Sound and Music Computing: Rhythm Analysis and Music Mashups

Matthew Davies
Sound and Music Computing Group
INESC TEC, Porto

Sistemas Multimedia, 28/04/15

About me

- PhD, Centre for Digital Music, QMUL, UK (2003-2007)
- Post-Doctoral Researcher
    - QMUL (2007-2011)
    - INESC (2011-2013, 2014-)
    - AIST, Japan (2013)
- SMC Group - http://smc.inescporto.pt
- My website - http://telecom.inescporto.pt/~mdavies

Sound and Music Computing Group

- 11 Full-time researchers
  (Post-doc, PhD and Masters)
- Two Main Research areas
  - **Music Description**
  - **Music Generation**
  - Particular expertise in analysis of musical **rhythm**

Research Motivation

- To extract musical information from audio signals…
  and then use it!

- Draws on many academic disciplines:

  - Digital Signal Processing, Machine Learning, Musicology, Physics, Mathematics, AI, Information Retrieval, Psychoacoustics, Music Psychology, Computer Science, etc…

- In particular for **SMC**: leverage high expertise in **music** with high expertise in **engineering / signal processing**

What kinds of musical information?

- We try to extract many things from the musical audio signal:

  - **temporal/rhythmic**: onsets, beats, metre, structure

  - **harmonic**: key signature, chords

  - **timbral**: instruments

  - **higher-level attributes**: mood, style, genre

Focus of today’s talk

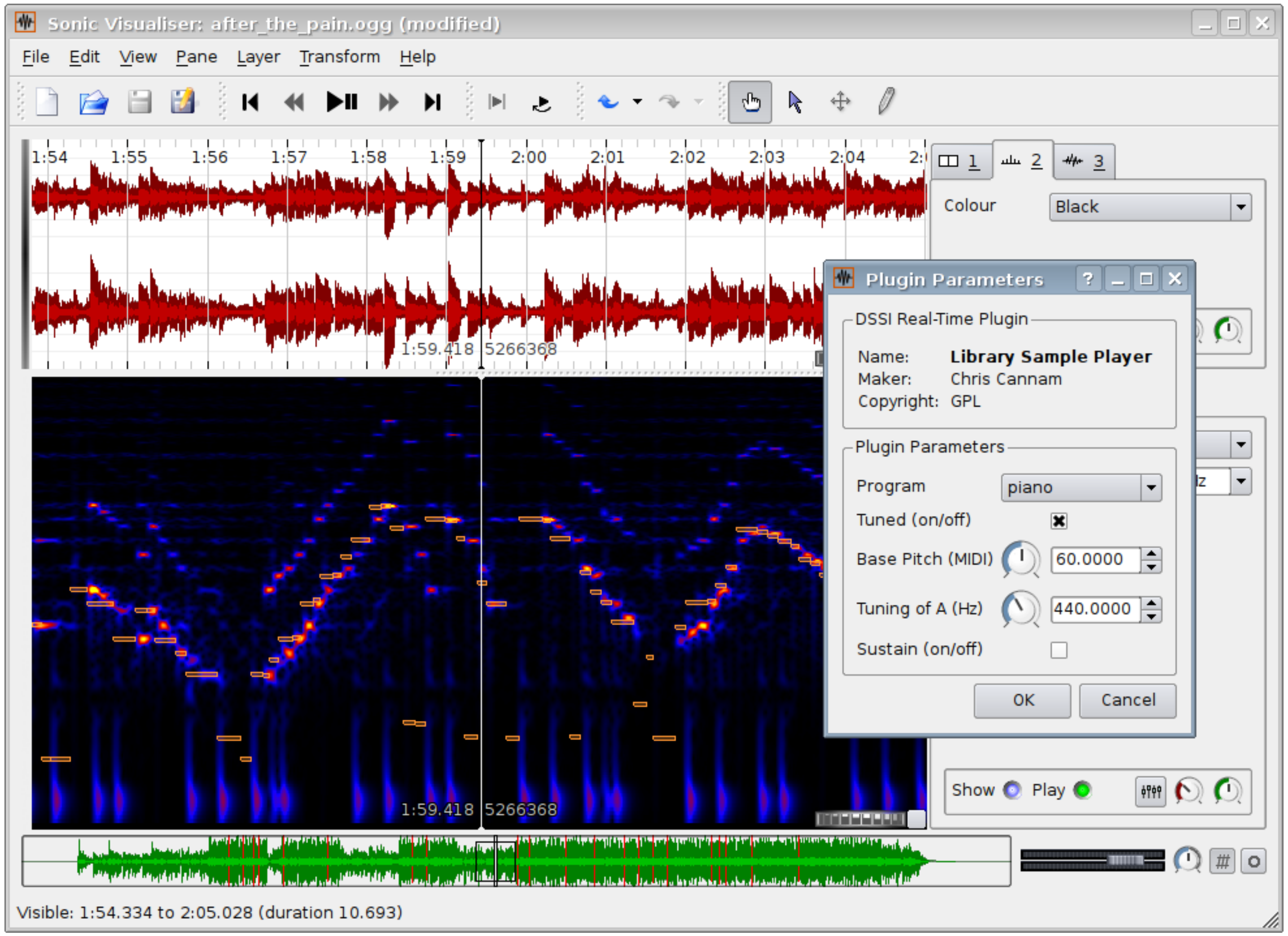
- Demonstration of techniques to extract temporal information from music signals

- Look at analysis in a bottom-up fashion

- Then apply music description for music manipulation and interaction

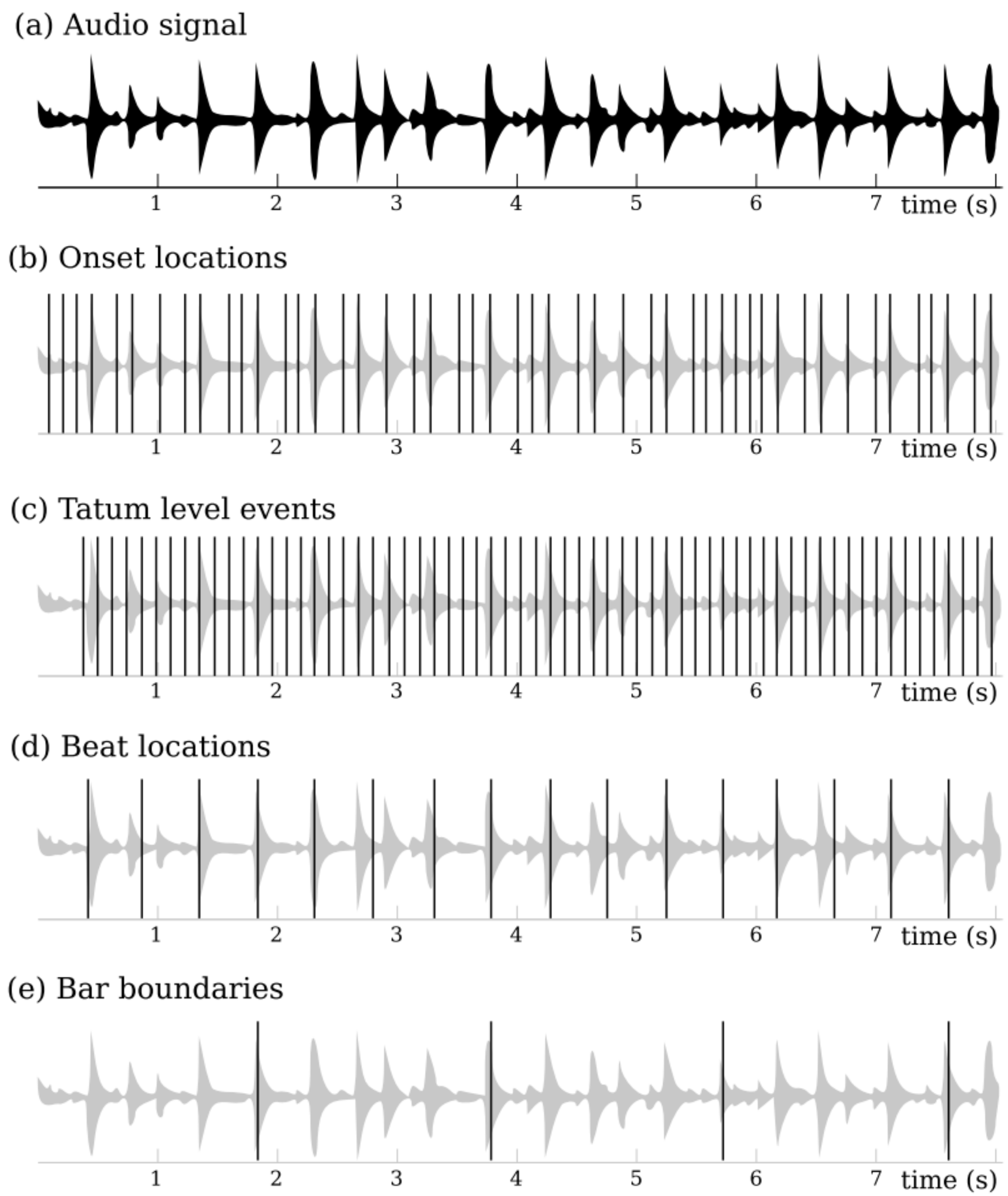
Part I
MIR and Rhythm

Sonic Visualiser



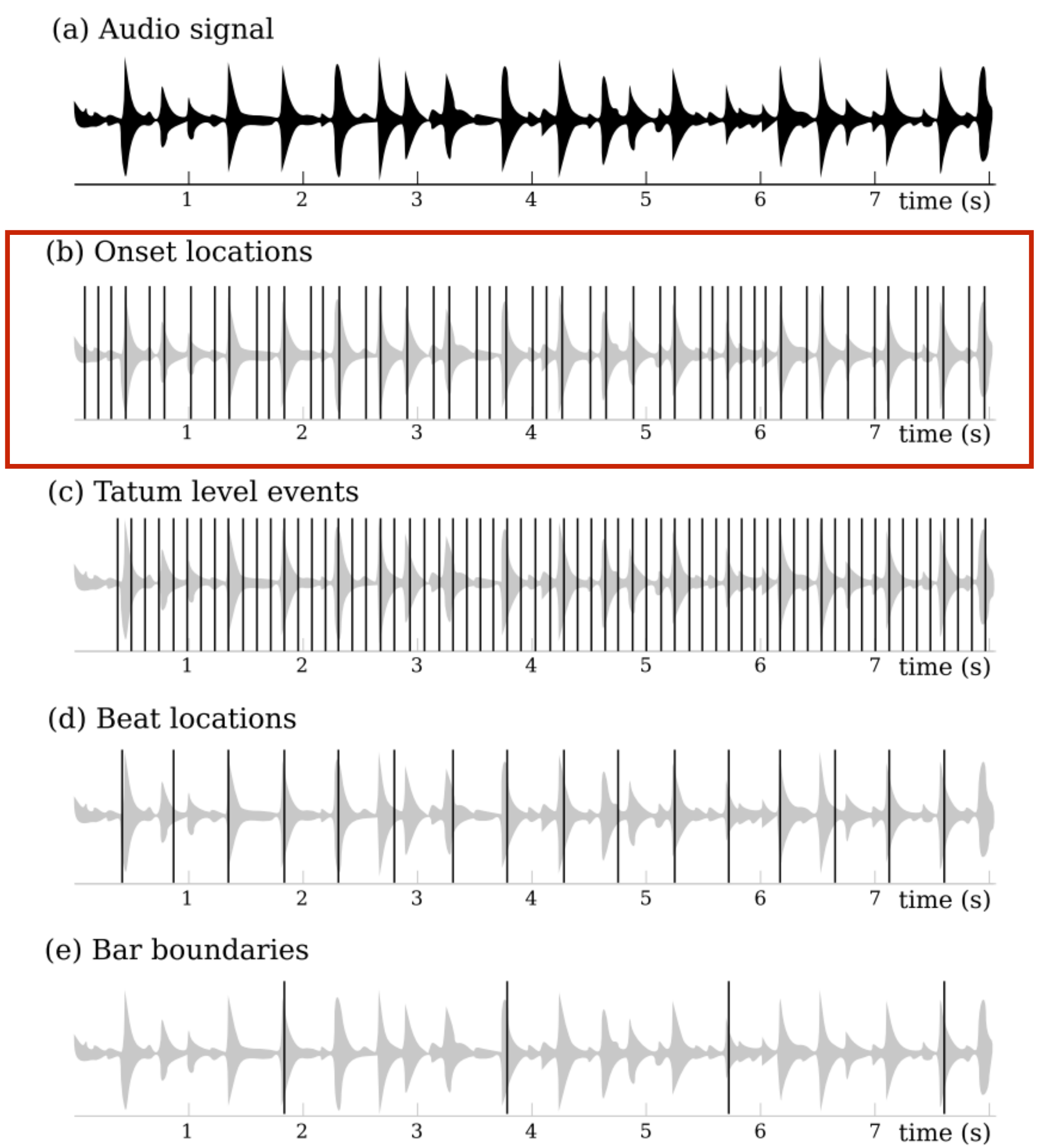
- Free open source analysis and visualisation tool for musical audio - www.sonicvisualiser.org

Musical context



- onsets - start times of note events
- tatum - fastest level
- beat - comfortable tapping level
- downbeats - grouping beats into bars

Musical overview

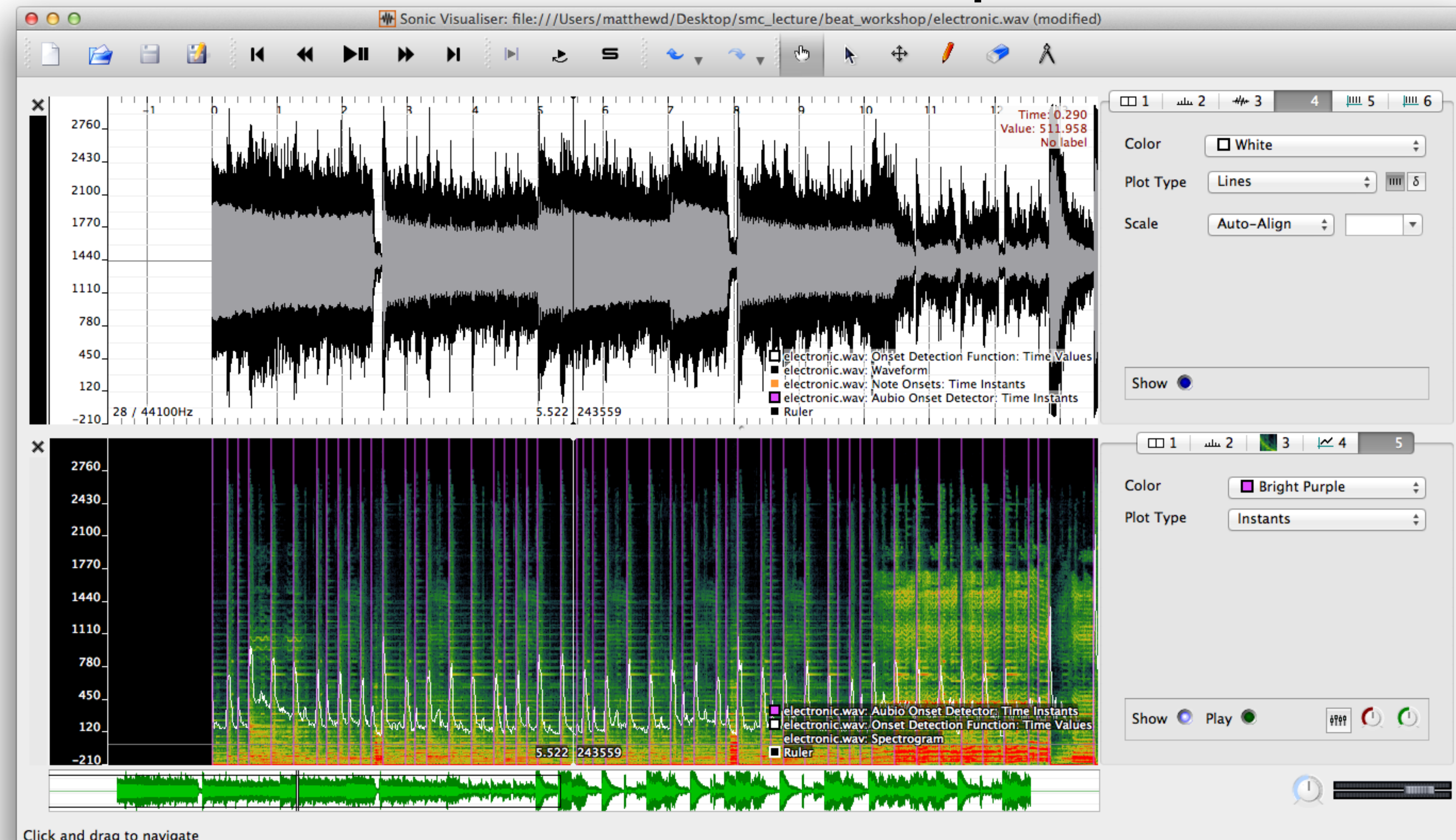


- **onsets** - start times of note events
- tatum - fastest level
- beat - comfortable tapping level
- downbeats - grouping beats into bars

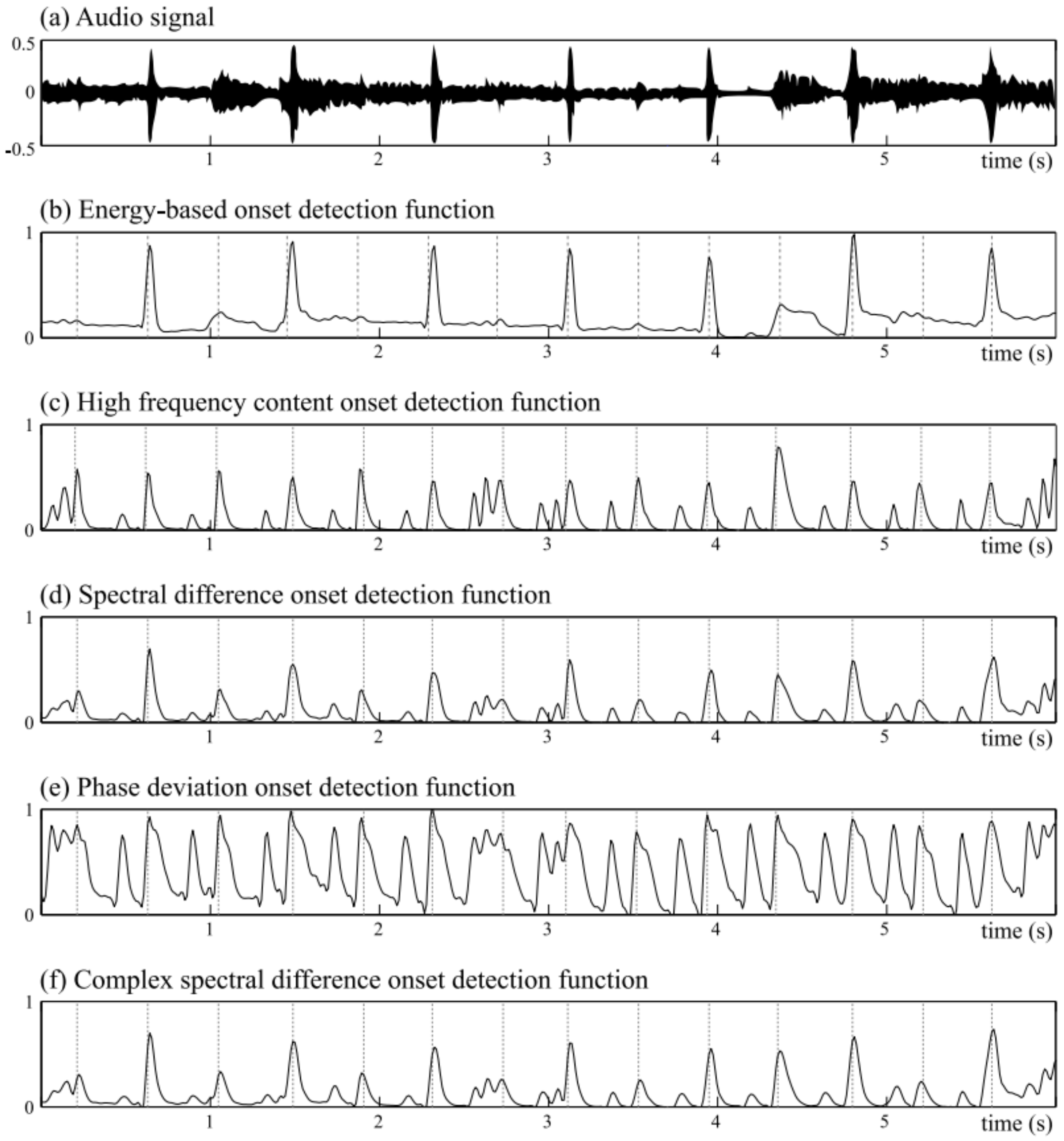
Basic approach: onset detection

- Go from audio domain to time-frequency domain
  - Use the **Short-Time Fourier Transform** (STFT)
- Make an **onset detection function** (ODF)
  - Measure difference in STFT across small time frames (e.g. 10-20ms)
- To obtain **onset locations**, “peak-pick” the ODF

Onset Example



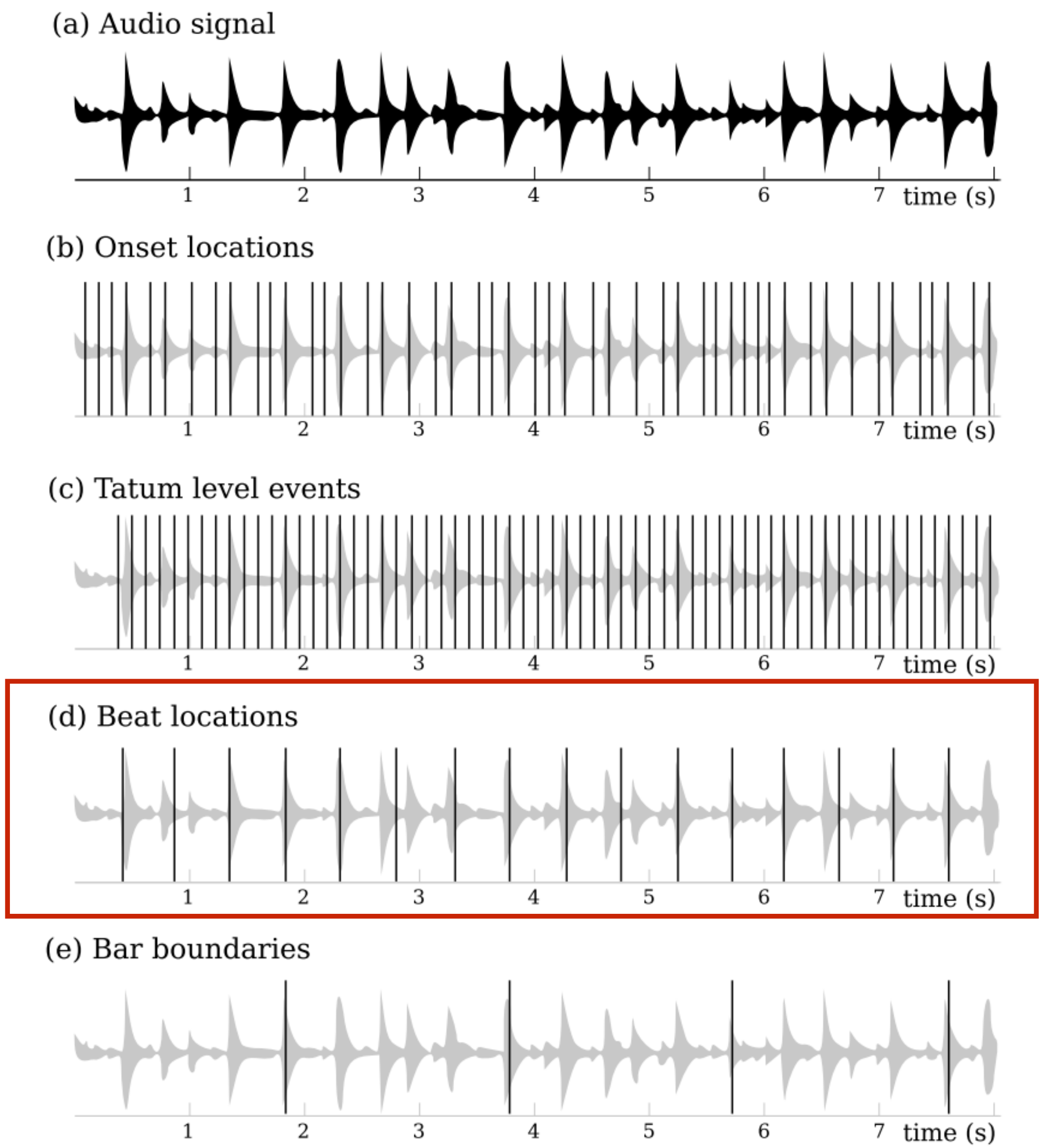
Many different onset detection functions



- We can make lots of different onset detection functions

- Using: energy, phase, spectral difference, emphasis on high frequencies

Musical overview



- onsets - start times of note events
- tatum - fastest level
- **beats** - comfortable tapping level
- downbeats - grouping beats into bars

What is beat tracking?

- Beat tracking is the computational task of getting a computer to “**tap it's foot**” in time to music

- The aim is to reflect the innate human ability to induce and follow a pulse in music

  - We often to do this without thinking - it's easy, right?

  - But how do we make the computer “**feel the beat**”?

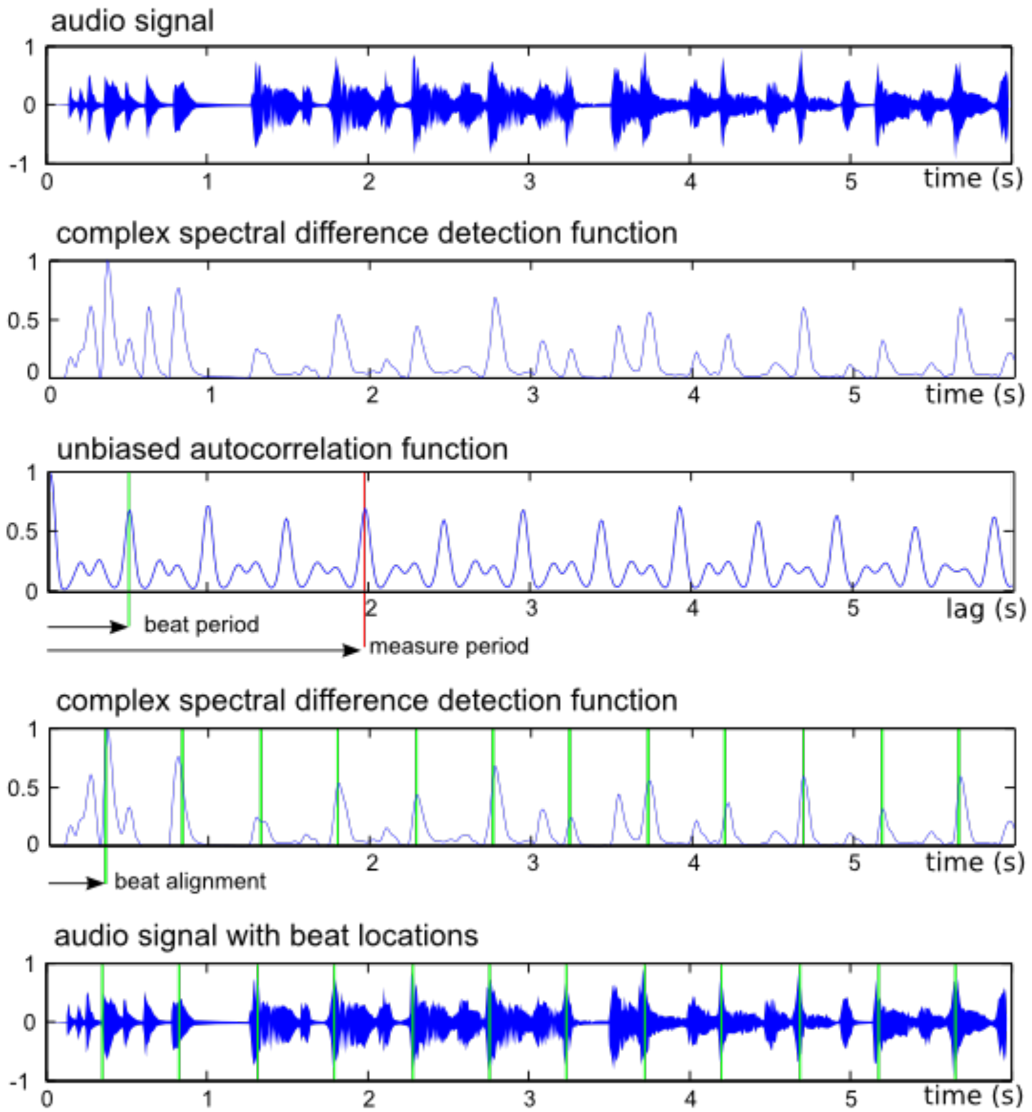
How is beat tracking done?

- In **lots** of ways!
  - There are probably over 100 “different” beat tracking algorithms out there
  - Using: comb filters, autocorrelation, neural networks, psychoacoustic models, dynamic programming, particle filters, bayesian models, etc, etc.

Basic approach

- Calculate an **onset detection function**
  - emphasises locations of start times of events
- Estimate **tempo** by some periodicity analysis
- Determine the **phase** of the beats given the periodicity

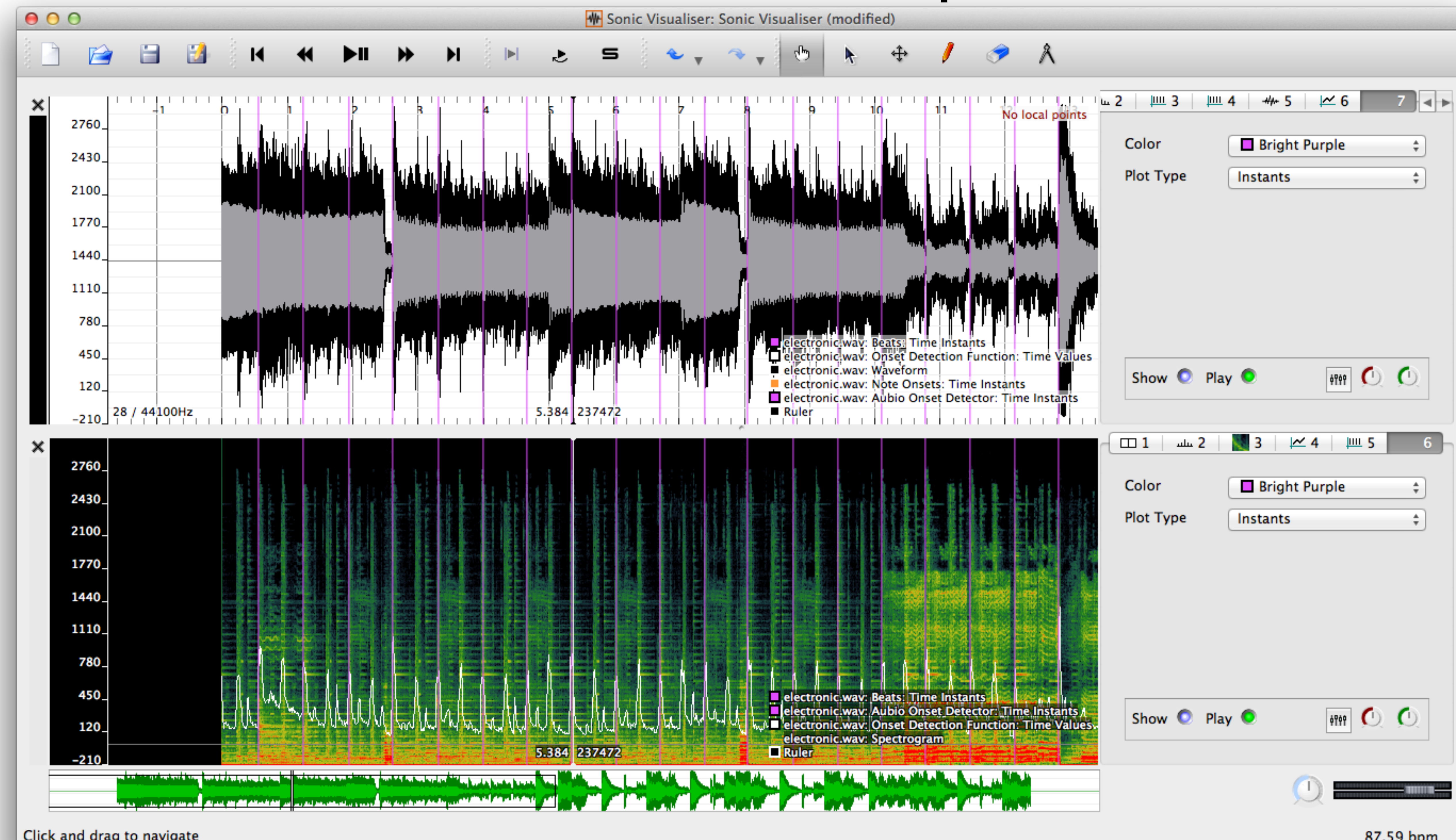
Graphical example



- Transform audio to onset detection function (ODF)
- Look for strongest beat periodicity
- Find periodic peaks in ODF
- Playback beats with audio

- Let's also look in Sonic Visualiser

Beat Example



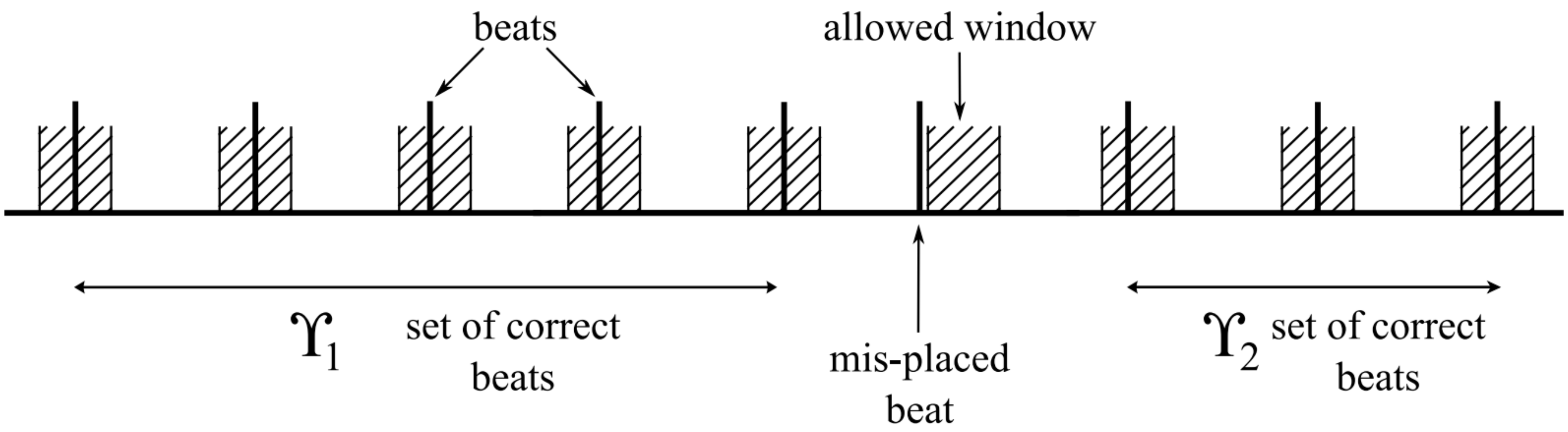
How is beat tracking used?

- What are the main applications of beat trackers?
  - In many music information retrieval (MIR) research tasks: chord detection, structural segmentation, finding cover-songs, music transcription - **analysis in musical time**
  - And in creative/performance applications: beat-synchronous audio effects, automatic accompaniment, automatic remixing, mashups - **synchronisation is very important**

How do we evaluate beat trackers?

- **Subjectively**
  - By listening back to the beats mixed with the original music signal
    - what's good? what's bad?
- **Objectively**
  - By marking up “ground truth” and comparing to beat estimates
    - what's good? what's bad?

Objective evaluation



- make **tolerance windows** around **ground truth**
- count number of correct beats (w/ continuity)
- allow different metrical interpretations (e.g. double/half tempo)

How good are beat trackers?

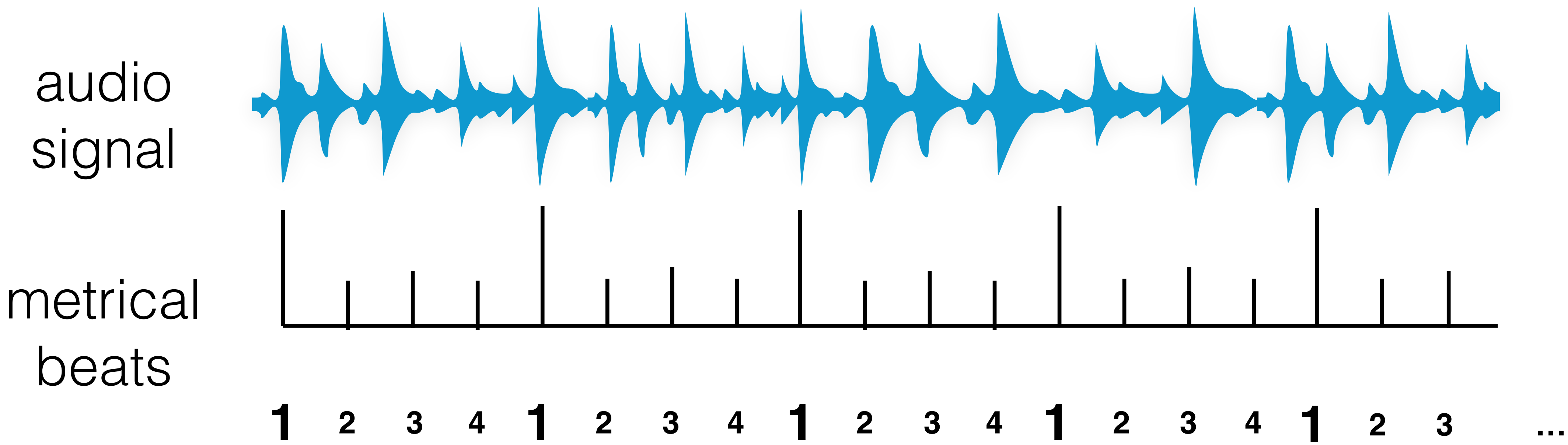
- The state of the art is very good for “**easy**” types of music:
  - rock/pop, (some) electronic dance music -> steady tempo with strong percussive content
- It's not so good for jazz, folk, classical
  - tempo variations, no drums, changes in metre

Part II
Basic Creative Transformations

Towards creative use of mir and rhythm

- Let's explore some simple transformations
  - First, do MIR analysis to extract beats and downbeats
  - Then, undertake transformations according to this information
    - i.e. only using knowledge of rhythm + metre
    - knowing about the beat is critical for synchronisation between different pieces of music

Beat and downbeat tracking

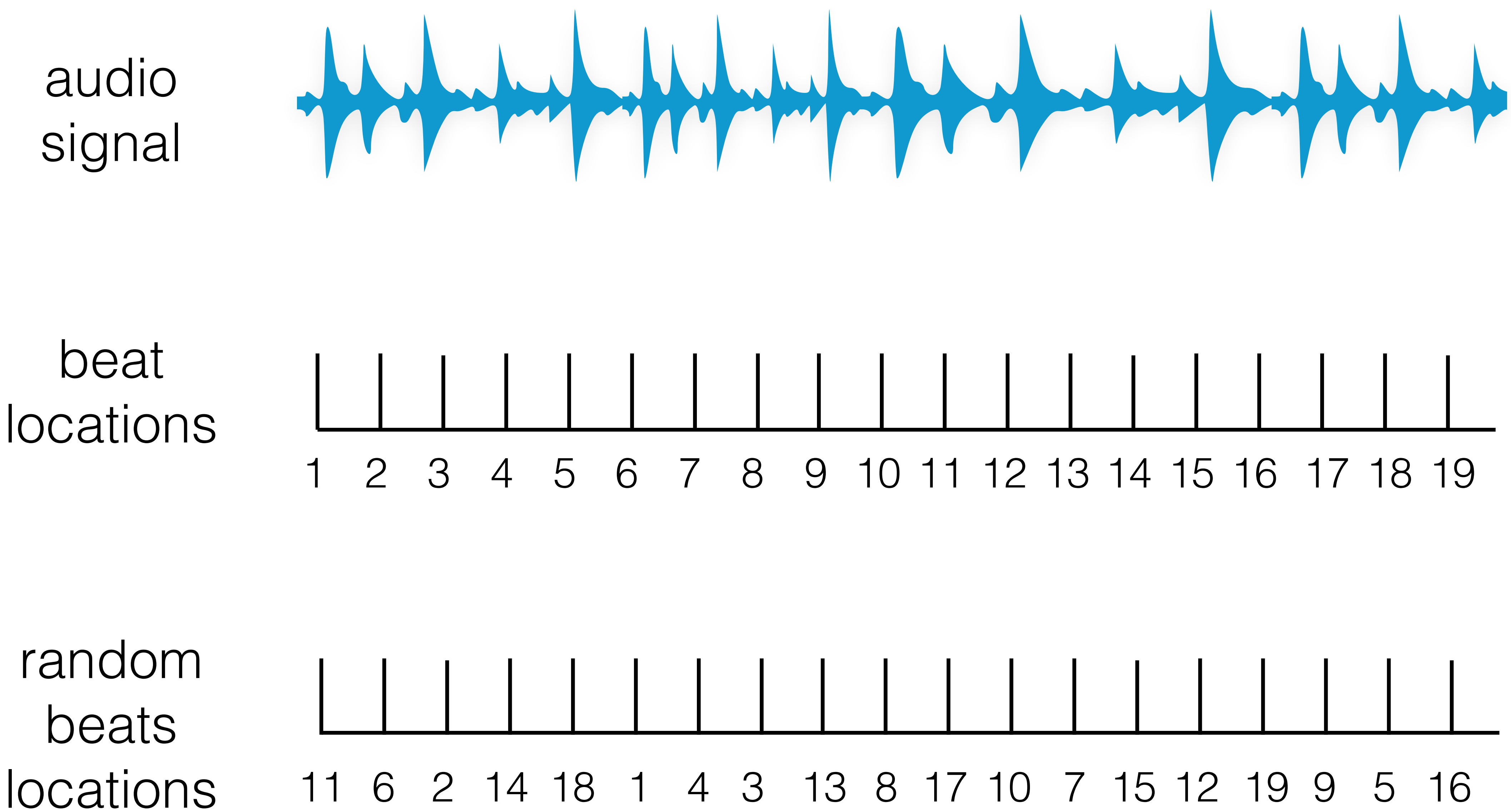


- The "metrical beats" are the main representation we'll work with for these simply transformations

Beat Tracking

- We'll explore:
  - scrambling the music in two different ways
  - changing the rhythmic structure
  - some automatic remixing

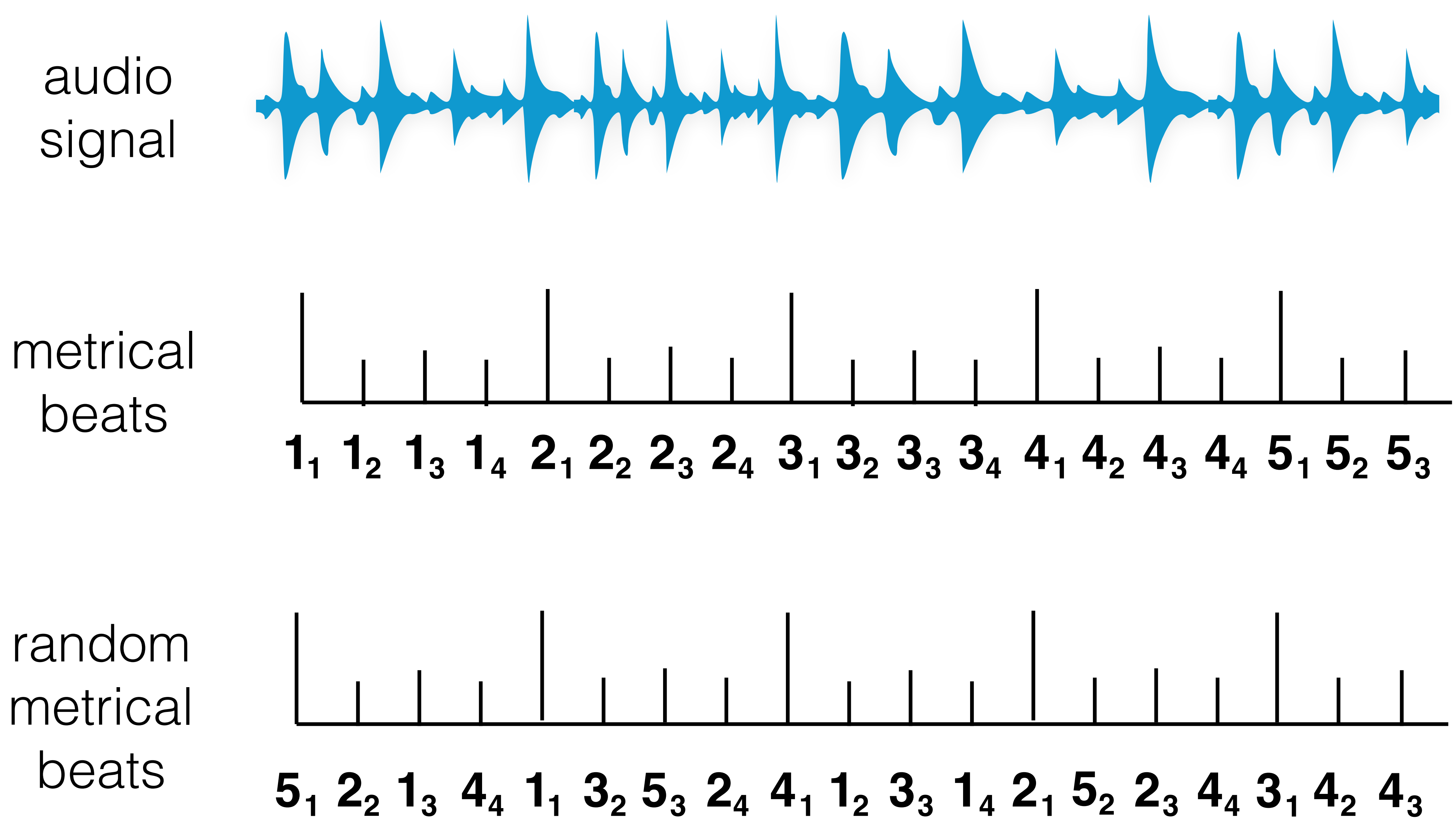
Beat Randomizer



Beat Randomizer with Metre

- Let's see if we can make the result of the "beat randomizer" a bit more musical
- Instead of picking a **totally** random beat each time, we will try to preserve the **metrical structure** by using **downbeat** information
  - This means we pick a random beat by chose the metrical positions in order
  - So, we pick a random first beat of each bar, then a random second beat, etc.
- In this way the result is somehow *less* random

Beat Randomizer with Metre



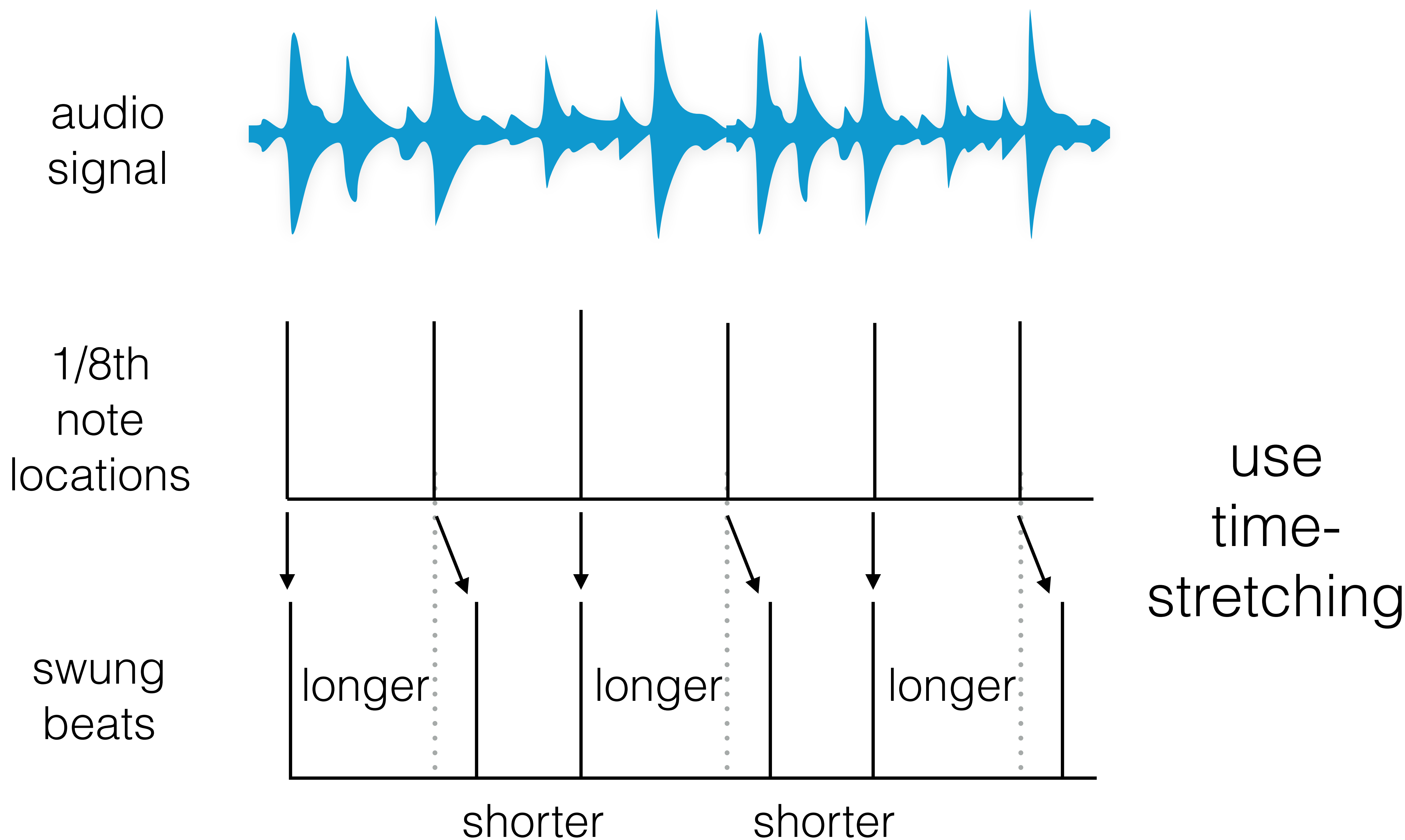
Beat Randomizer with Metre

- We can vary number of sub-divisions per beat,
- e.g. 2, 4 or 8

Beat Swinger

- Given the beat times we can use a **time-stretching** tool to alter the rhythmic structure of the music
  - Time-stretching alters the speed of the music without changing the pitch
  - To add **swing-feel** we modify the duration of 1/8th notes (i.e. half-beats) in an alternating *long-short* pattern
  - e.g. we make the first 1/8th note 30% longer and the second one 30% shorter

Beat Swinger



Beat Swinger

- To experiment more, we can modify how the beats are sub-divided, and change the stretch factor, e.g.
- Best results are obtained through experimentation, and finding the right metrical level to *swing* (normally the fastest)
- But, we could do really weird things too
    - reverse-swing feel at the beat level rather than the 1/8th note level

Beat Swinger

- The beat swinger transformation is different from the beat randomizer in two ways
    - It's much slower to run :(
    - Audio quality is worse :(
- Time-stretching is not perfect and can create some artefacts

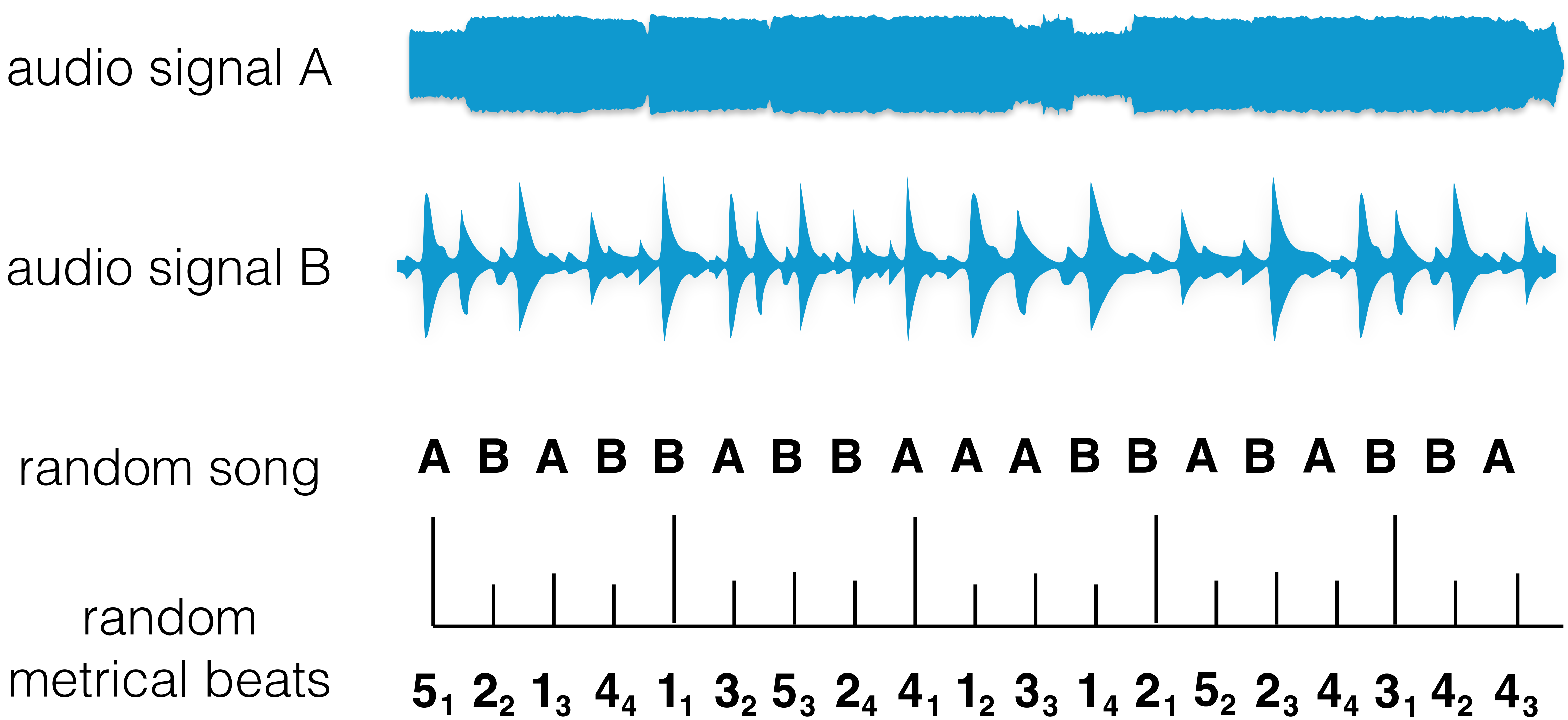
Beat remixer

- If we can modify the timing of beats in music - as shown with the beat swinger, we can apply the same principle for mixing songs together
  - i.e. we can **beat-match** to make two songs have the same tempo and be synchronised for playback
- Instead of a straight beat-matching application, we'll do something a bit different, built around the beat randomizer as well

Beat remixer

- Given two input music files, we'll follow the "Beat Randomizer with Metre" process for each of them
- But then, also randomly **switch** between which song we playback at each beat
- Making sure that we've **time-stretched** each the beats to be at the same tempo

Beat Remixer



Let's try to make an actual remix

- The musicality of the results can be improved a lot by imposing some **higher level structure**, and **repeating** the sections we create

- So, let's make multiple short mixes and glue them together to make some kind of **structured remix**

- As with the other functions we can provide some additional input parameters to shape the result, e.g.

    - the **sub-beat** level for randomizing beats

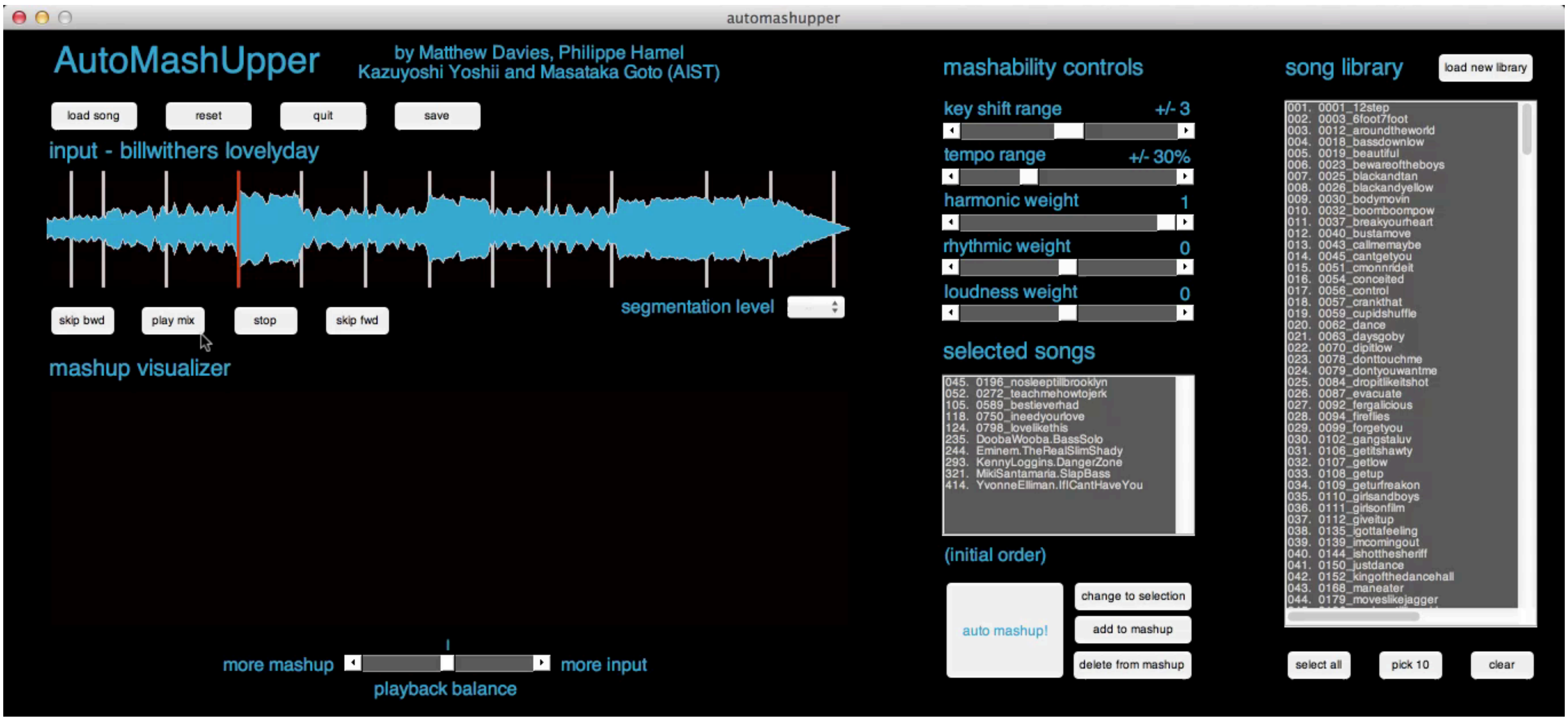
    - the **probability** of choosing one song over another

Beat remixer perspectives

- It's not really a complete automatic remixing system
- But there's nice scope to vary the input files and parameters as well as the higher level pattern structure
- It just takes a little experimentation
- It would be much better to make something which makes a more informed decision about how to mix songs -> **music mashups**

Part IV
Music Mashups

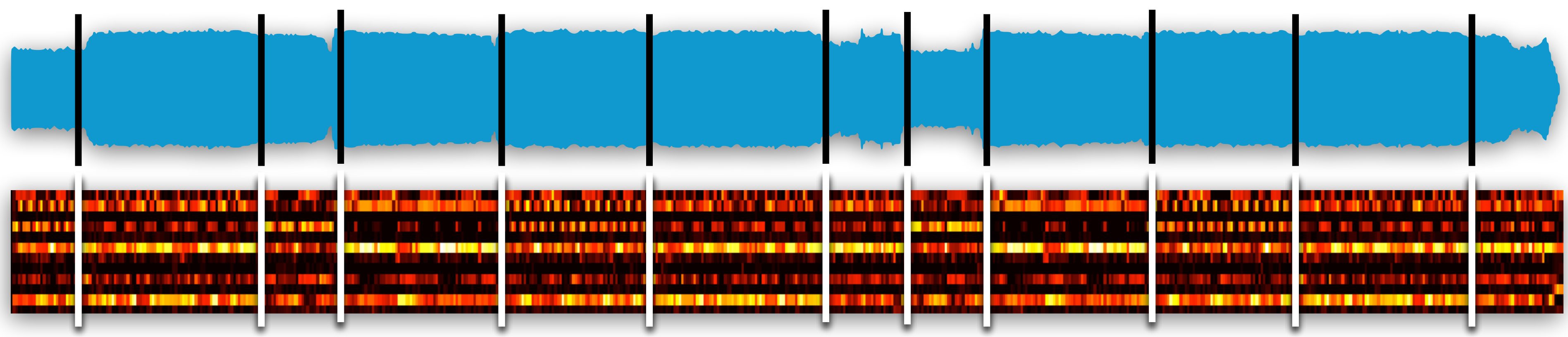
AutoMashUpper



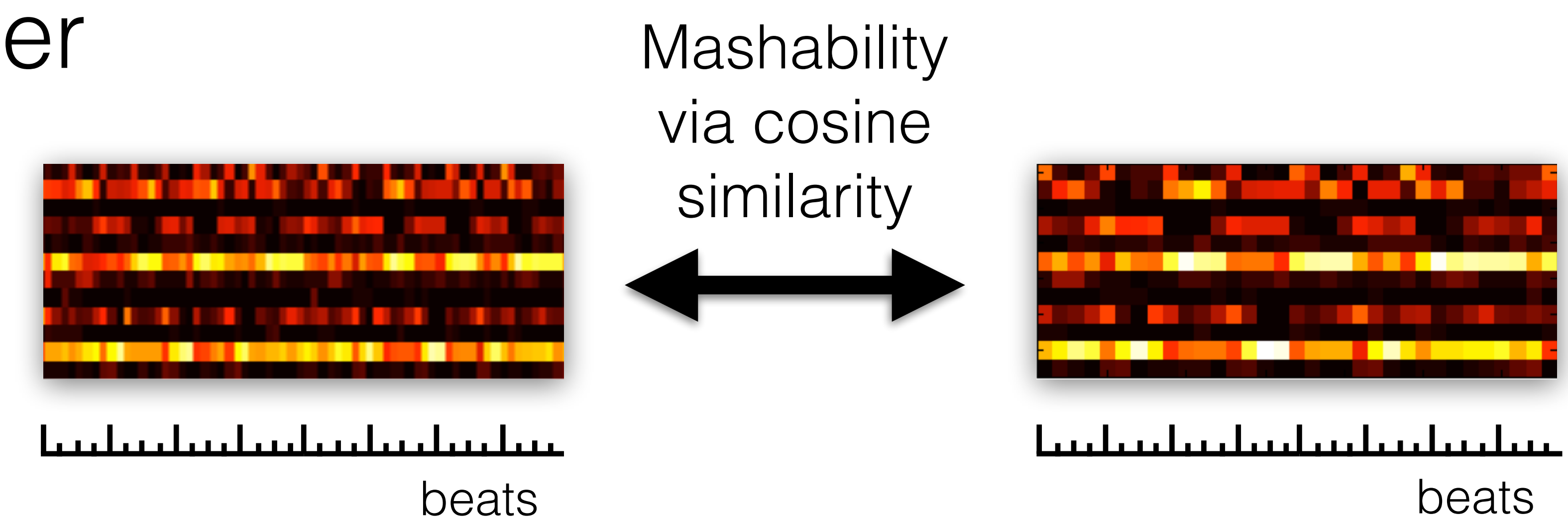
*Developed while at AIST, Japan in 2013

System Components

- **Music signal analysis** - phrase segmentation and mashup signal representation

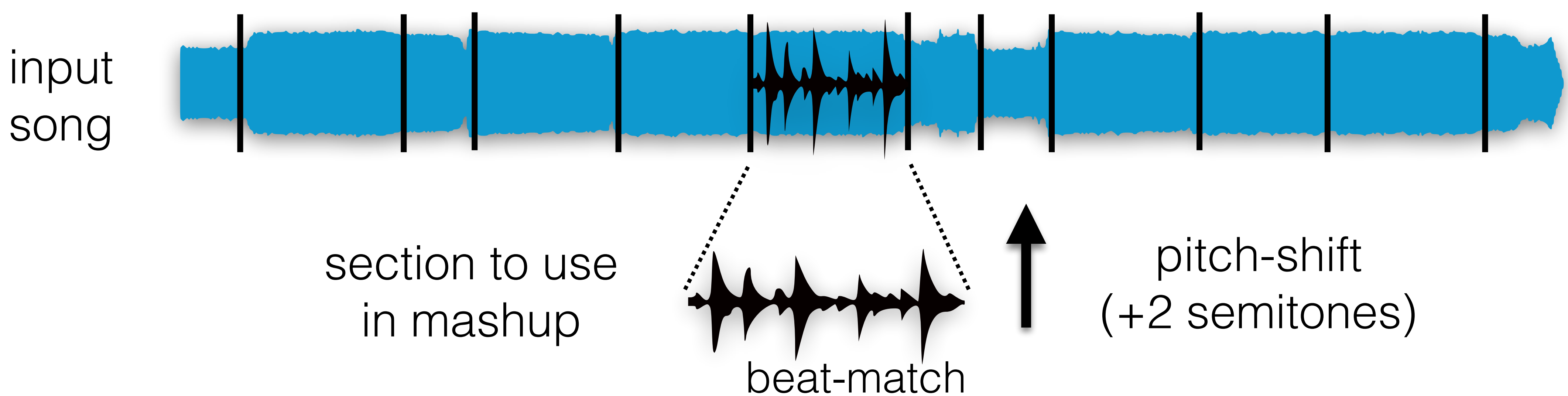


- **“Mashability”** - measure how well two sections fit together

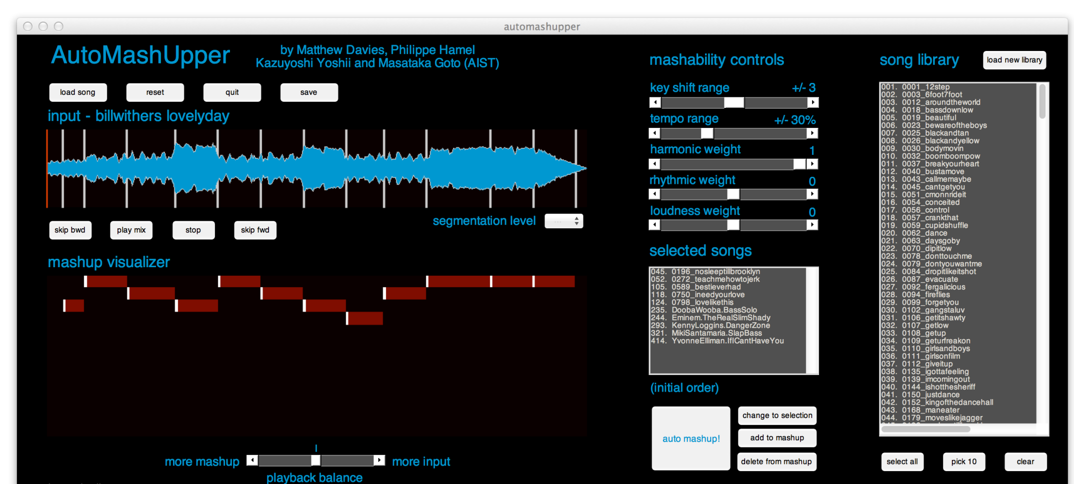


System Components

- **Mashup creation** - use time stretching and pitch shifting to create the musical result

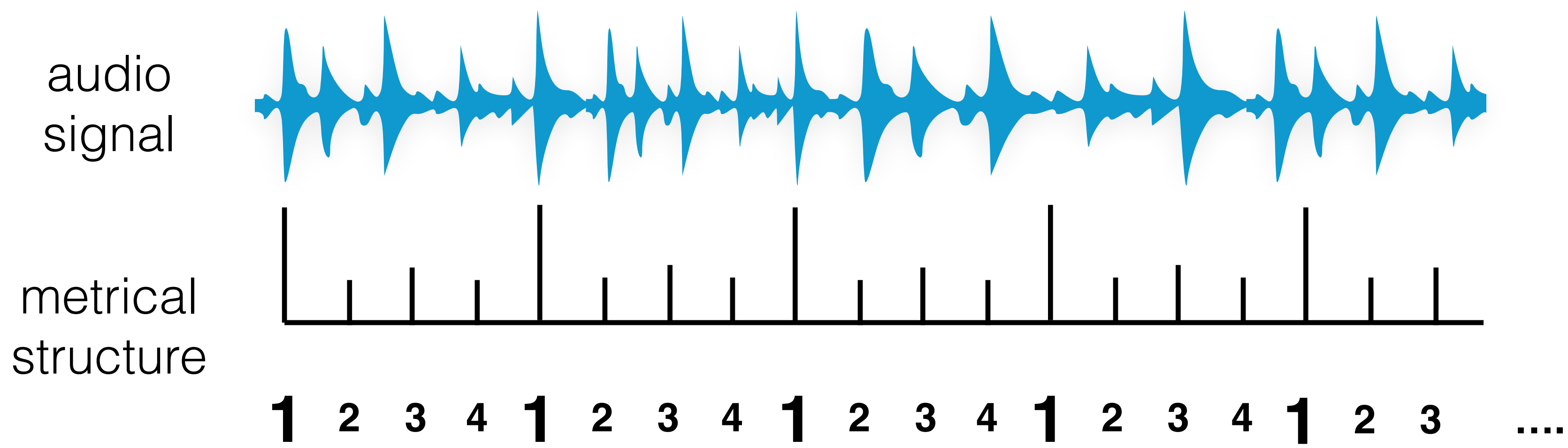


- **User Interface**
  allow users to interact with mashup creation process and manipulate result



Beat tracking

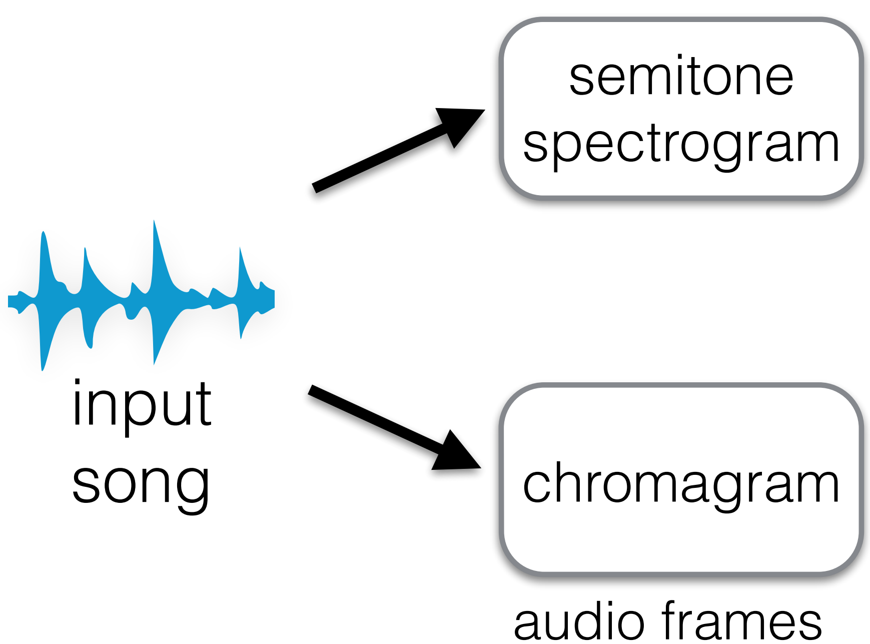
- **Critical** first step in mashup system
  - Enables temporal **synchronisation** between songs
  - Underpins **all** subsequent analysis



  - Estimate beat and downbeat locations
    - **assume** approx. constant tempo and 4/4 time signature

Beat-synchronous harmonic representations

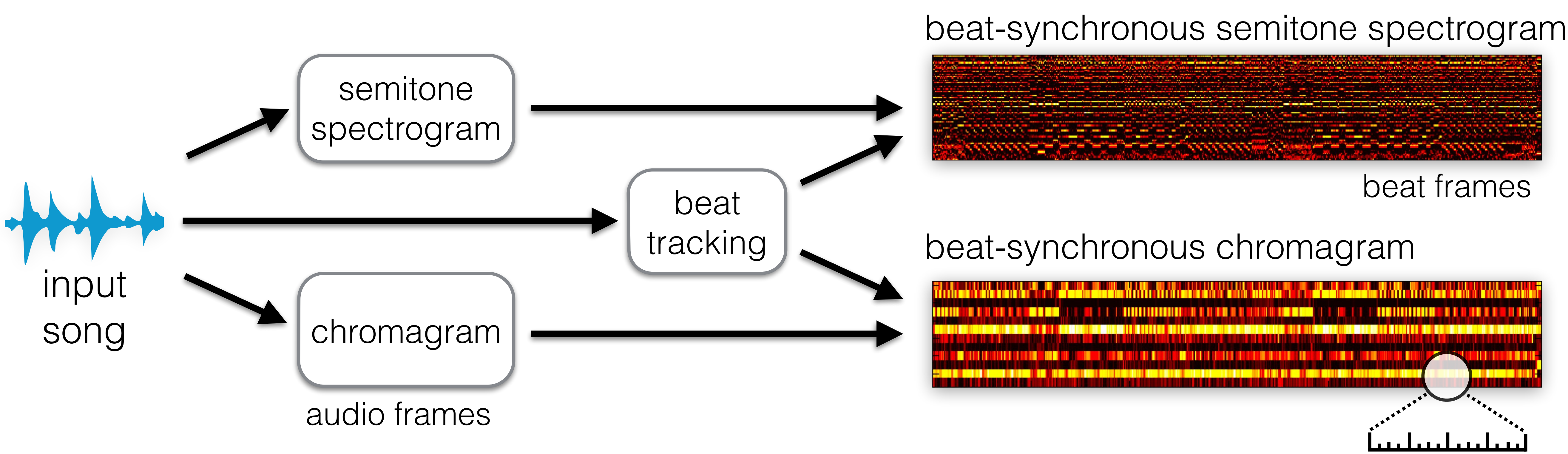
- AutoMashUpper uses **two** harmonic signal representations
  - **semitone spectrogram** and **chromagram**
- Use beat information to make **beat-synchronous** versions



- Use NNLS chroma [Mauch, 2010] to extract chromagram and semitone spectrogram

Beat-synchronous harmonic representations

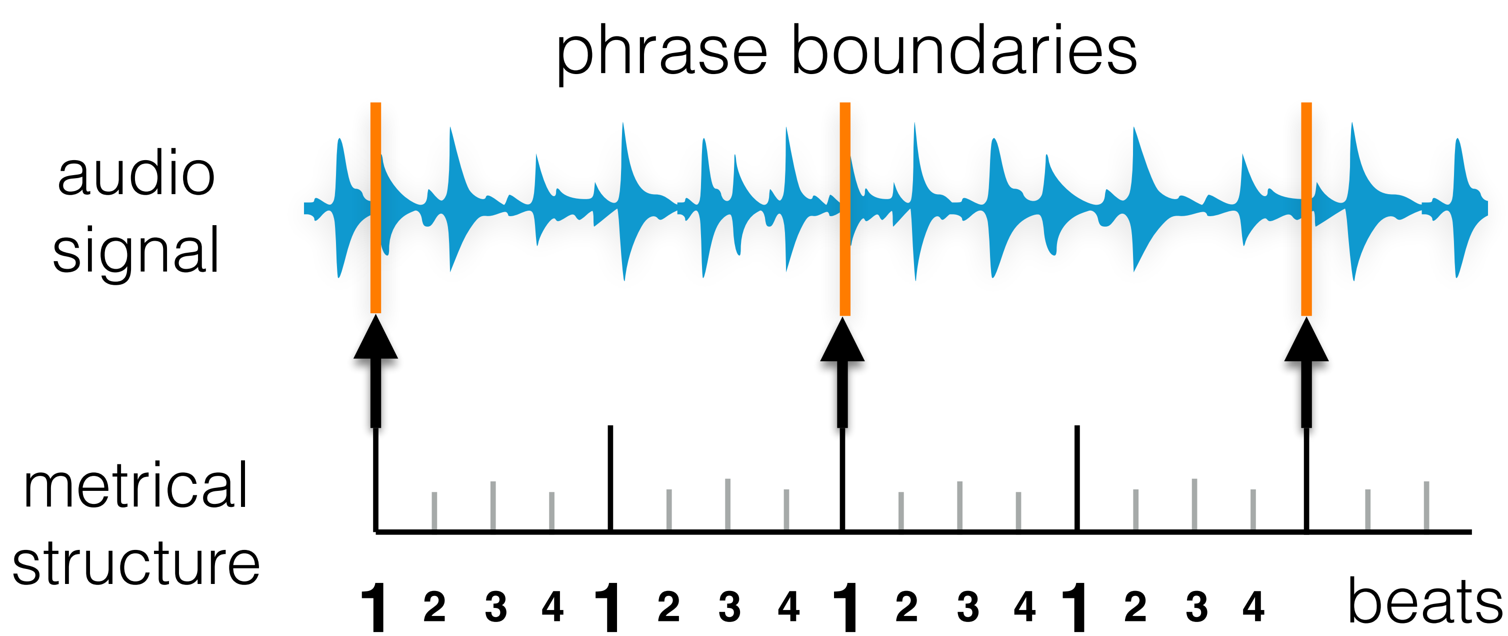
- AutoMashUpper uses **two** harmonic signal representations
  - **semitone spectrogram** and **chromagram**
- Use beat information to make **beat-synchronous** versions



- Use NNLS chroma [Mauch, 2010] to extract chromagram and semitone spectrogram

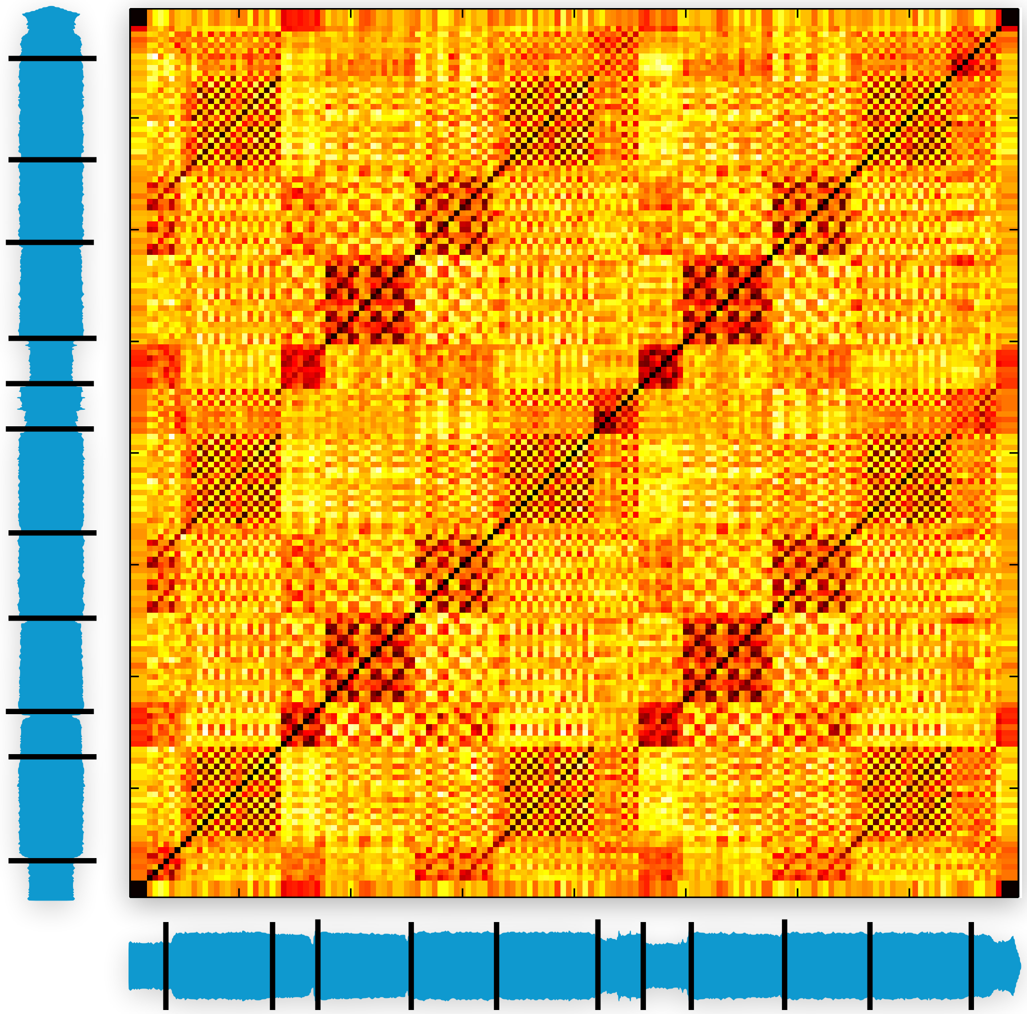
Phrase level segmentation

- Extract **precise** temporal boundaries between phrases
  - **Assume** phrase sections change on downbeats
  - Expected phrase length ~ 8, 16, 32 beats



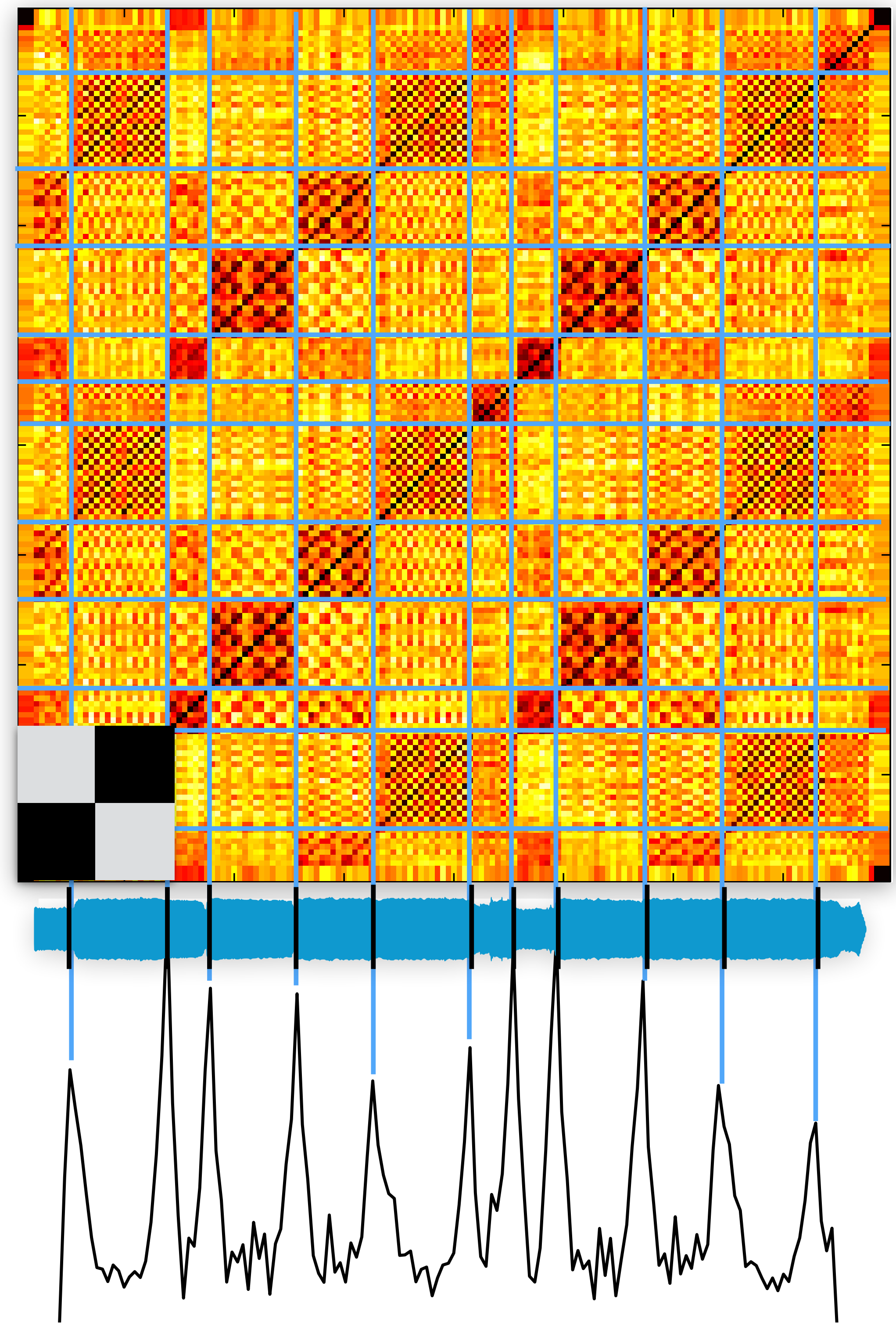
Phrase level segmentation

- To find phrase boundaries, we modify Foote's classical approach for structural segmentation based on the **self-similarity matrix**
- It is calculated by measuring the distance between features in **every time frame** to **every other time frame**
- We're trying to find the boundaries between the squares along the main diagonal of the self-similarity matrix



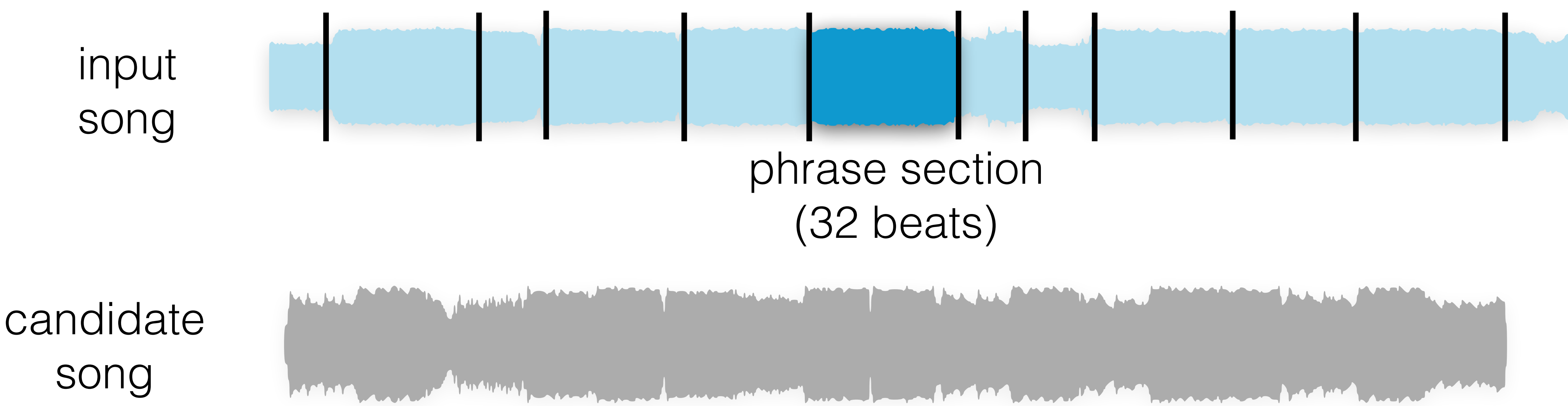
Phrase level segmentation

- Calculate a **checkerboard** kernel
- Slide kernel along main **diagonal** of self-similarity matrix
- Calculate novelty function to capture **block changes**
- **Peaks** of novelty function give section **boundaries**



Mashability

- For each phrase section of input song,
  search for the "best match" in the user's song library
- Estimate **mashability** between songs



where is best
matching 32 beat
region in the
candidate song?

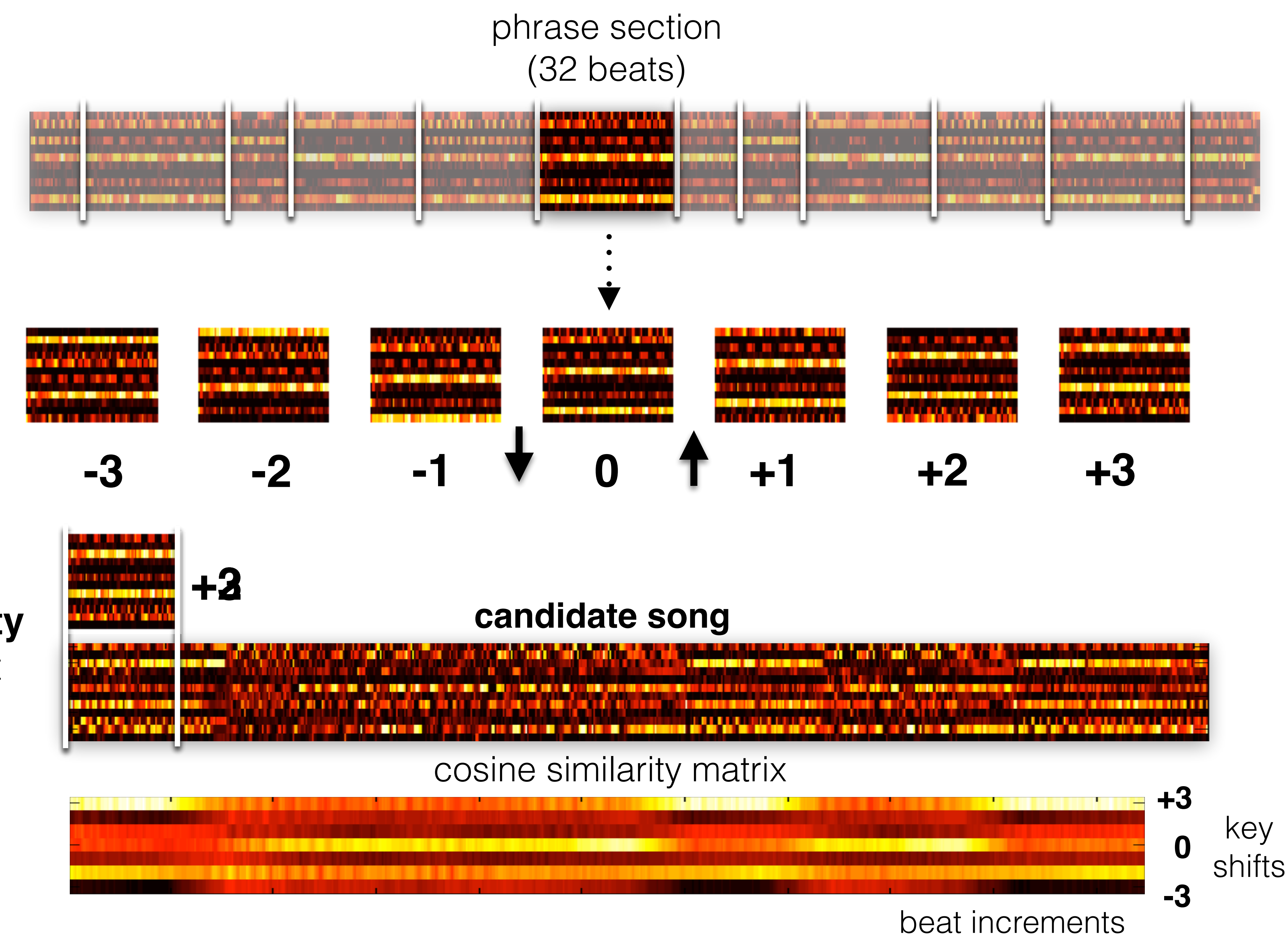
how could the
candidate song be
made to match the
input phrase section?

Mashability estimation

input song
beat-
synchronous
chromagram

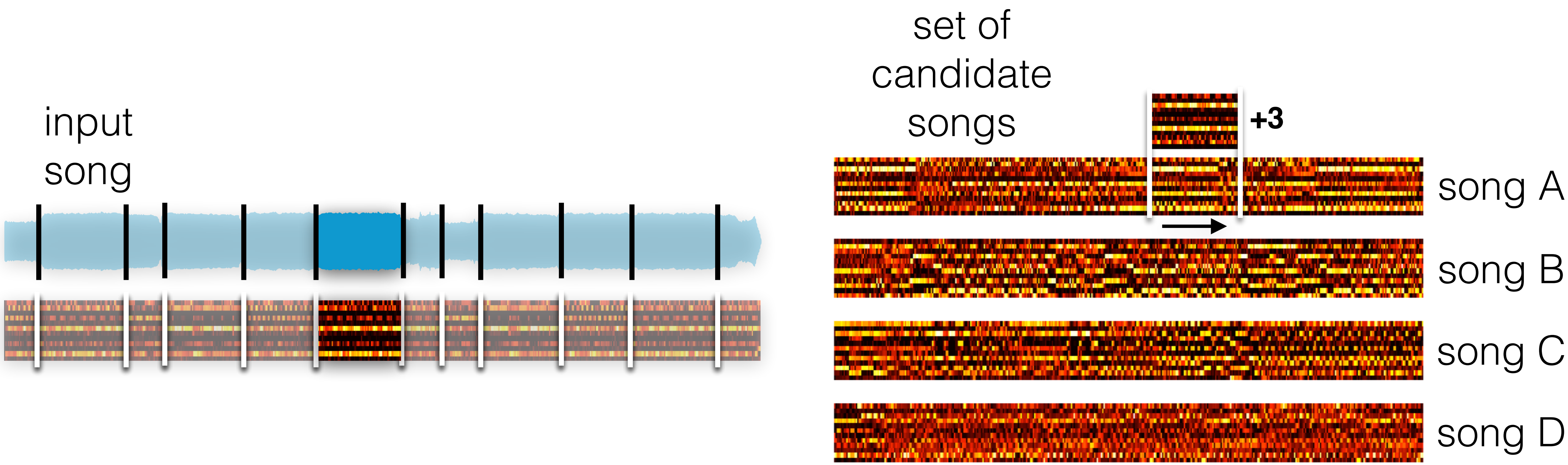
rotate **input**
chroma
to test
multiple
key shifts

measure
cosine similarity
across **all** beat
increments
of candidate
song for
all key shifts



AIST

Mashability estimation

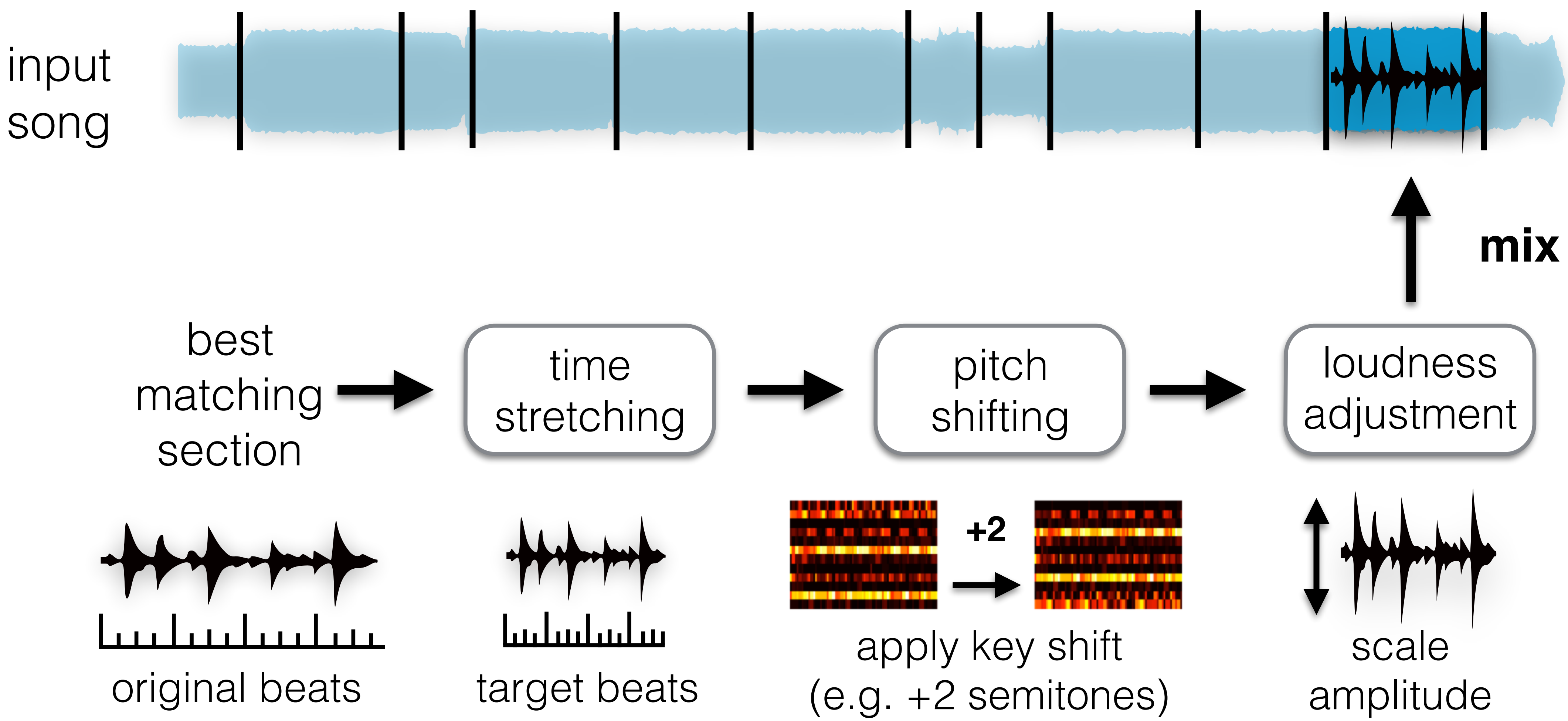


- Repeat mashability estimation across all candidate songs and **rank** maximum mashability score per song
- Search space is **huge**
  - e.g. **7** key shifts x **500** beats = **3500** possible mashups for **one** phrase section matched to **one** song, but computation is acceptable (for up to 500 songs)

Mashup creation

- For each phrase section, the ranked mashability tells us:
  - **which** song to use in the mashup
  - **when** in the song (starting beat)
  - **how** to transform it to make the match (key shift)
- The final step is to create the mashup

Mashup creation

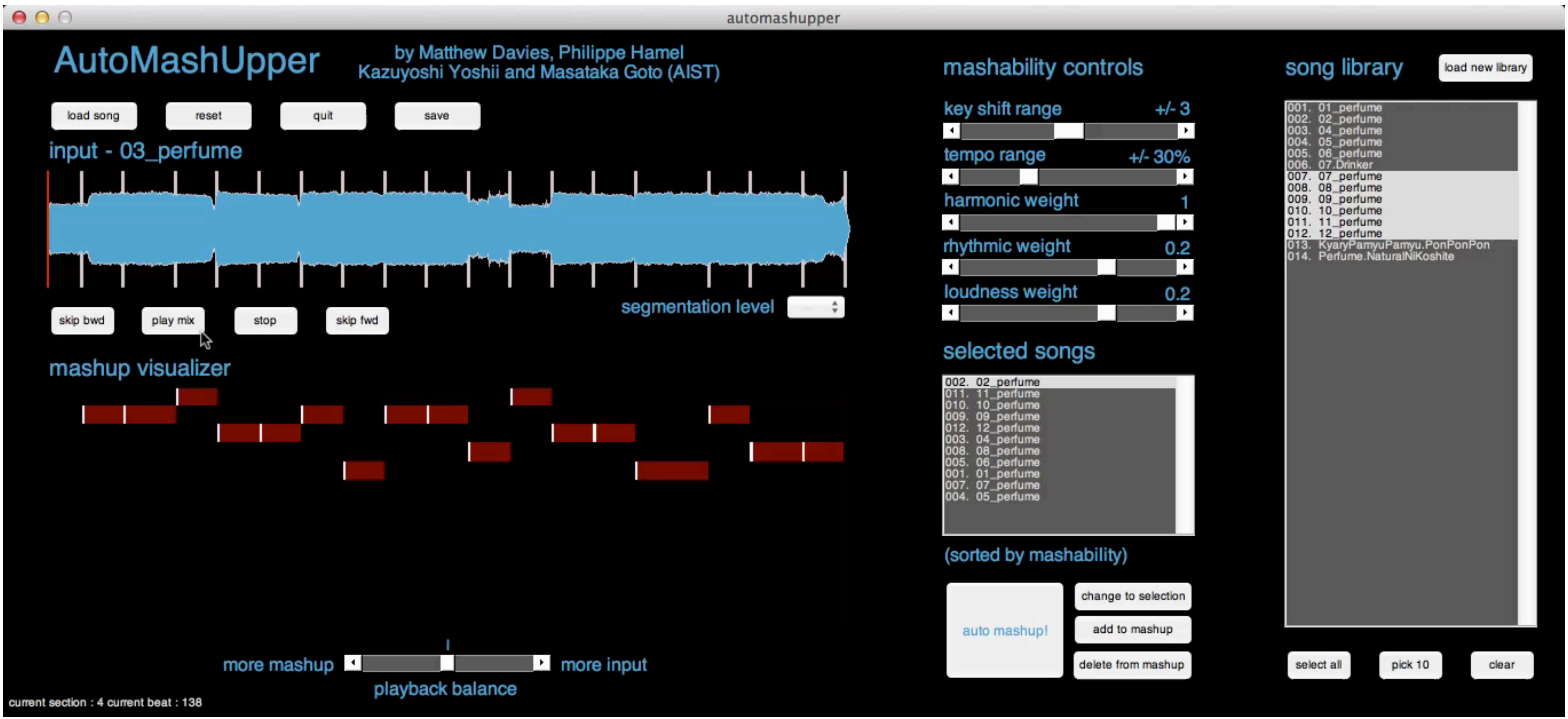


AIST

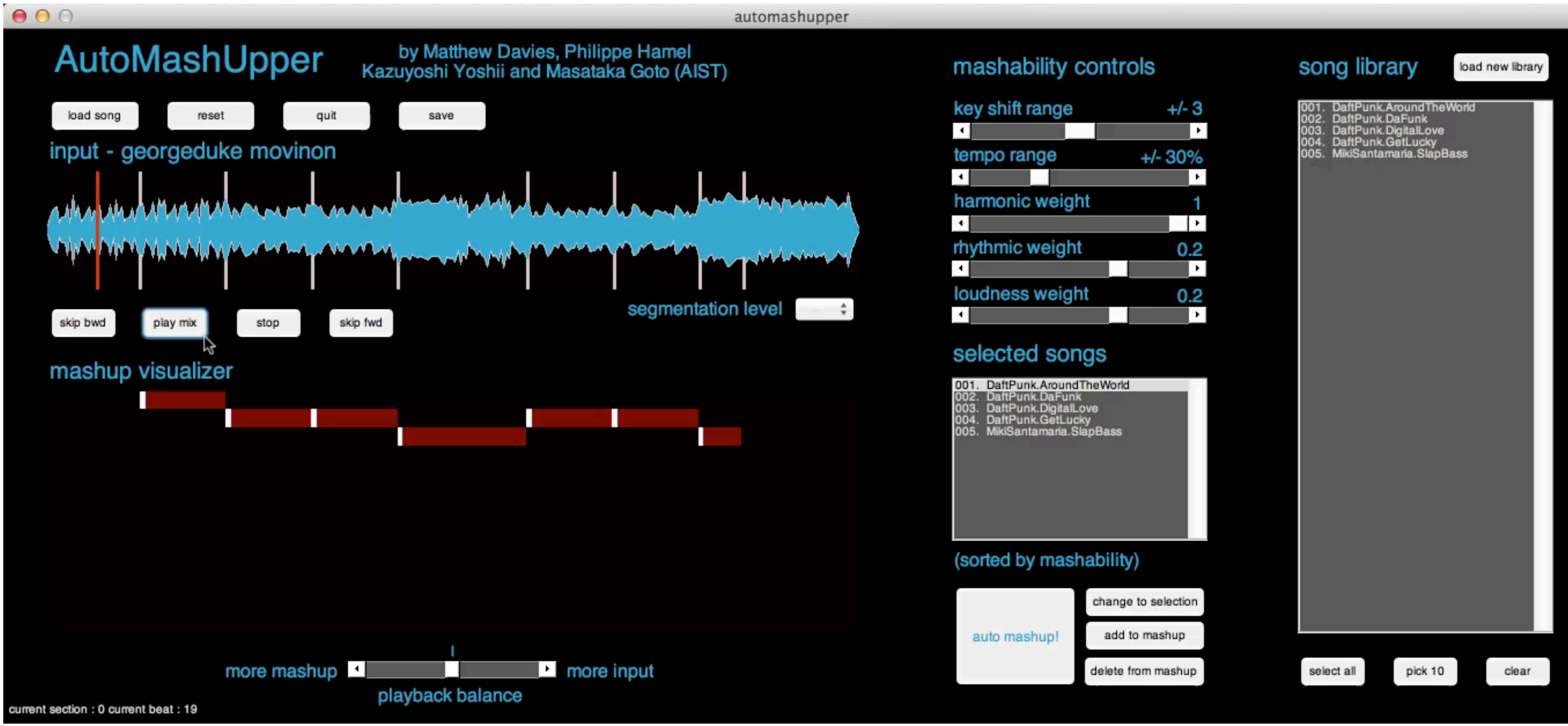
Modes of operation

- AutoMashUpper can be customised to give different musical results by changing the candidate songs:
  - **Standard mode**: full song library or pick ten random
  - **Artist/album mode**: only use songs from same album
  - **Auto-slap bass**: mashup input with solo slap bass

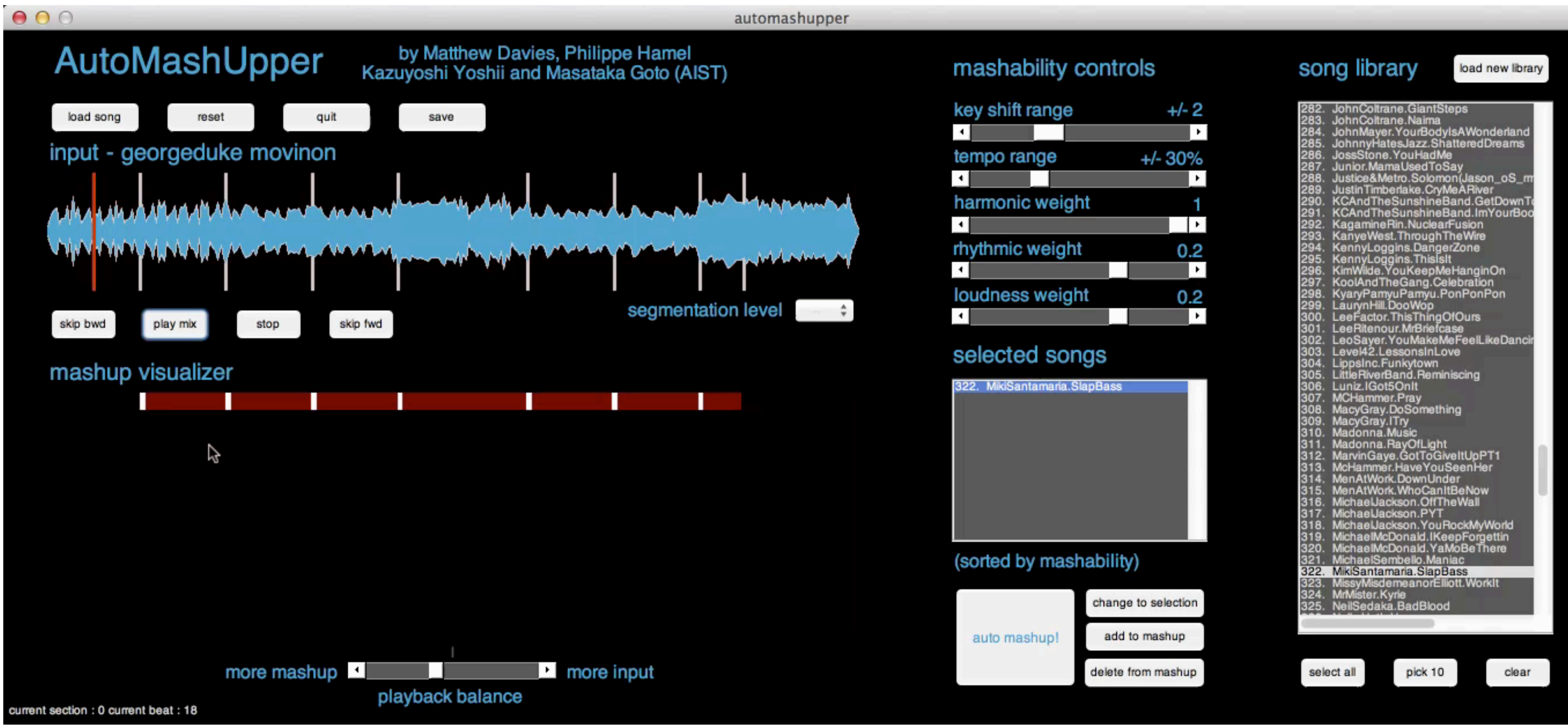
Album Mode


automashupper
AutoMashUpper
by Matthew Davies, Philippe Hamel
Kazuyoshi Yoshii and Masataka Goto (AIST)
load song
reset
quit
save
input - 03_perfume
skip bwd
play mix
stop
skip fwd
segmentation level
mashup visualizer
more mashup
more input
playback balance
current section : 4 current beat : 138
mashability controls
key shift range
+/- 3
tempo range
+/- 30%
harmonic weight
1
rhythmic weight
0.2
loudness weight
0.2
selected songs
002. 02_perfume
011. 11_perfume
010. 10_perfume
009. 09_perfume
012. 12_perfume
003. 04_perfume
008. 08_perfume
005. 06_perfume
001. 01_perfume
007. 07_perfume
004. 05_perfume
(sorted by mashability)
auto mashup!
change to selection
add to mashup
delete from mashup
song library
load new library
001. 01_perfume
002. 02_perfume
003. 04_perfume
004. 05_perfume
005. 06_perfume
006. 07.Drinker
007. 07_perfume
008. 08_perfume
009. 09_perfume
010. 10_perfume
011. 11_perfume
012. 12_perfume
013. KyaryPamyuPamyu.PonPonPon
014. Perfume.NaturalNiKoshite
select all
pick 10
clear

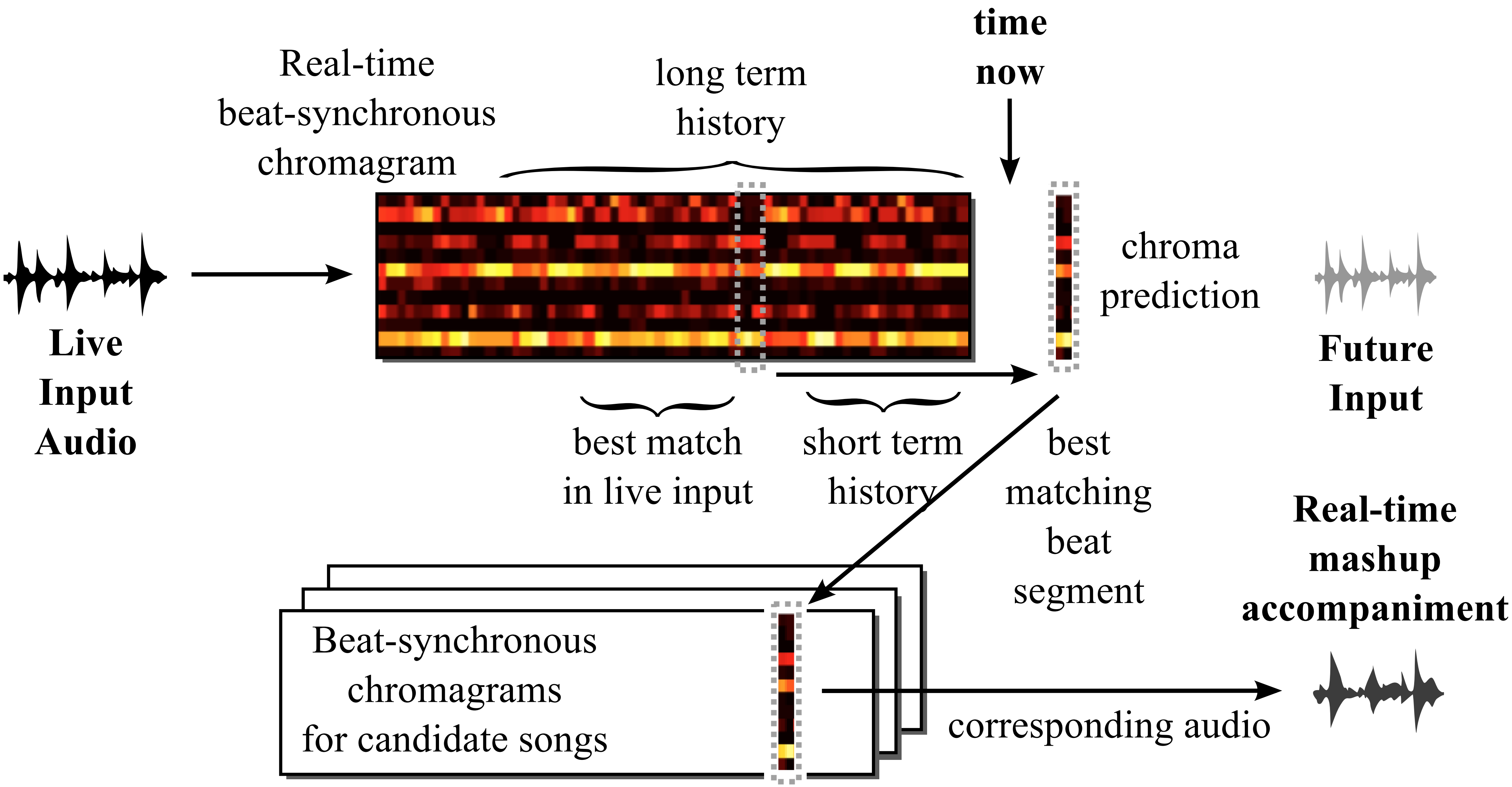
Artist/Style Mode



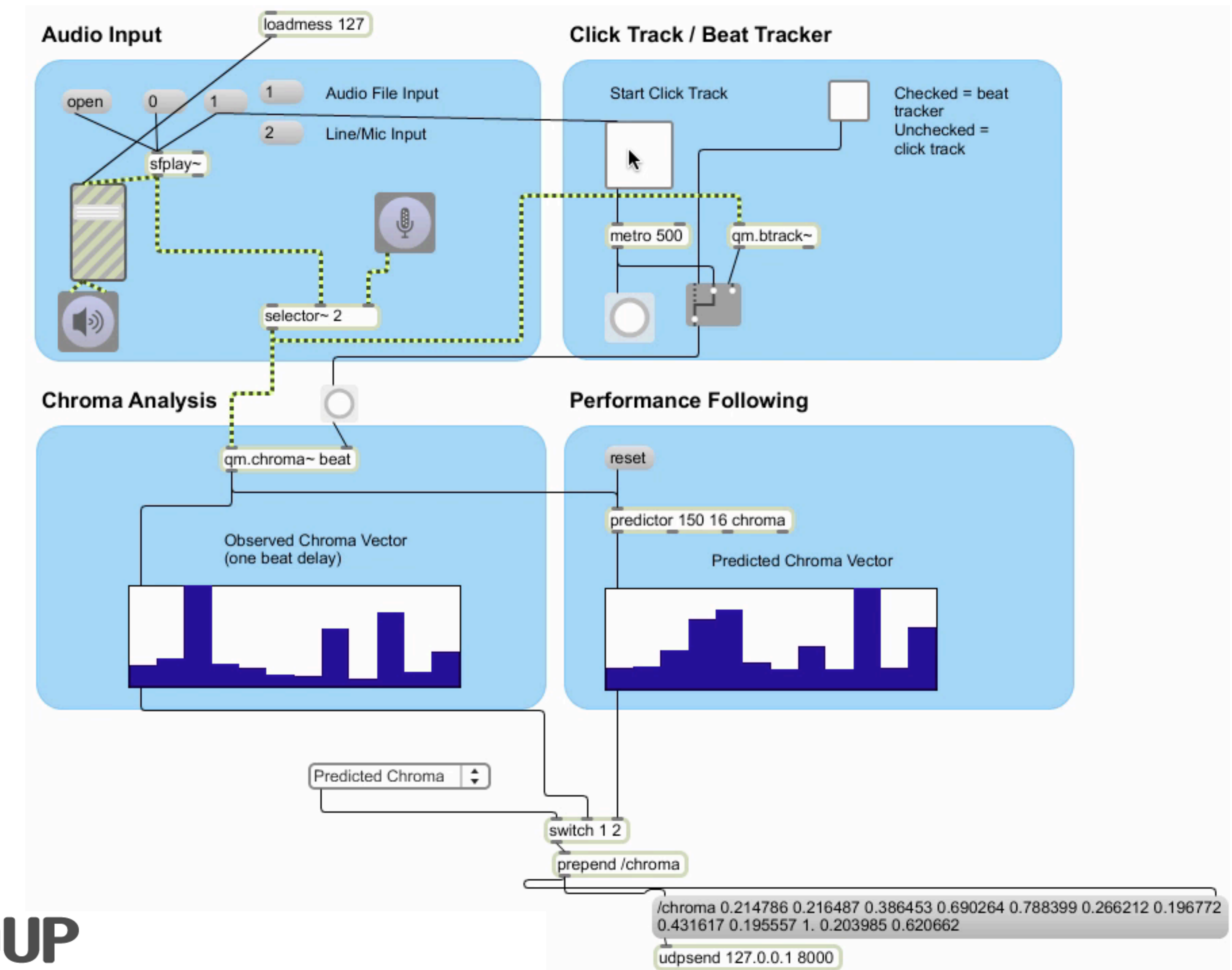
Musician Mode



Real-time mashup



Real-time mashup:
DJ mode



Conclusions

- **AutoMashUpper** - assistive technology to help users make music mashups
  - **interactivity** is important
- Automatic approach to mashability can reveal **unknown relationships** between songs
- Lots of ways to extend the original concept to allow greater scope of musical creativity and interaction