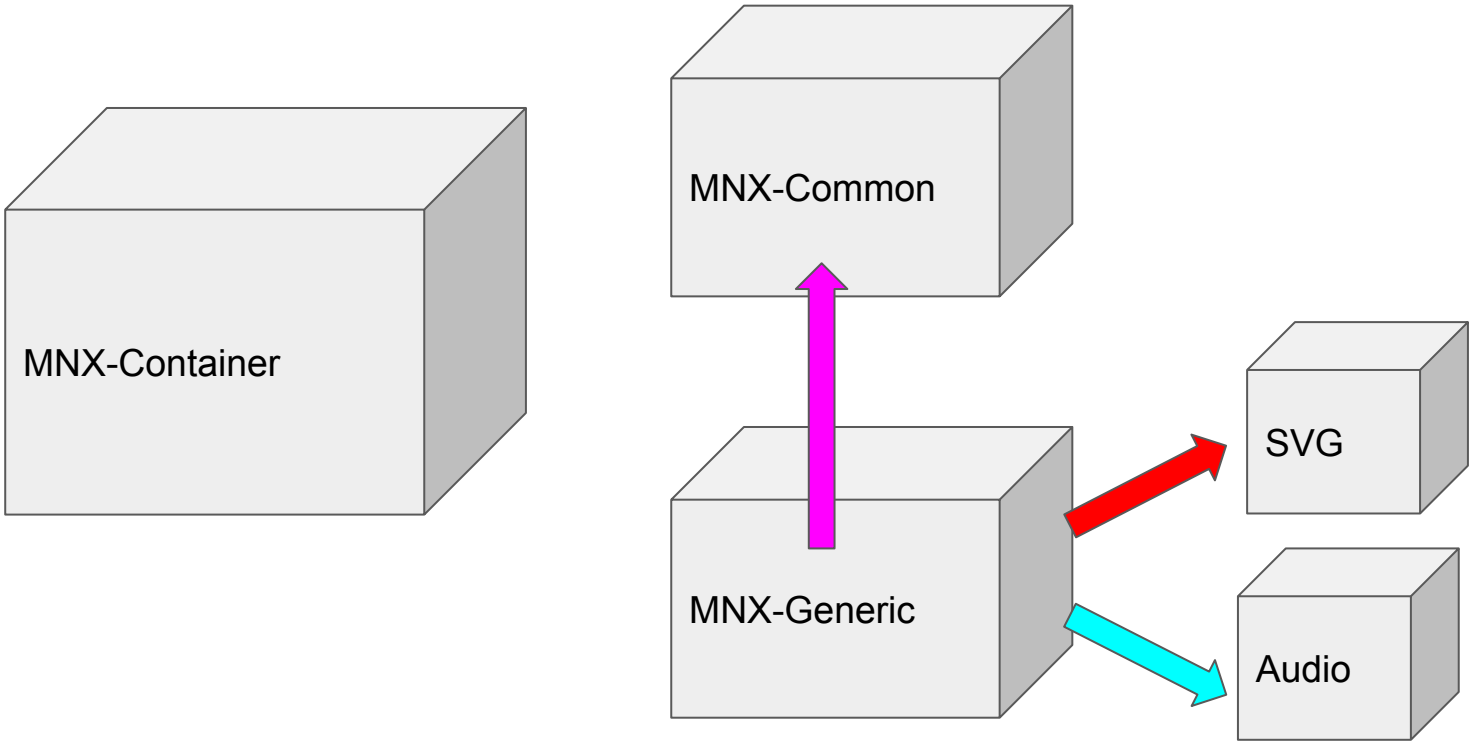
Packaging MNX Scores (separate files)



Packaging MNX Scores (bundled files)



Inter-document references



Graphics in MNX-Generic

- **<score-view>** element represents a page: a bunch of graphics intended to be viewed as a visual unit.
- "Plain old SVG" file format can be read/written by any tool
- SVG can be a simple wrapper around bitmap graphics
- Regions are bounding boxes of any SVG element within some <score-view>
- Cursors are connected sets of points in a region
- Any SVG element can represent any semantic element (e.g. an MNX-Common <event>)

Audio in MNX-Generic

- Audio media are files in existing standard audio formats
- **<performance-audio>** references a collection of synced tracks
- **<performance-audio-media>** references an individual track
- **<performance-mapping>** links media timeline to regions, cursors

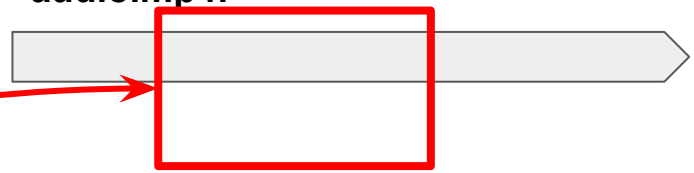
Performance data in MNX-Generic

- Performance data can be thought of as "MIDI-like"
- **<performance-data>** contains all the data for a performance
- **<performance-part>** contains the data for a part within it
- **<performance-event>** represents a single musical event (typically note)
- **<performance-mapping>** links media timeline to regions, cursors
- **<interpret>** includes performance data directly in MNX-Common

Region linkage

```
<mnx-generic>
  ...
  <performance-audio-media src="audio.mp4"/>
  <performance-region
     start="0.24" end="1.29"
     view="page1" region="region1"/>
  ...
  <score-view id="page1" src="view1.svg"/>
  ...
</mnx-generic>
```

audio.mp4:



view1.svg:

```
<svg>
  ...
  <g id="region1">...</g>
  ...
</svg>
```

Semantic linkage

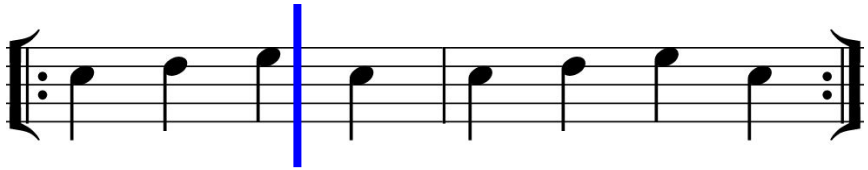
```
<mnx-generic>
  ...
  <score-view src="view1.svg">
    <score-mapping
      graphics="note1" semantics="e1n1"/>
  </score-view>
  ...
</mnx-generic>
```

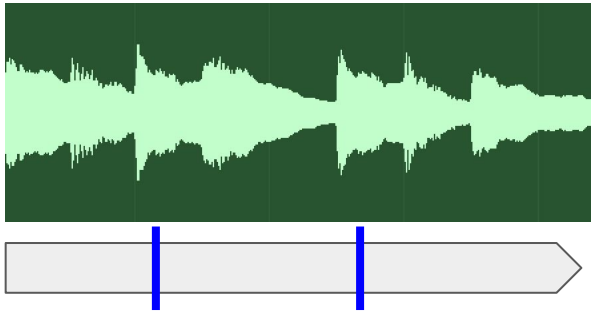
```
<mnx-common>
  ...
  <event value="/8">
    <note id="e1n1" pitch="C4"/>
  </event>
  ...
</mnx-common>
```

view1.svg:

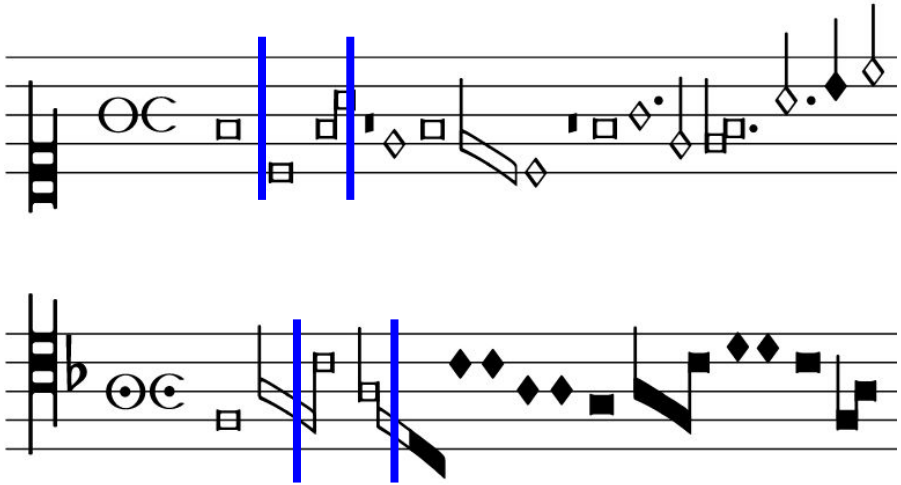
```
<svg>
  ...
  <g id="note1">...</g>
  …
</svg>
```

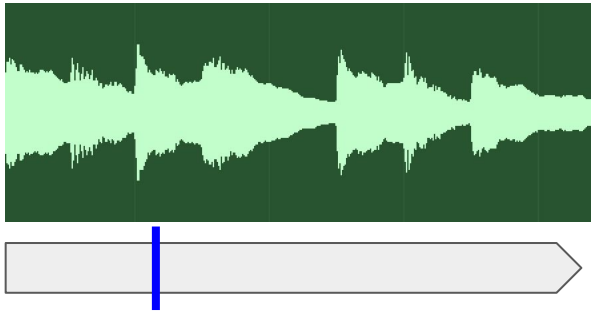
Notated Time and Cardinality: Form and Repeats





Notated Time and Cardinality: Ockeghem's *Missa Prolationum: Kyrie*





Why semantics isn't embedded in MNX-Generic

- Would force graphical structure to mimic semantic structure
- In its extreme form, would force performance structure to mimic graphical structure
- Would create a separate "Generic flavor" of every semantic encoding
- The same graphical object may belong to multiple semantic structures, in which case there is no possible unified structure.

The syncing controversy

- The performance/audio bits of MNX-Generic *almost* suffice to synchronize audio with semantic data
- Some applications will work directly from MNX-Common to render reflowable music, yet still want audio syncing.
- Alternatives in play so far:
  - Use MNX-Generic (minus graphics) to encode syncing for CWMN, and any other notational systems.
  - Use some of MNX-Generic inside MNX-Common (as in <interpret>)
  - Invent a brand new profile for MNX-Common with its own syncing concepts

Towards an MNX-Common layout model

Joe Berkovitz, Risible LLC
co-chair, W3C Music Notation Group

Questions to explore

- Is standardizing CWMN layout possible? Practical?
- Is there a logical sequence of "building blocks" that progress in this direction?
- What are the benefits and drawbacks at different points along this sequence?
- What's the CG's feeling about how far to go down this path?

Brief Recap: Layout in MusicXML 3.x

- Positioning model relies on absolute coordinates (**default-x, default-y)**
- **default-x** is brittle, cannot support reflow
- **default-y** works for reflow (but not transposition)
- **relative-x, relative-y** definitions have no defined origin
- Fine details of registration not defined (is y=0 centered on top staff line?)
- Implementation support for all of these is highly variable

What we're doing here

- Exposing potential ways to express layout more flexibly
- Using horizontal positioning as a "laboratory"
- Considering a spectrum of approaches, from loose to tight
- Look at how style properties could drive these approaches

Whom does layout serve?

- **Composer**: creates the musical content
- **Publisher**: applies a particular style and sensibility to the whole
- **Engraver**: applies human musical judgment to every detail
- **Performer**: reads the music in multiple environments and contexts

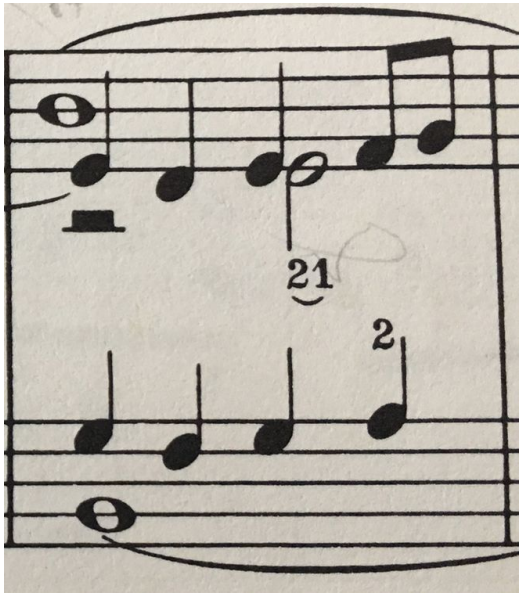
Serving the publisher and engraver

Several significant engraving decisions were made here, following from a potential stem/note collision in the upper staff.

These are human judgments that an algorithm would not reliably make.

Do we want such decisions to be preserved in a reflowable environment?

BWV 849, G. Henle Verlag



Levels of the Game

Depending on how far we go down this path, we can wind up in several "levels of the standards game". Each level builds on the previous one:

1. The Wild West
2. Explicit Positioning
3. Explicit Space Requirements
4. Algorithmic Space Requirements
5. Algorithmic Layout

Level 1: The Wild West

- Consumers do whatever they want
- Producers have no way to control consumers

If we stay at this level, there are no guarantees to producers about what consumers will show, and no concept of "layout compliance".

Level 2: Explicit Positioning

At this level, objects are positioned absolutely relative to some fixed point like the start of the measure. This would put MNX where MusicXML is today, but with added rigor.

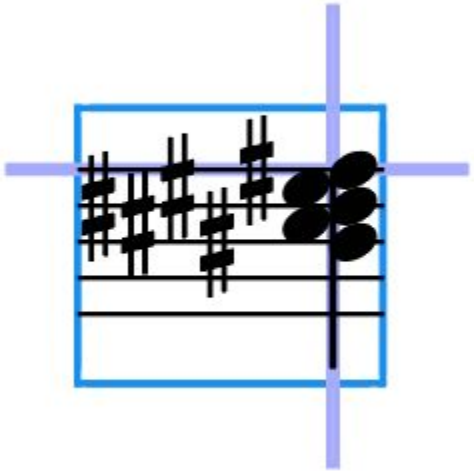
- Each object is put in an absolute position (a la **default-x** or **-y**)
- Consumers "slavishly" reproduce layout for one geometry only
- Extremely fine-grained: each event/direction has its own position
- Cannot survive editing or reflowing

Consumers may not pay attention to this data because it's so brittle and conflicts with their internal approach to flexible layout.

To go further, we need
a layout vocabulary...

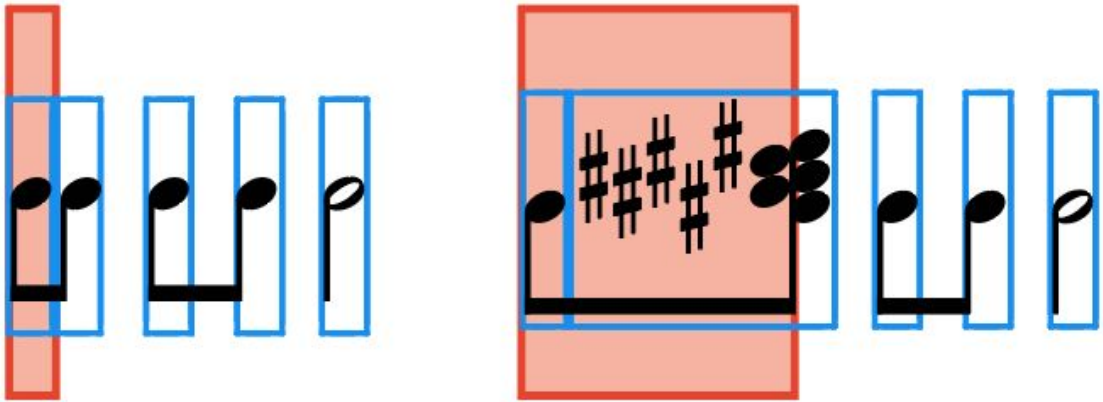
"Boxes"

Consider an event or direction. We can put a **box** around it, representing its fixed space requirements relative to a well-defined anchor point (x=beatline, y=top staff line). This can govern other layout decisions that we may choose to specify.



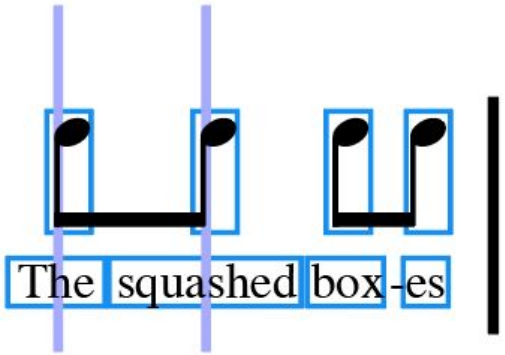
"Blocking Width"

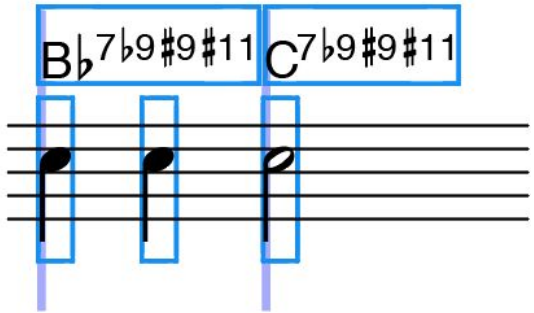
Given any pair of events or directions, there will be a minimum distance between their beatlines to avoid collisions. This distance determines a **blocking width** between objects, as shown below in red:



Interacting boxes

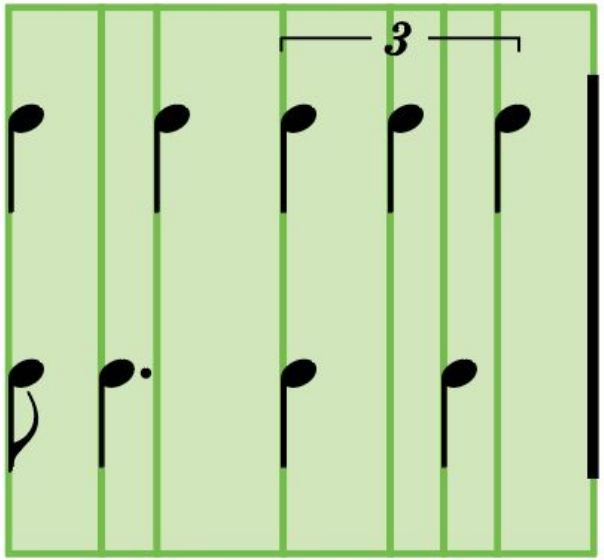
- Many kinds of boxes can block the positioning of events: chord symbols, lyrics, etc.
- Boxes in one "lane" of collision can affect layout in other "lanes".





"Sims" or "Simultaneities"

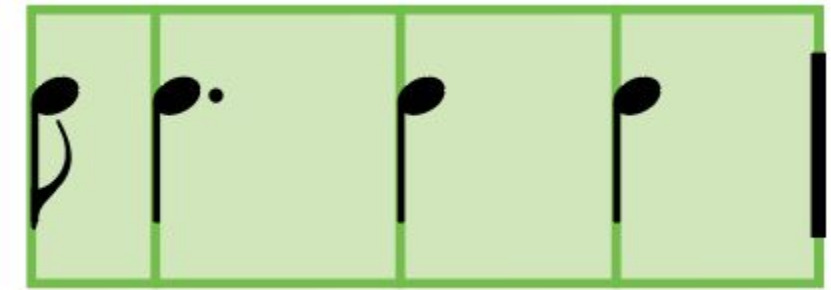
A **sim** or **simultaneity** is a horizontal extent bounded by one or more simultaneous events, directions or measure boundaries. The green boxes below show the sims in a simple polyphonic measure:



"Ideal width"

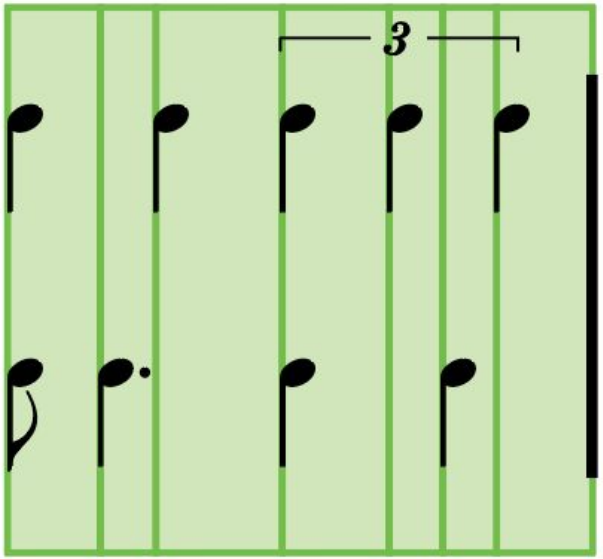
Each event or direction has an **ideal width**. Think of it as a weight relative to other objects, rather than an absolute value: the units are "stretchy".

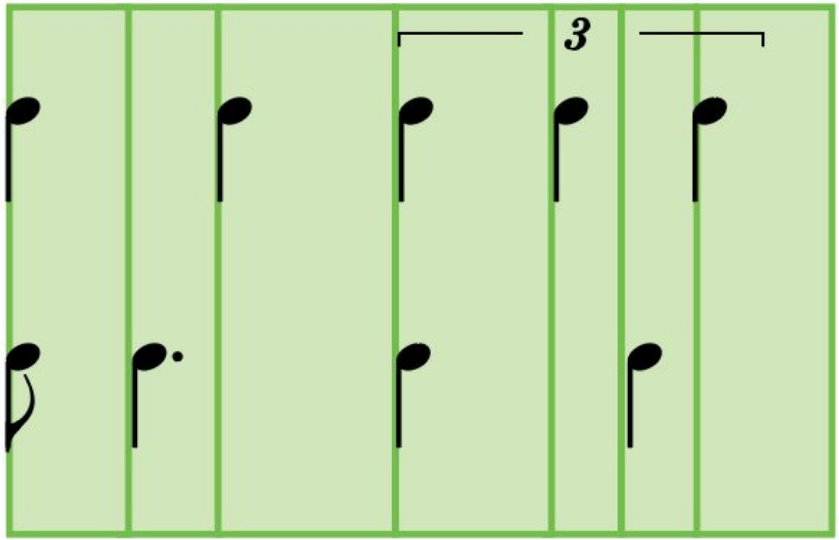
In the simplest case, each sim has only one event, and *vice versa*. The ideal width of the event, based on its duration, determines the width of the corresponding sim:



Sims can stretch

An ideal width is just an ideal. In nearly every case, the ideal width will be stretched (or shrunk) to optimize the view of the music, for example, to justify a system. You can think of this as deciding the visual unit of ideal width.

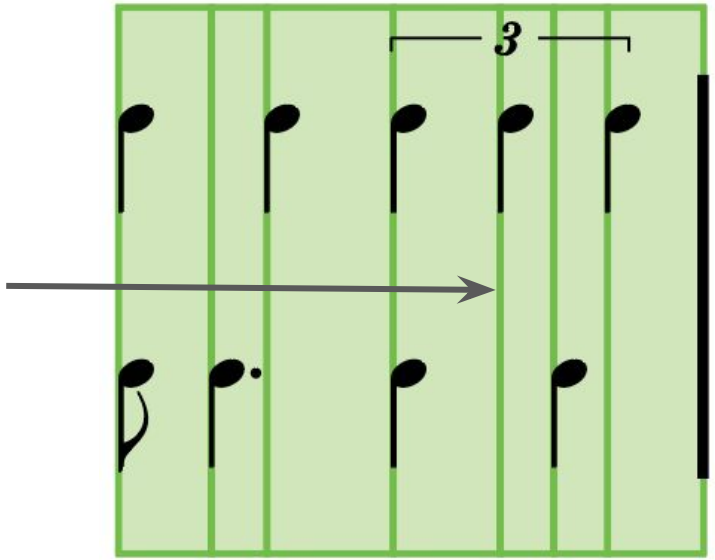




Sims coincide with some events, and divide others

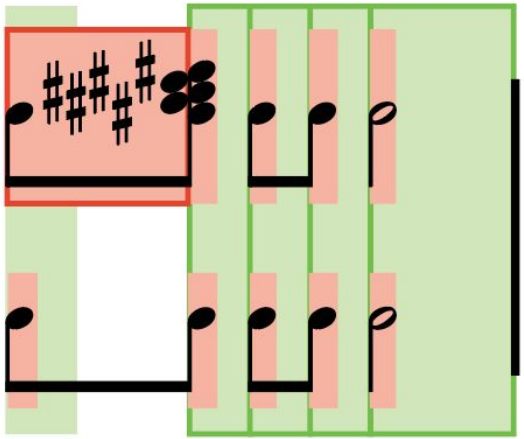
Polyphonic case: each sim may coincide with, and intersect with, multiple events. These events have their own individual ideal widths, and the sim's ideal width is a function of them.

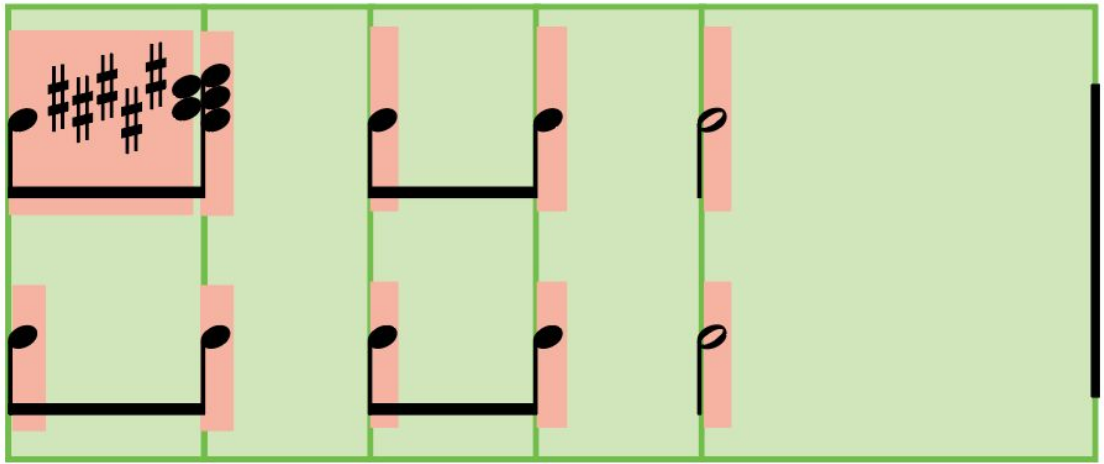
This sim begins an event in the upper voice, and intersects an event in the lower. The events will have different "ideal widths".



Combining it all: blocking/ideal widths and stretching

- Ideal widths are assigned definite units, thus stretching/shrinking them.
- Each sim has a blocking width determined by nearby boxes
- Each sim takes up the maximum of its stretched width and its blocking width.





Back to our roadmap...

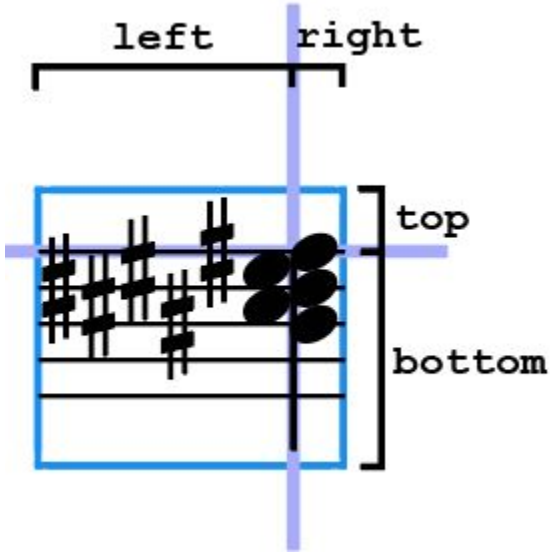
Level 3: Explicit Space Requirements

At this level, producers attach style properties that state the space that each object requires, and consumers honor this information within their own approach to layout.

- Consumers reproduce layout "reasonably" well
- Consumers make use of the space requirements in varying ways
- Extremely fine-grained and verbose (positioning for each event/direction)
- Can be reflowed
- Cannot survive editing

Boxes: Explicit space requirements

At this level, scores explicitly state how much space every box takes up relative to its origin. Verbose, but it does the job and doesn't require an algorithm.



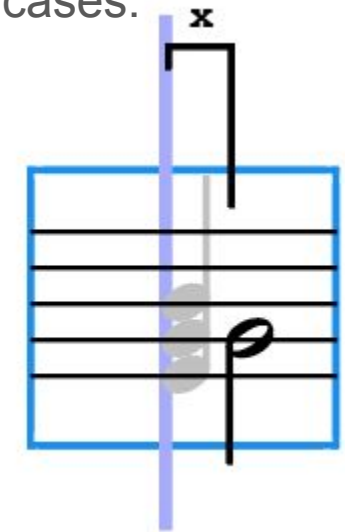
Boxes: Explicit space requirements

Using styles, this approach might be encoded like this:

```
<event value="/4"
       style="left: …; right: …; top: …, bottom: …">
```

Intra-box positioning

We'll also need to position objects relative to their origin, e.g. for a crossed voice, or to displace a lyric to the left or right. These values can likely be explicit, as rules may not work well across many cases.



Sims: Explicit Space Requirements

- Each event's ideal width is prescribed by a style property, e.g.:

  ```
  <event value="/4" style="ideal-width: 5">...</event>
  ```

- This is separate from the box dimensions: we are saying, "here is the horizontal space ideally occupied by the event, ignoring its box".

- In the monophonic case, event ideal width is same as sim width.
- In the polyphonic case, it's more complicated (but still simple!)

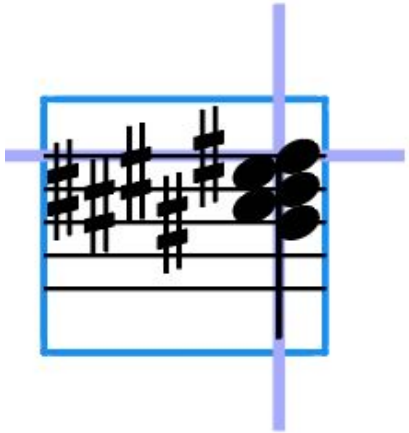
Level 4: Algorithmic Space Requirements

At this level, producers can provide a set of style parameters that allows any object's space requirements to be determined algorithmically. Document-wide parameters establish a "house style" that can be overridden for measures, sequences or specific objects.

- Consumers reproduce layout reasonably well
- There is still variation in how consumers make use of the space requirements.
- Document can specify the look of a score once, at a high level
- Individual objects may still override with their own requirements
- Can be reflowed
- Layout decisions survive editing

Boxes: Algorithmic space requirements

We can specify styles that control how the interior of a box is laid out, and derive the box dimensions by applying them. This isn't a full music layout algorithm, but it specifies how a box's contents are laid out: noteheads, stems, accidentals, dots...



```
accidental-padding
stem-length
accidental-ordering
```

Horizontal Layout: Algorithmic Space Requirements

- Each event's ideal width is determined by a algorithm based on its duration, using a house style property.
- Can use table lookup with interpolation to avoid prescribing specific math.
- At document level, a style provides a table, *e.g.*:

$\log_2$ duration (note value)	ideal width in staff lines
0 (whole)	5
-1 (half)	4
-2 (quarter)	3
-3 (eighth)	2

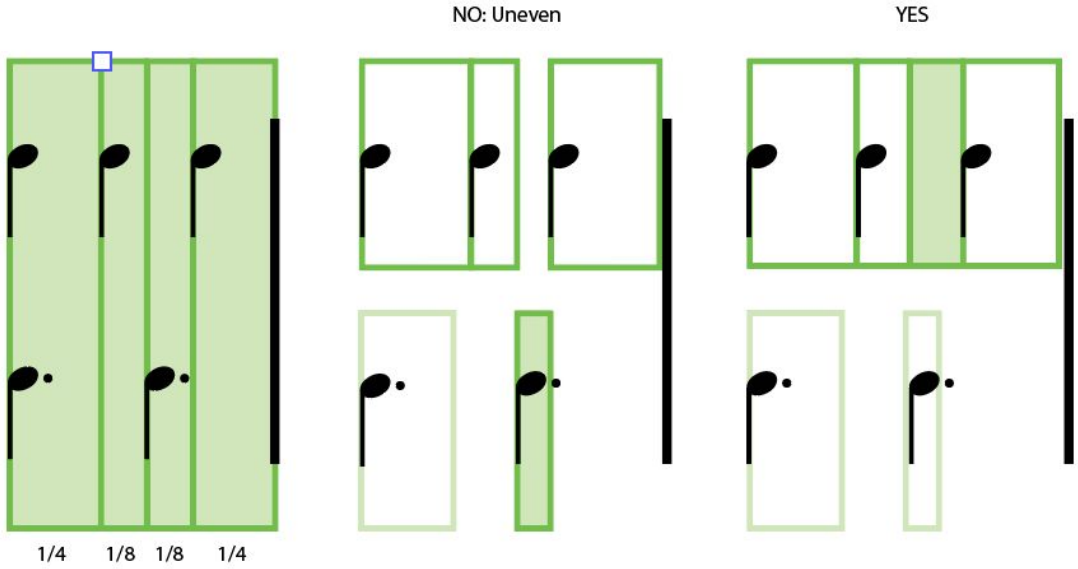
Level 5: Algorithmic Layout

At this level, both consumers and producers employ a layout algorithm that produces deterministic results based on the space requirements from Level 4.

- Consumers reproduce layout perfectly
- Document can specify the look of a score once, at a high level
- Can be reflowed
- Layout decisions survive editing

Algorithmic Topic 1: polyphonic ideal widths

- Events are assigned ideal widths based on styling.
- Each event contributes a pro-rated portion of its ideal width to each sim.
- Each sim assumes the **maximum width** contributed by any of its events.



Algorithmic Topic 2: Blocking width across sims

- Blocking can occur across multiple sims
- In this case, the blocking width can be allocated in a pro-rated fashion