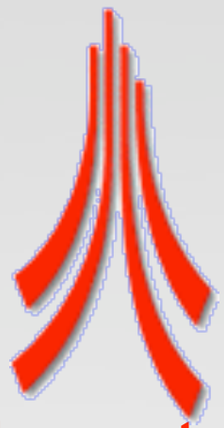
Remixing Music Pythonically

Adam T. Lindsay

PyCon 2010, Atlanta, GA, USA

@atl

http://atl.me/remix



Lancaster University

Introduction

- What's the Echo Nest Remix API?
- Why change it?
- Try to make the API more fluent/expressive
- Try to make the API more capable with multitrack/effects
- How's that working out?

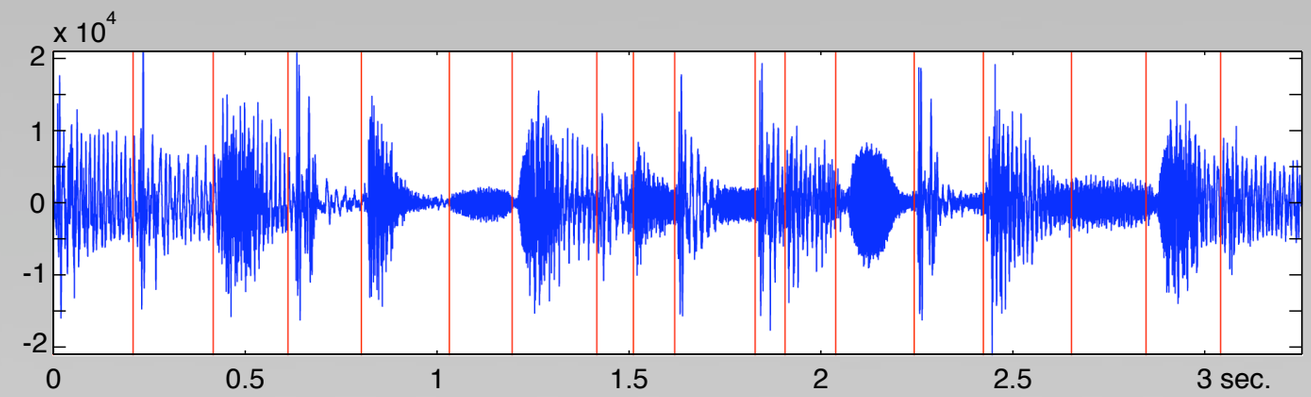
the echonest

http://the.echonest.com/

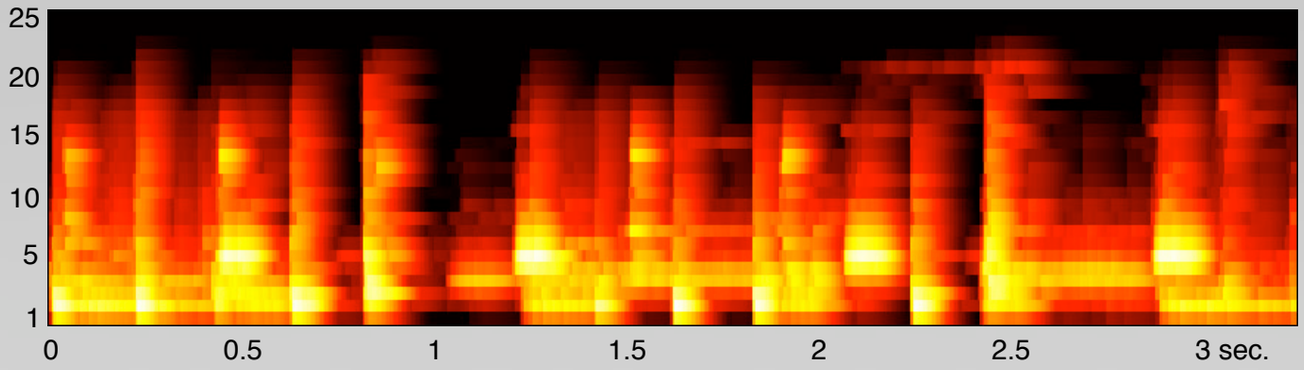
The Echo Nest

- "Music Intelligence" company based near Boston
- Data feeds, Recommendation engine…
- Music listening API ("Analyze")
  - Oriented towards mixed/recorded music
  - Global features
    - Key, tempo, meter, &c

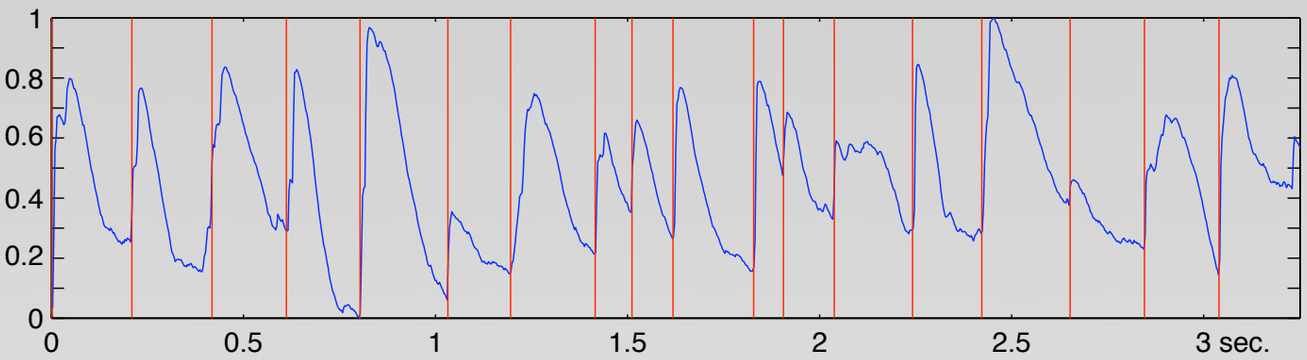
waveform



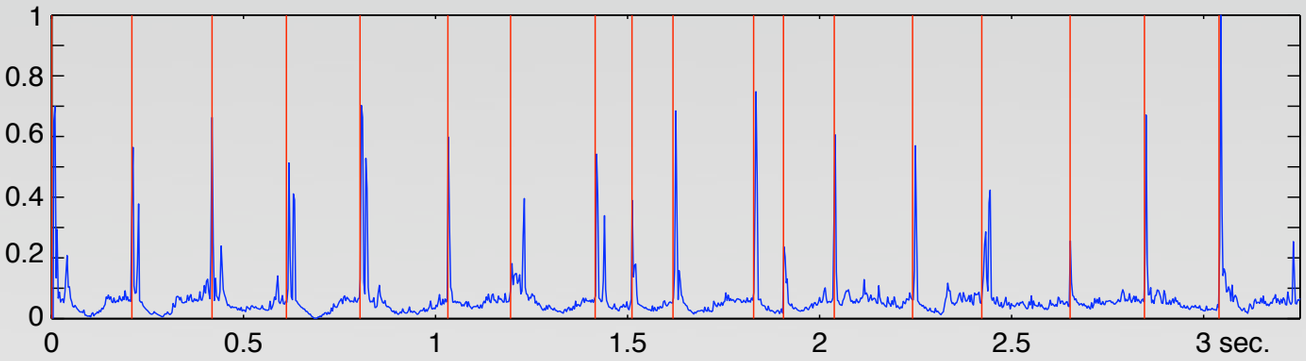
auditory spectrogram



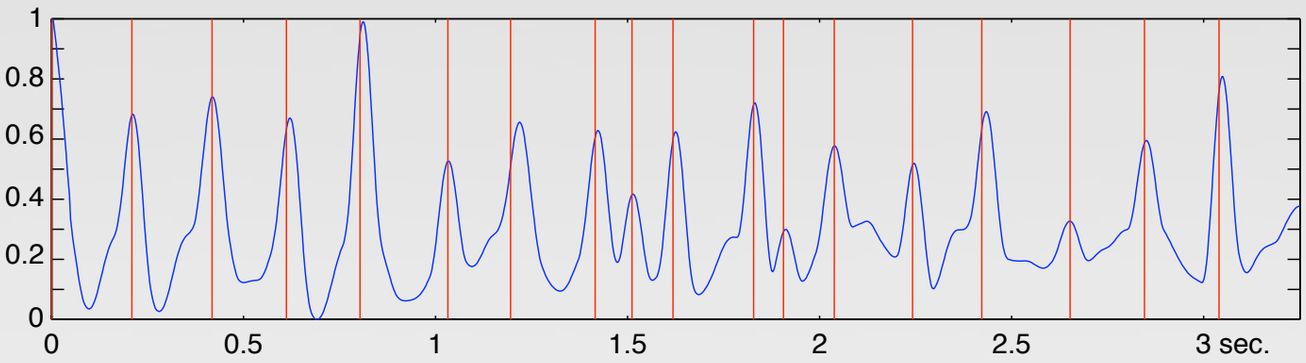
loudness curve



raw detection function



filtered detection function



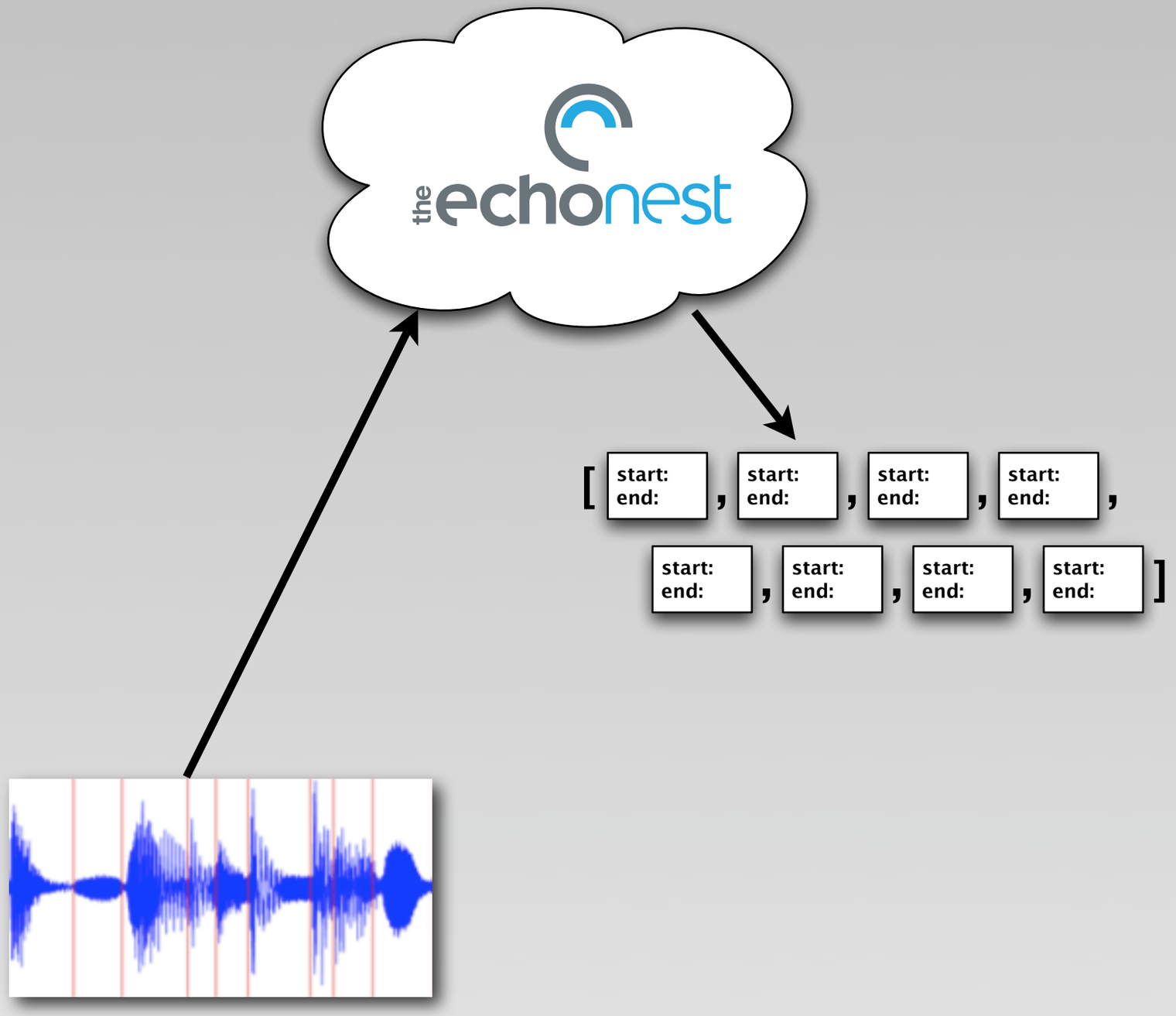
The Echo Nest Remix API

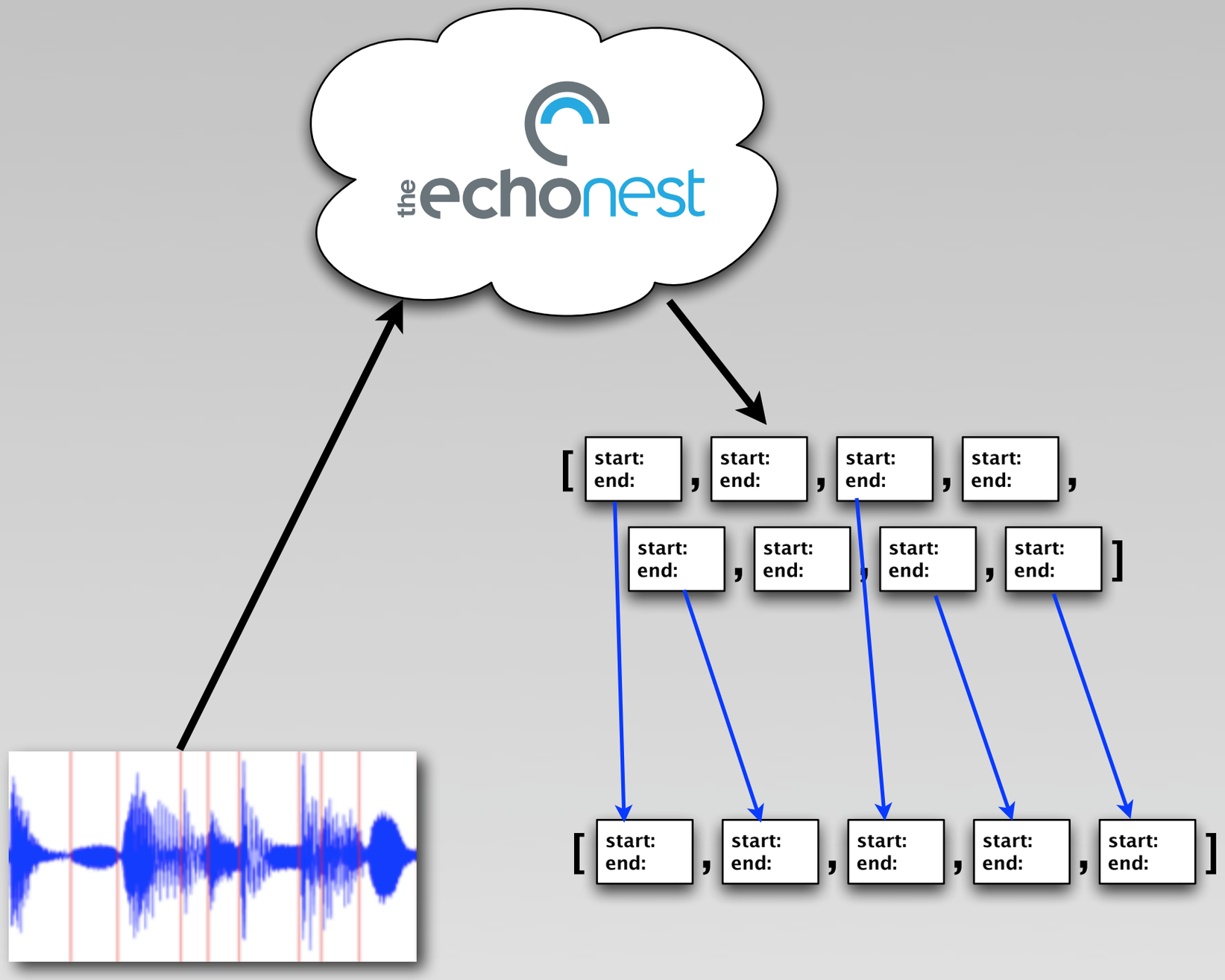
- *…as originally released*
- Python libraries, using **`NumPy`**
- Loads audio from file
- Uploads to Web API
- Uses Analyze transparently via the web
- Glues together metadata and audio

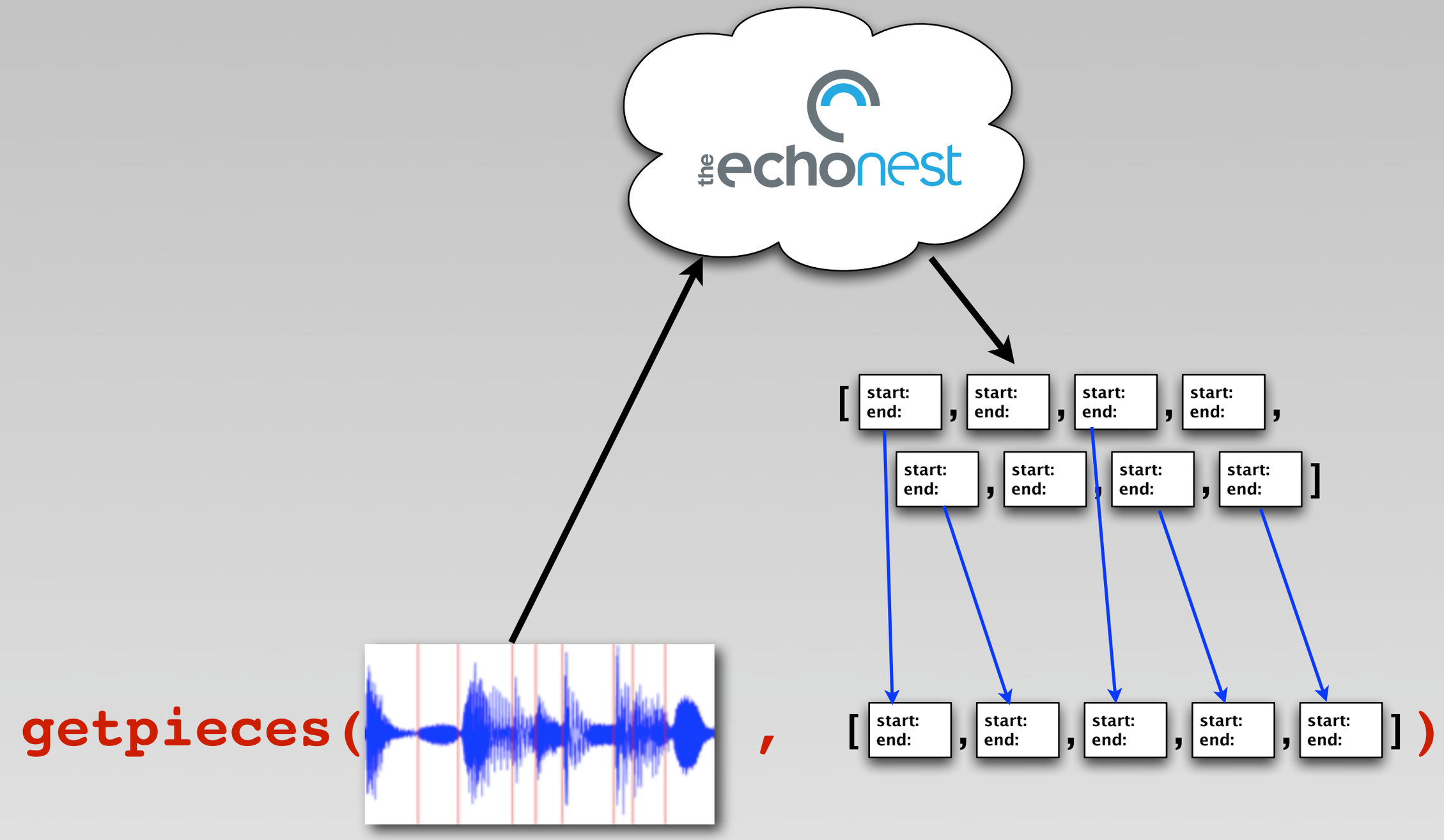
Genesis of Remix API

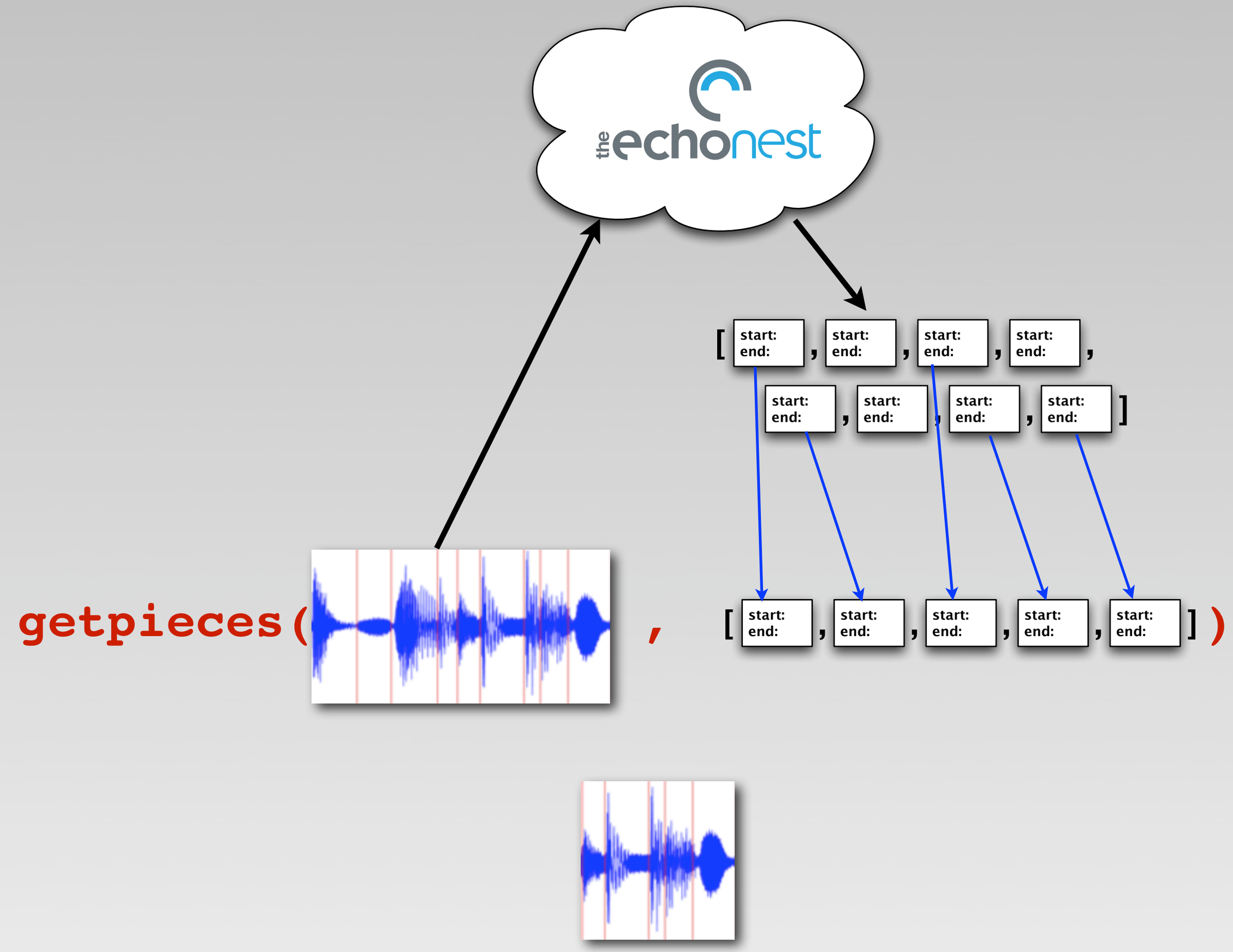
- Key abstraction: **`AudioQuantum`** class, unifying rhythmic units & segments
  - Start & duration
  - Audio output
    - Make a **`list`** of **`AudioQuantums`**
    - Collect the samples from the original file, indexed by the start/duration in each **`AudioQuantum`**


the echonest


the echonest
[start: end: , start: end: , start: end: , start: end: ,
start: end: , start: end: , start: end: , start: end:]


the echonest
[start: end: , start: end: , start: end: , start: end: ,
start: end: , start: end: , start: end: , start: end:]
[start: end: , start: end: , start: end: , start: end: , start: end:]


the echonest
[start: end: , start: end: , start: end: , start: end: ,
start: end: , start: end: , start: end: , start: end:]
getpieces(
,
[start: end: , start: end: , start: end: , start: end: , start: end:])


the echonest
[start: end: , start: end: , start: end: , start: end: ,
start: end: , start: end: , start: end: , start: end:]
getpieces(
,
[start: end: , start: end: , start: end: , start: end: , start: end:])

Modifying Remix

- Towards a more declarative style
  - Filtering, selection, sorting, navigation
- Flexibility in multiple dimensions
  - Many sources, effects, rendering

SLAP CHOP

Keeping America Skinny one slap at a time

Making Remix more Fluent

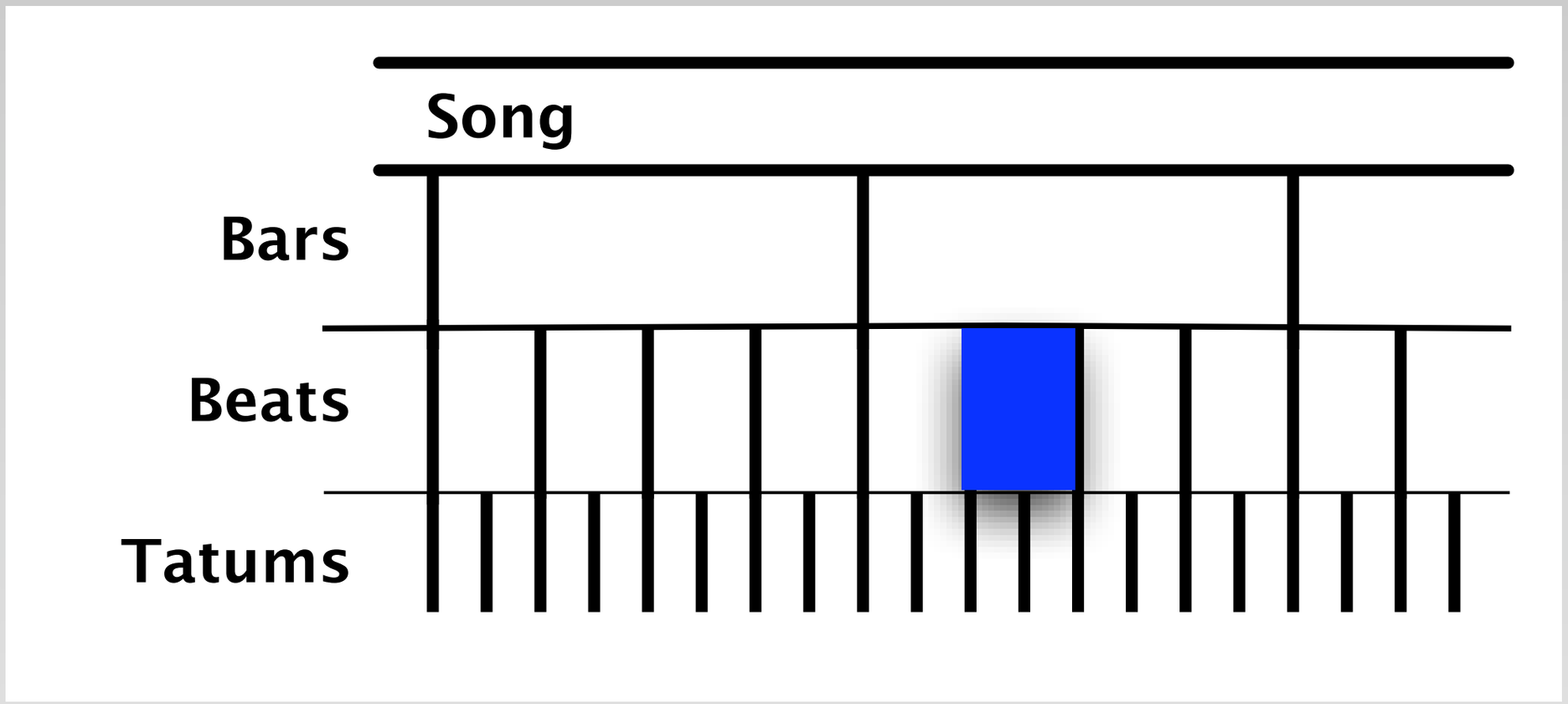
- Introducing the **`AudioQuantumList`** class
- Clear that the typical pattern was to create lists from source lists
- Filtering, Selection, Sorting on various features
- The filtering method is "**`that()`**"
- Makes for interesting naming (and definition) of filters

```
mybeats = song.analysis.beats.that(fall_on_the(1))
```

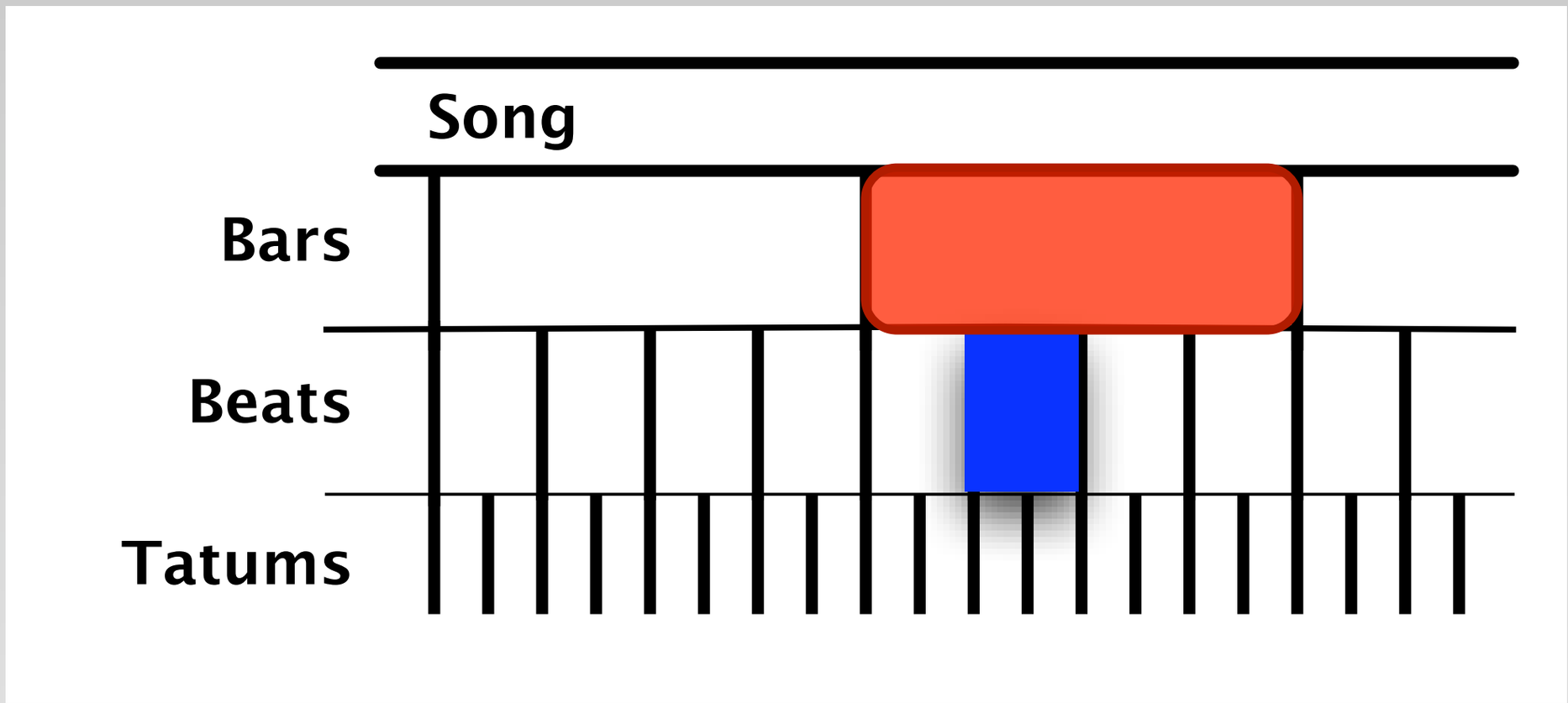
Granting contextual knowledge

- Implement back-links from **`AudioQuantum`** to **`AudioQuantumList`**
- The rhythm hierarchy is exposed via a series of "**`that()`**" queries on selected **`AudioQuantumLists`**
  - **`tatum.parent()`** → beat
  - **`beat.parent()`** → bar

Navigation

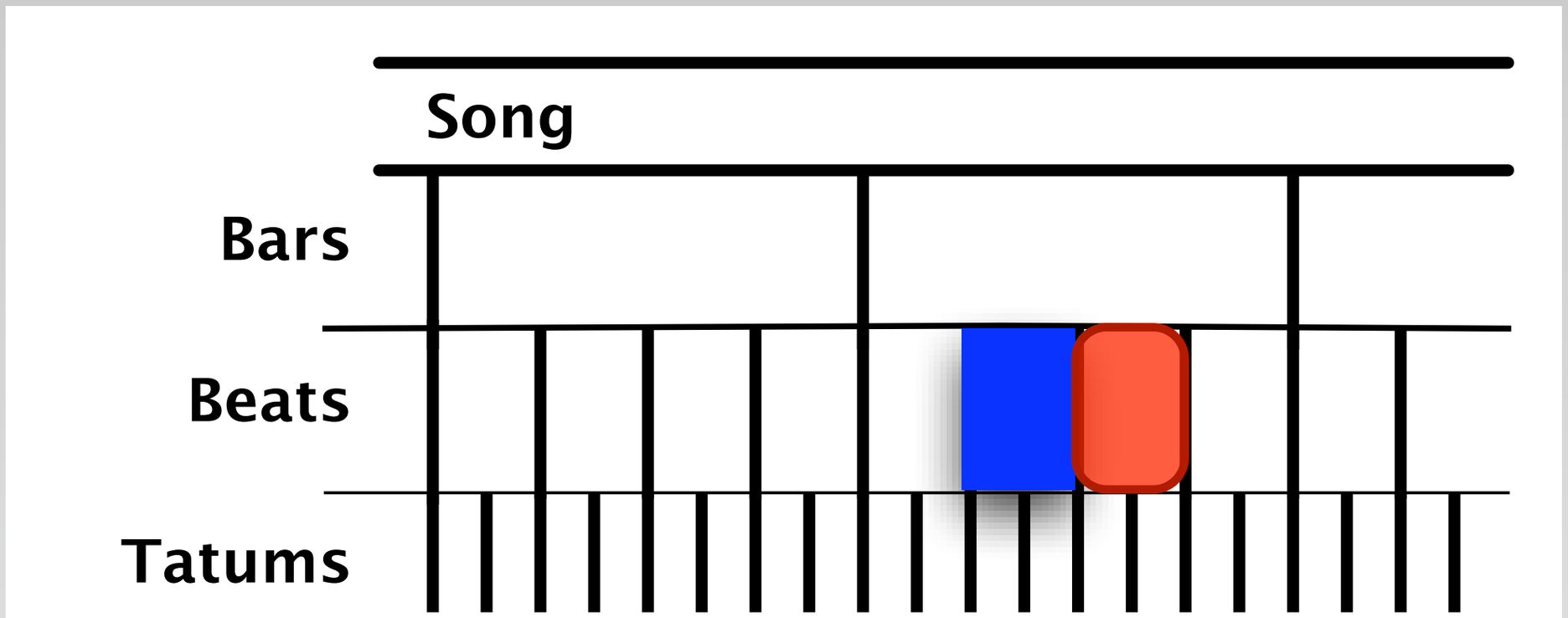


Navigation



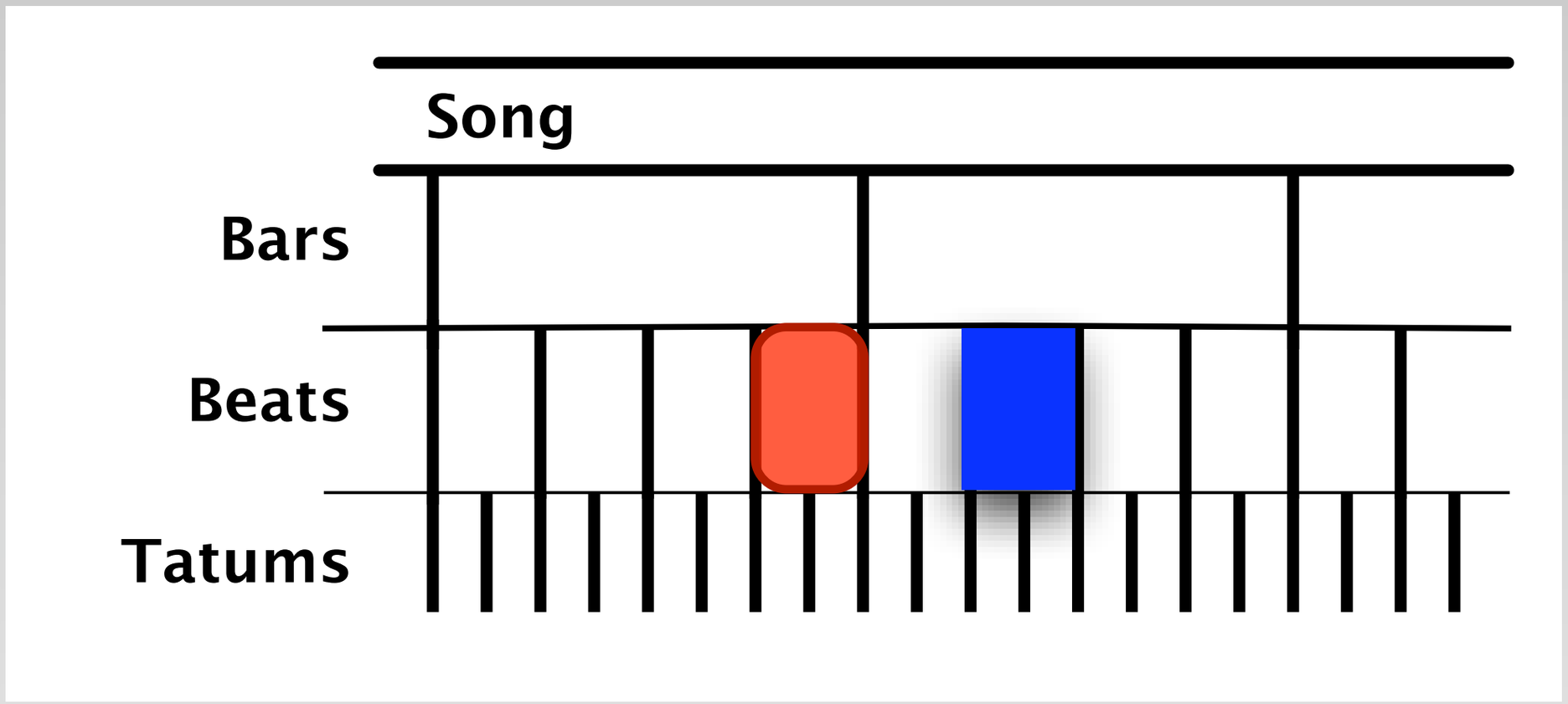

```
beat.parent()
```

Navigation



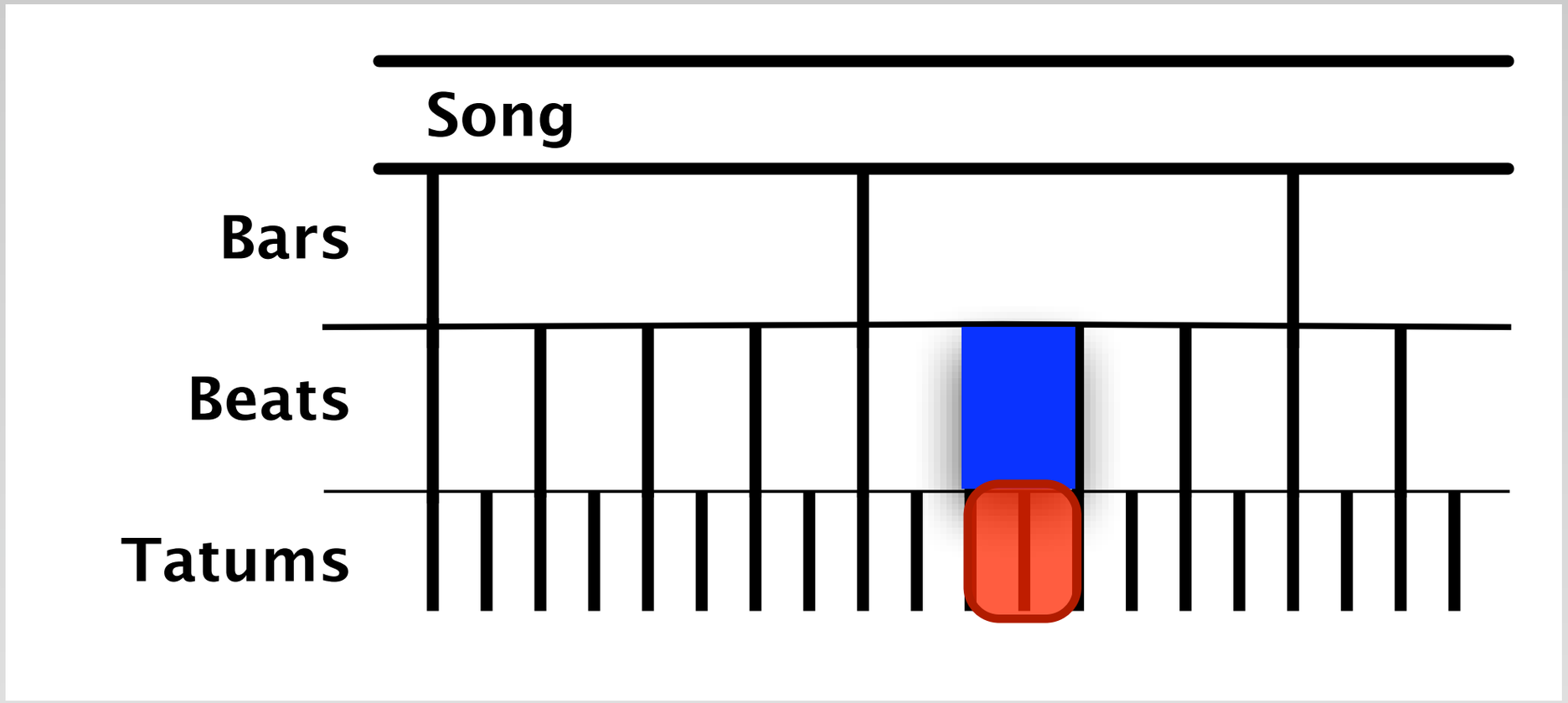

```
beat.next()
```

Navigation



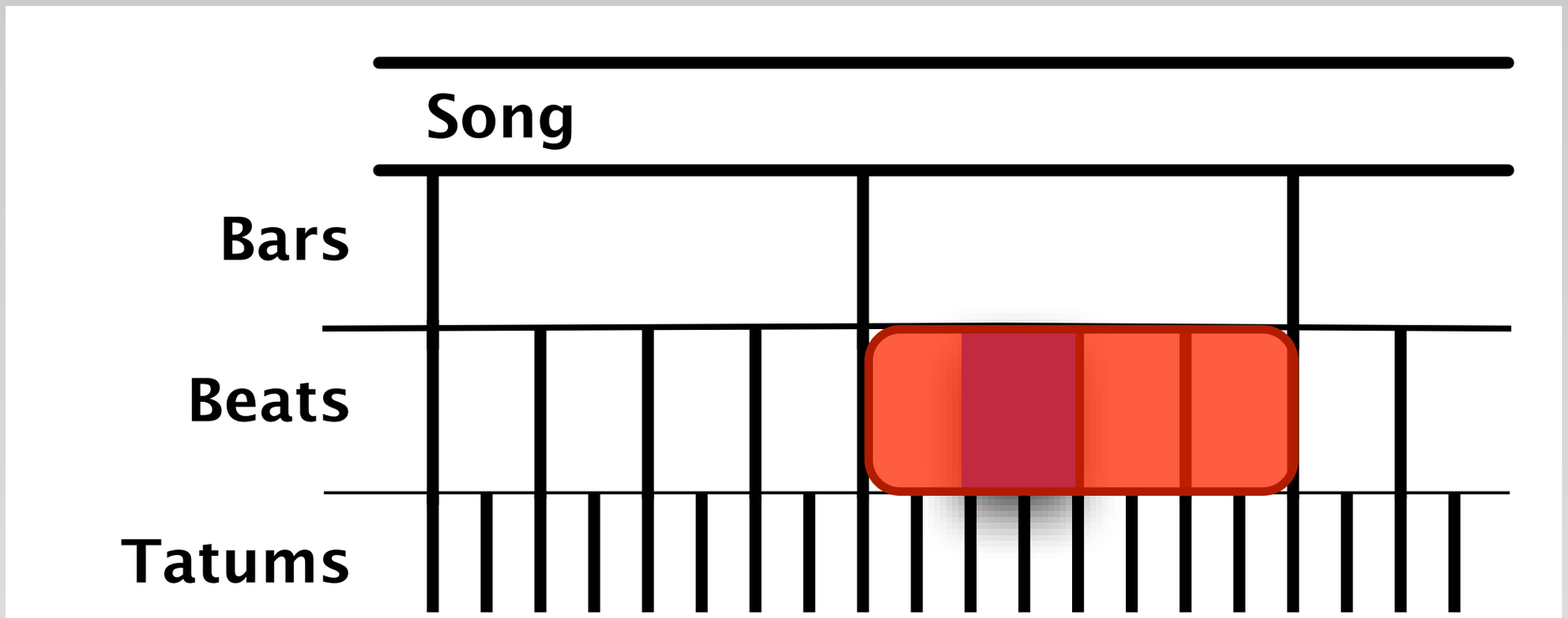

```
beat.prev(2)
```

Navigation




```
beat.children()
```

Navigation



beat.group()

More fluency

- **`sorted_by()`**

```
segments.sorted_by(timbre_distance_from(x))
```

- **`beget()`** allows you to return an arbitrary list

- **`changed_by()`** changes in place (on a conditional)

```
remix = song.analysis.beats                    \
    .changed_by(adding('snare.wav'),
        if_they=fall_on_the(2))
```

Multidimensional

- What if you want to remix more than one source?
- The `Simultaneous` class
    - An `AudioQuantumList` of elements that sound at the same time
- Experiment with a rendering pass
    - Good with a couple sources

http://www.flickr.com/photos/futurilla/3567392514/

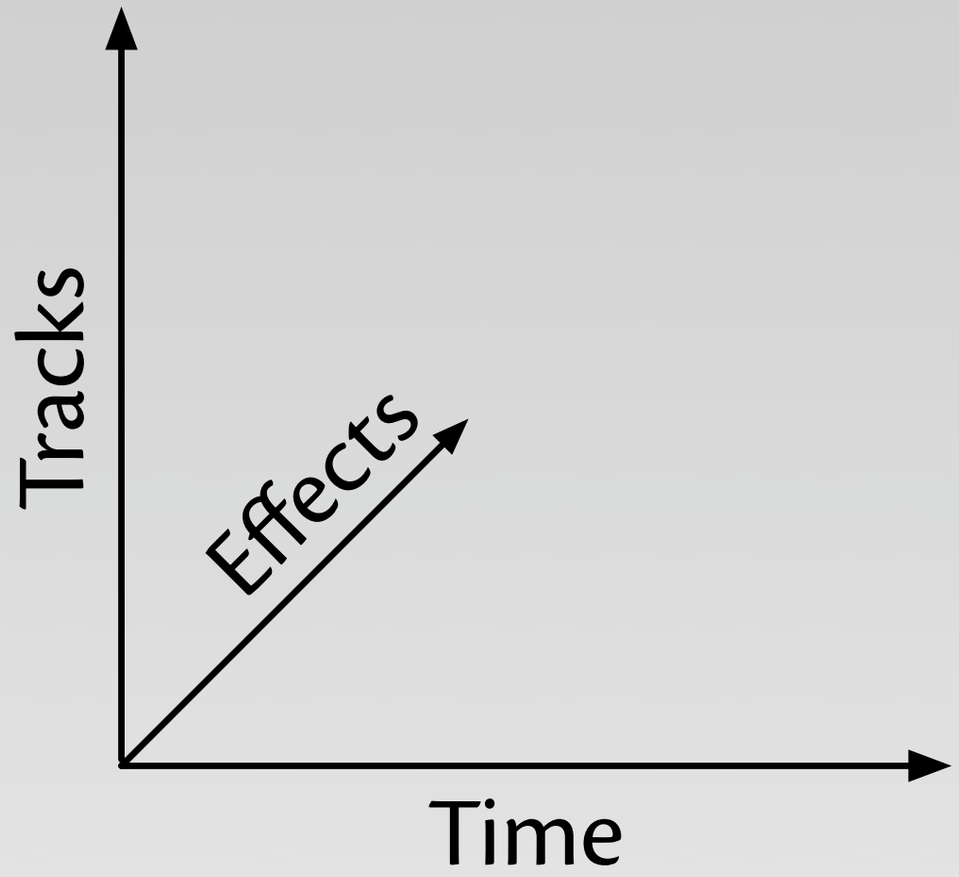
The rendering chain

- Walk a list of lists
  - Necessitates lazy loading (and thinking about memory management)
- Accumulate the results
  - Use a 32-bit sample array for lazy users
- Normalise and output

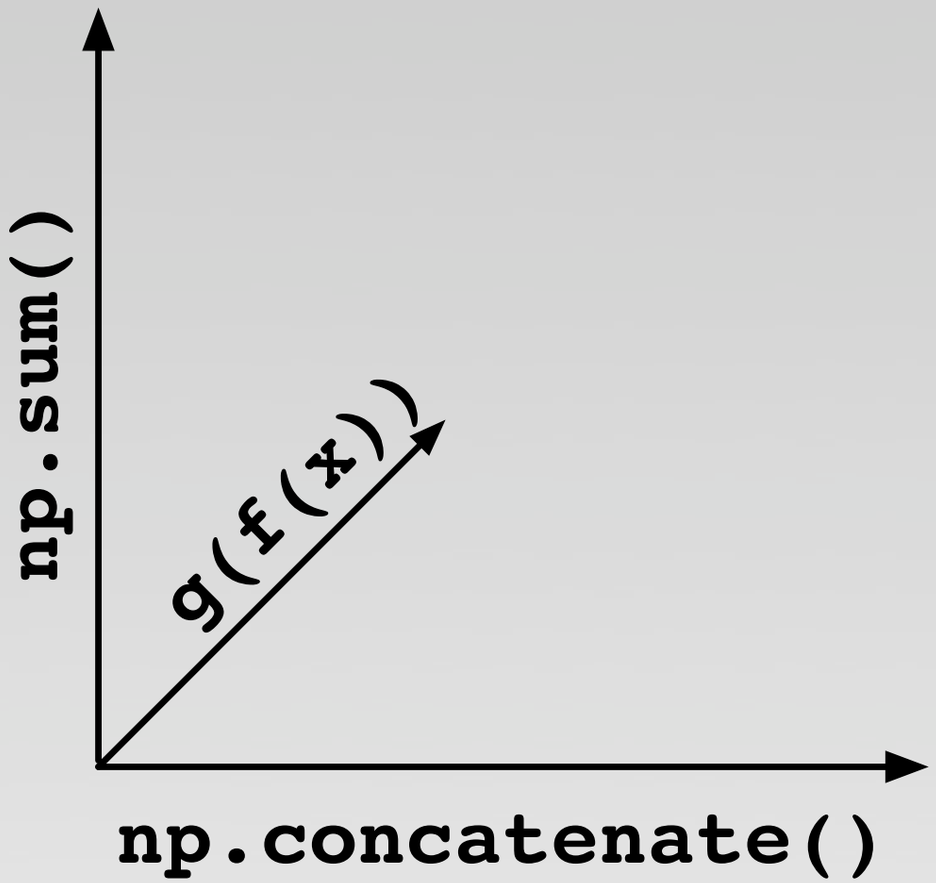
Audio effects

- The **`UserAudioQuantum`** was born
  - A sort of prototype-object, inheriting from (*proxying*) the original segment
    - Created upon user access
- Causally-connected self-representation
- Insert modification functions into the rendering chain

Three-dimensional remixing



Three-dimensional remixing



What happened?

- A declarative, fluent, natlang approach
- More connections, exposed with syntactic sugar
- Change from eager, linear rendering to something three-dimensional & lazy
- A framework for defining and applying effects

```
from echonest import audio

audio.AudioFile("input.mp3").analysis \
   .beats                             \
  .changed_by(reversing,              \
    if_they=fall_on_the(4))           \
  .encode("output.mp3")
```

Win/Fail?

- Filtering methods
- Navigation
- Multiple sources
- Effects

Win/Fail?

- Filtering methods
- Navigation
- Multiple sources
- Effects

???

Where?

http://go.atl.me/remix-api

http://atl.me/remix

@atl