

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

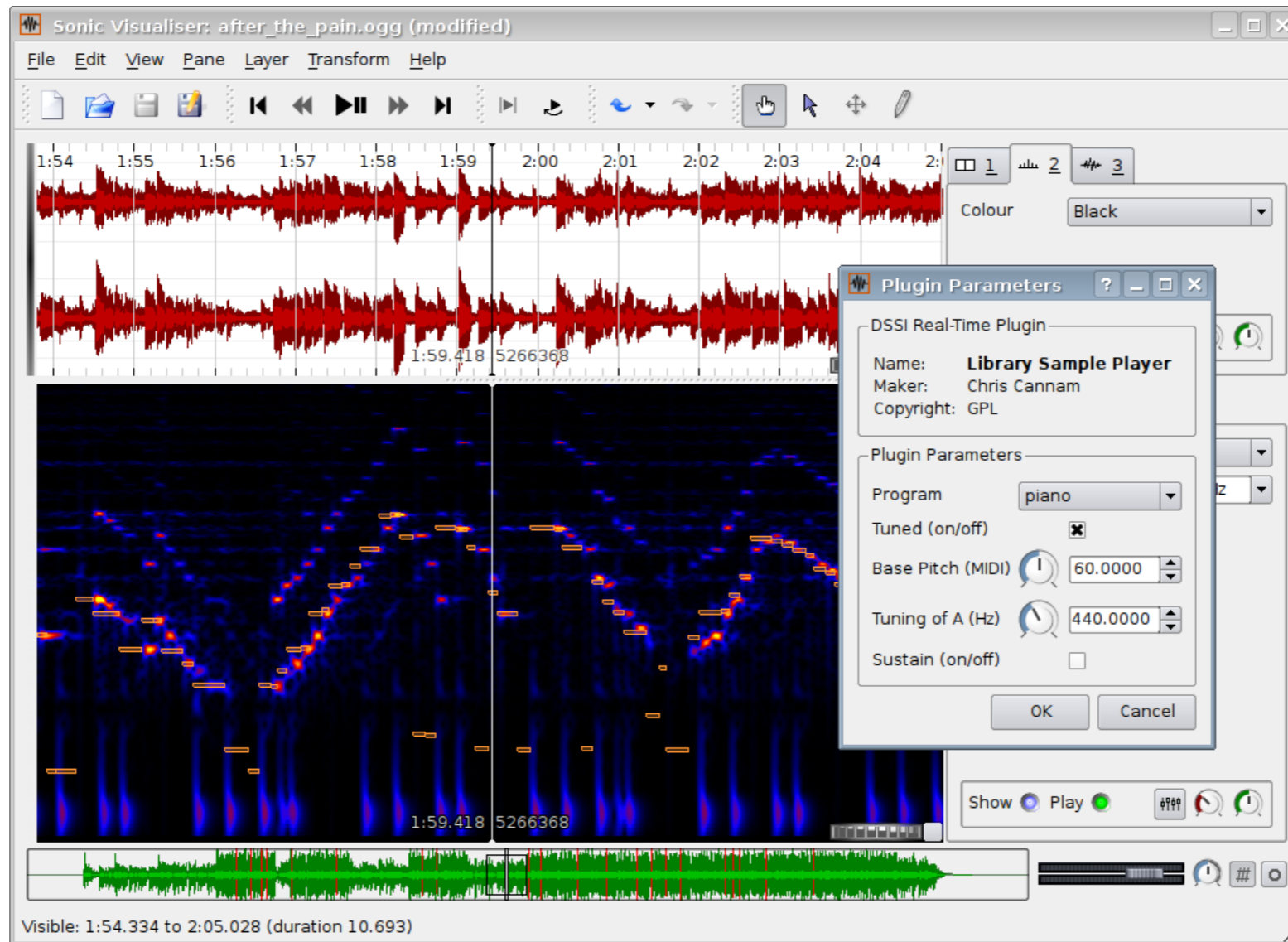
Focus of today's talk

- 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

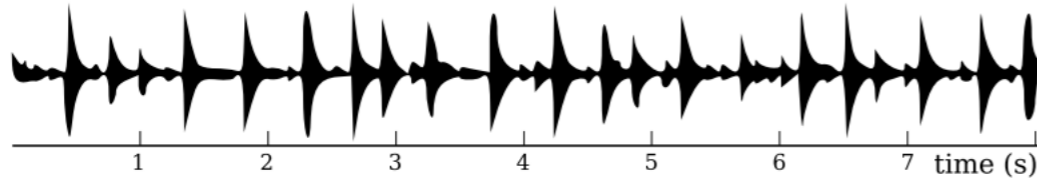
Sonic Visualiser



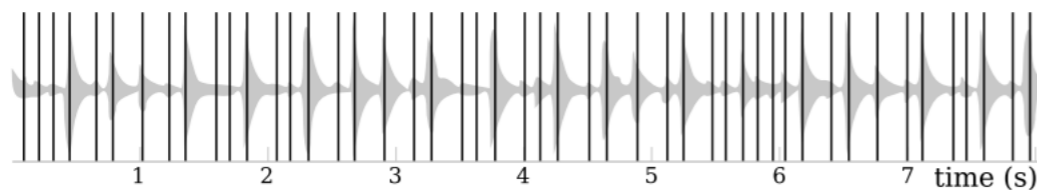
- Free open source analysis and visualisation tool for musical audio - www.sonicvisualiser.org

Musical context

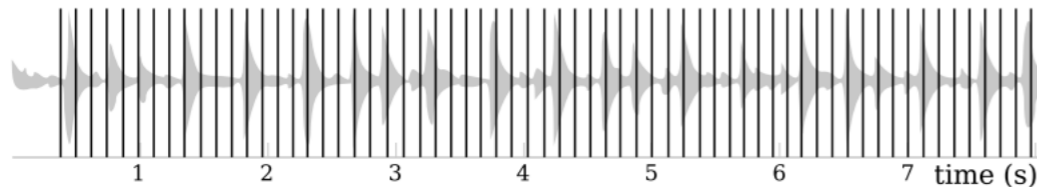
(a) Audio signal



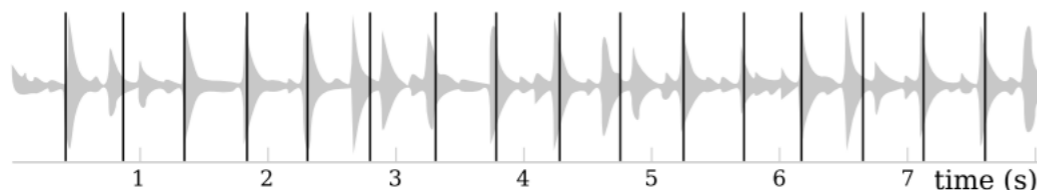
(b) Onset locations



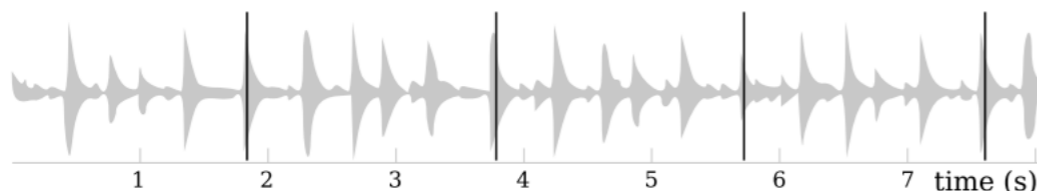
(c) Tatum level events



(d) Beat locations



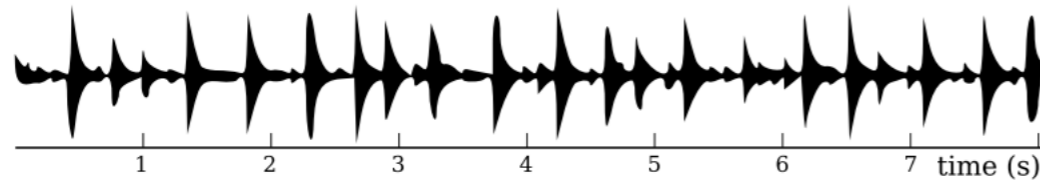
(e) Bar boundaries



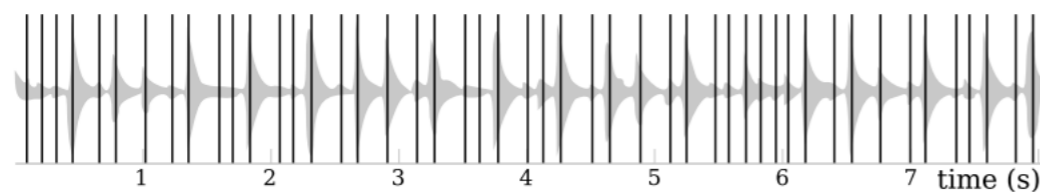
- onsets - start times of note events
- tatum - fastest level
- beat - comfortable tapping level
- downbeats - grouping beats into bars

Musical overview

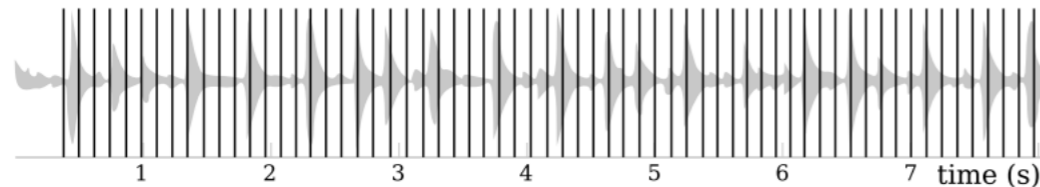
(a) Audio signal



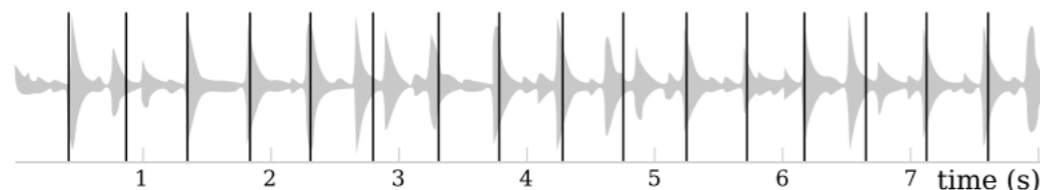
(b) Onset locations



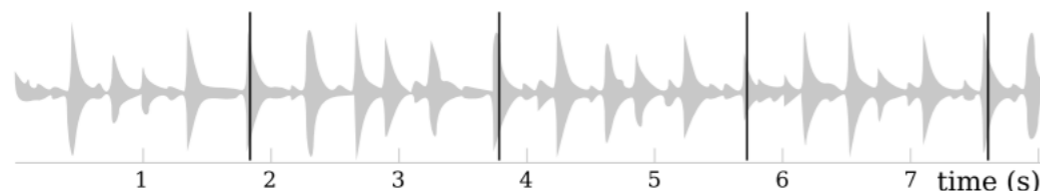
(c) Tatum level events



(d) Beat locations



(e) Bar boundaries

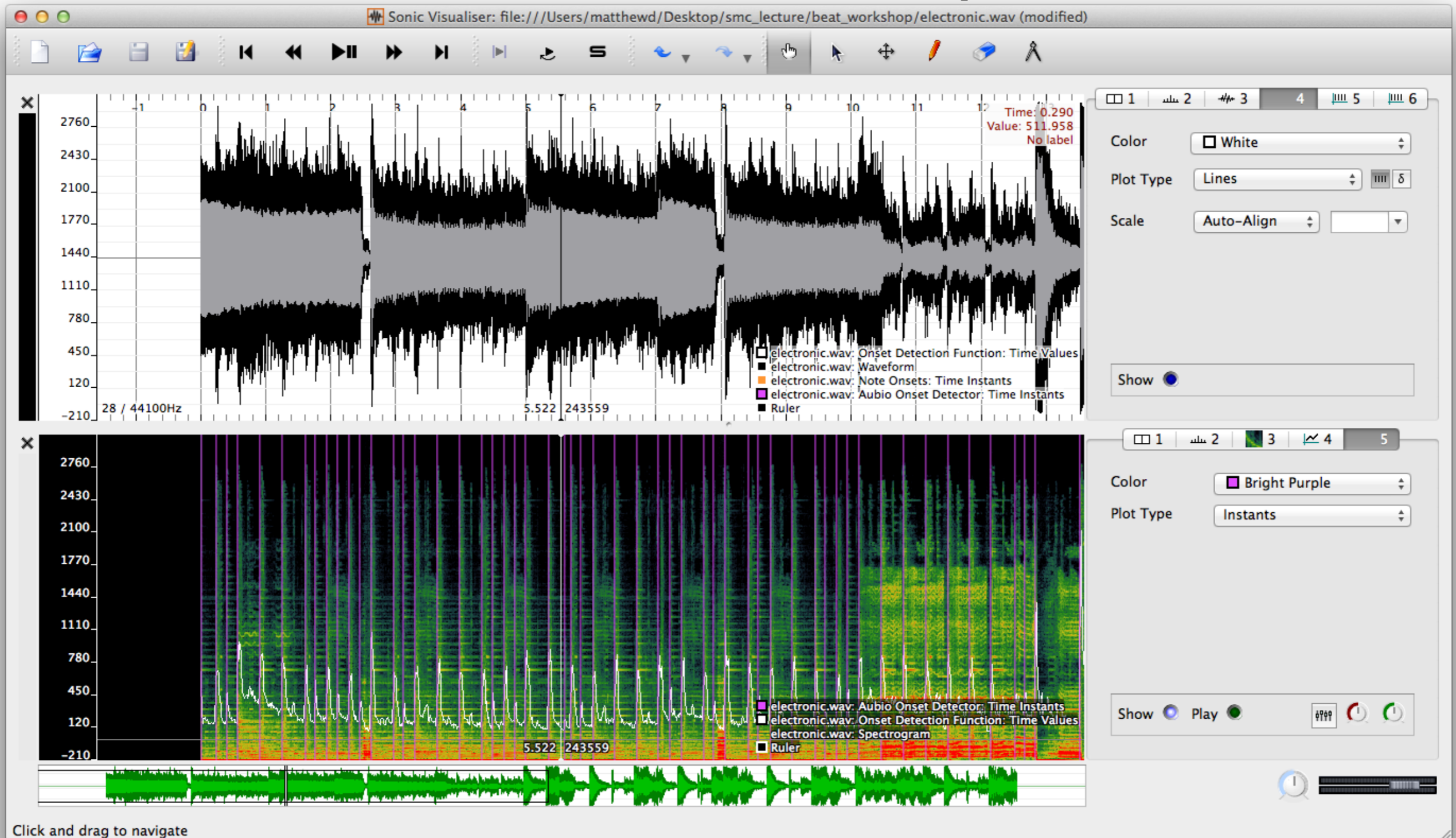


- **onsets** - start times of note events
- tatum - fastest level
- beat - comfortable tapping level
- downbeats - grouping beats into bars

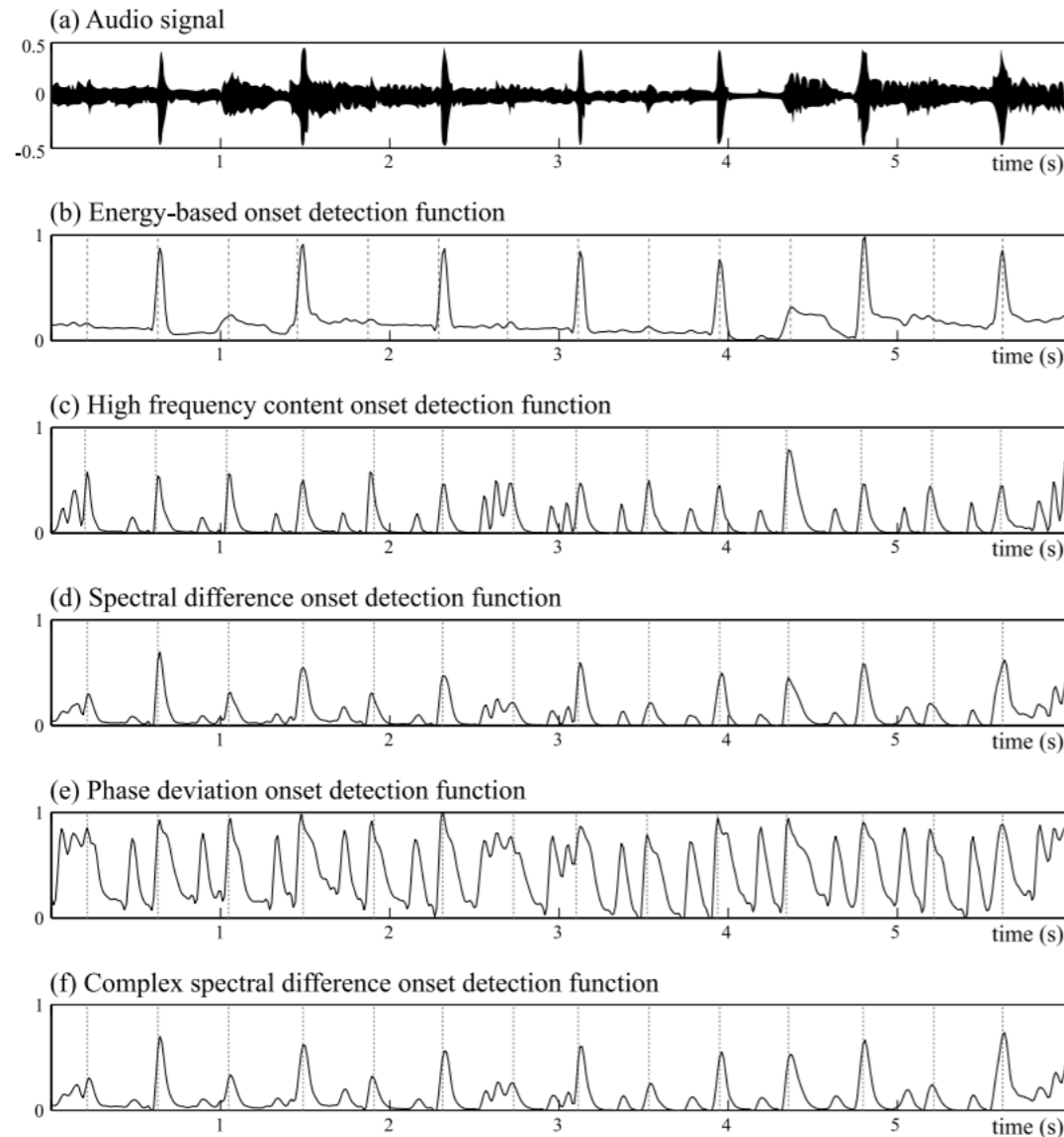
Basic approach: onset detection

- 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



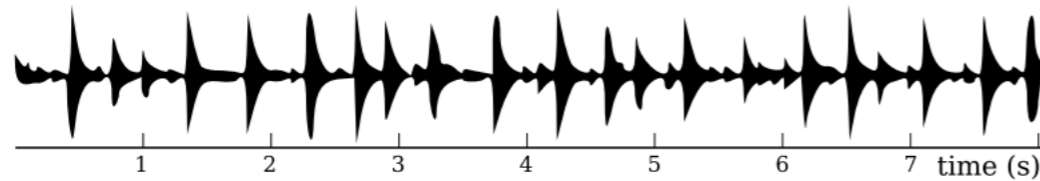
Many different onset detection functions



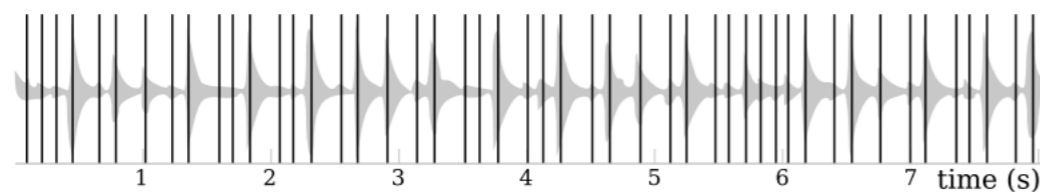
- We can make lots of different onset detection functions
- Using: energy, phase, spectral difference, emphasis on high frequencies

Musical overview

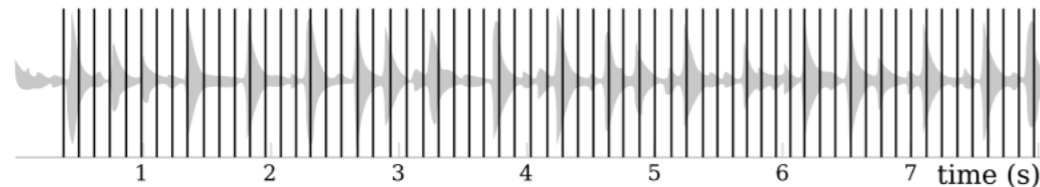
(a) Audio signal



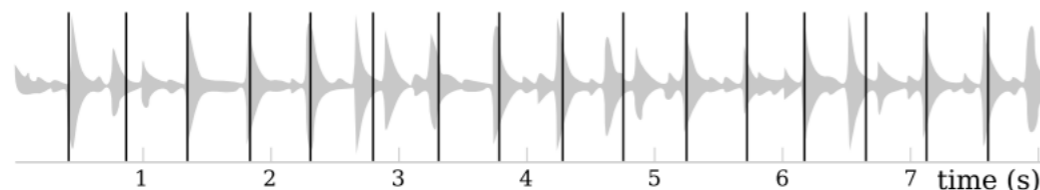
(b) Onset locations



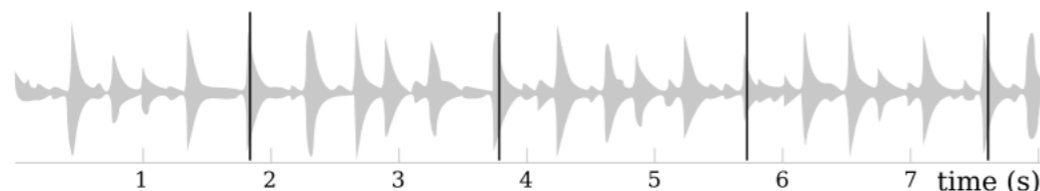
(c) Tatum level events



(d) Beat locations



(e) Bar boundaries



- 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 do this without thinking - it's easy, right?
 - But how do we make the computer “**feel the beat**”?

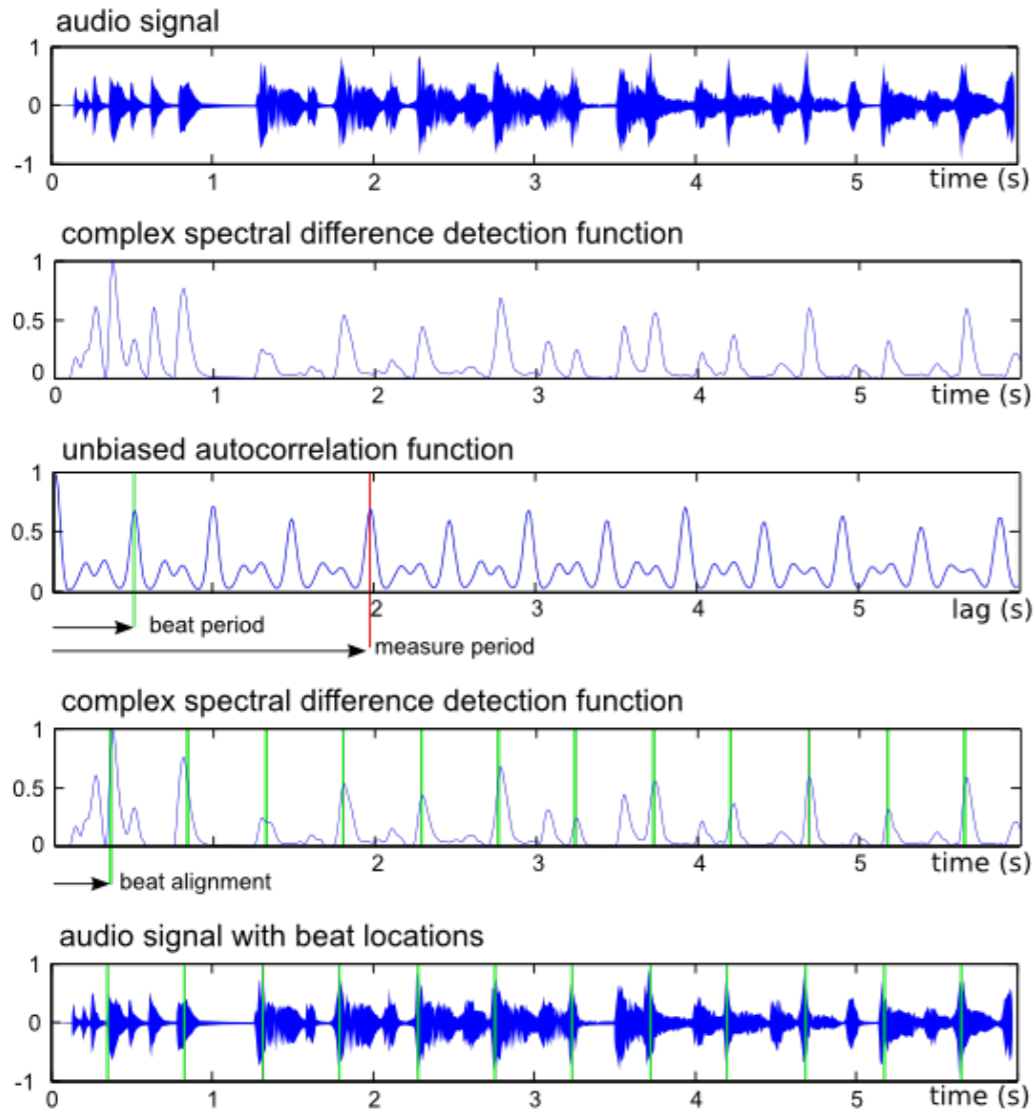
How is beat tracking done?

- 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

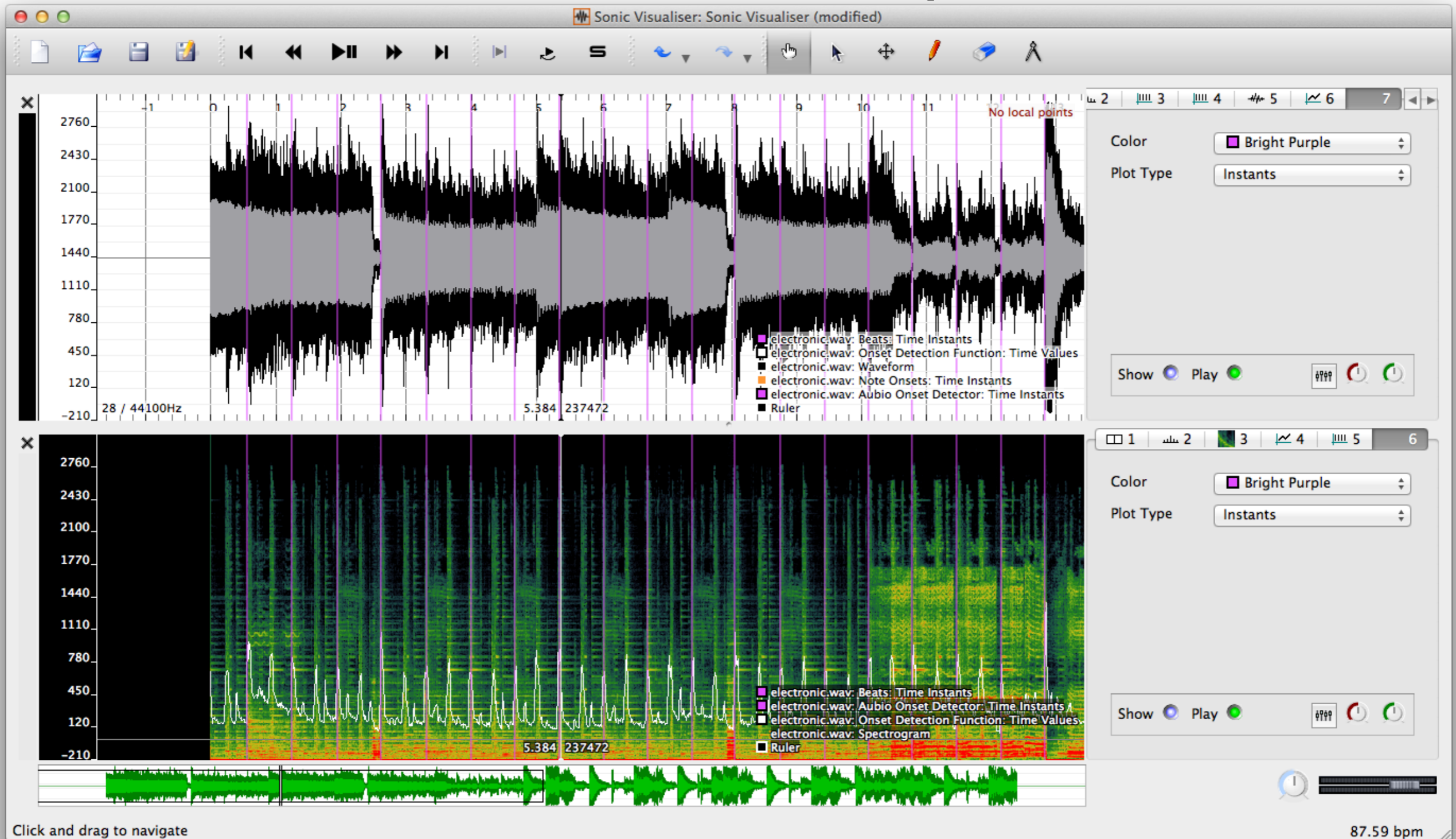
Graphical example



- 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?

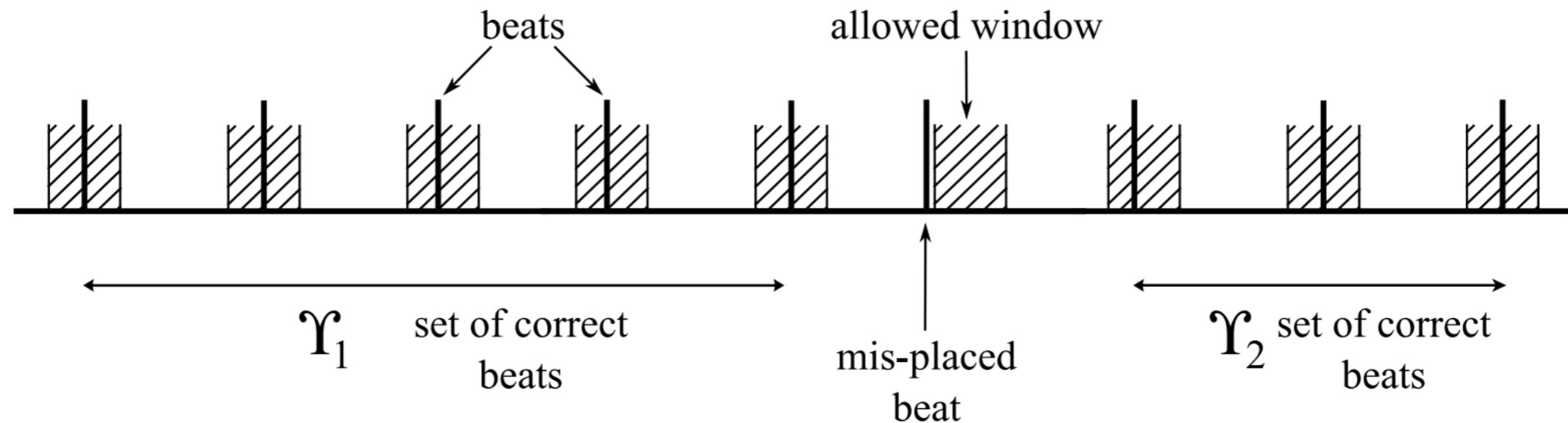
- **Subjectively**

- 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

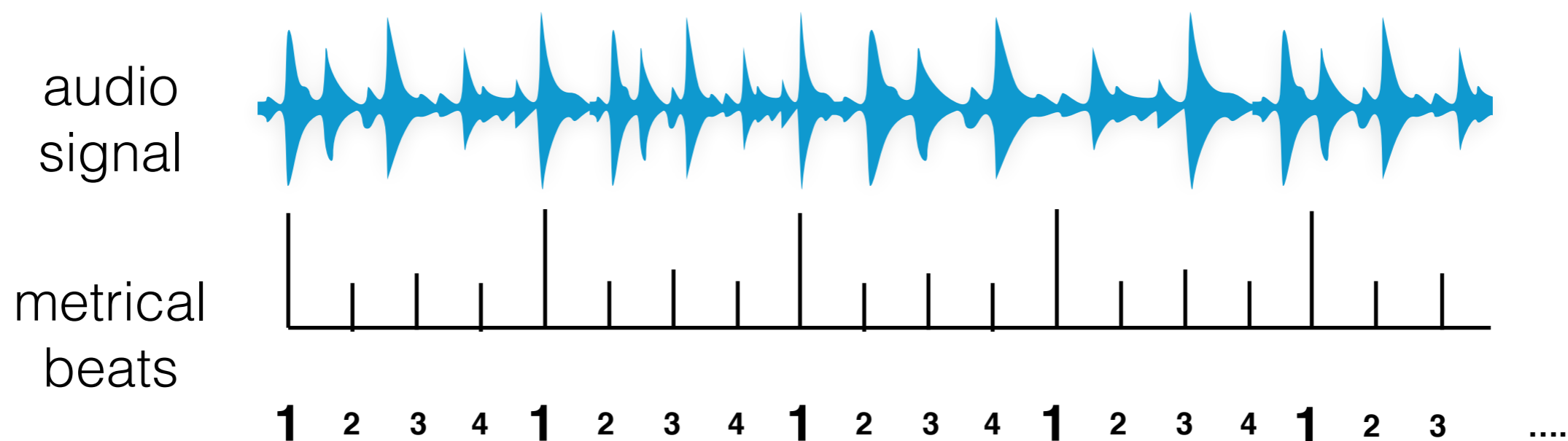
Part II

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



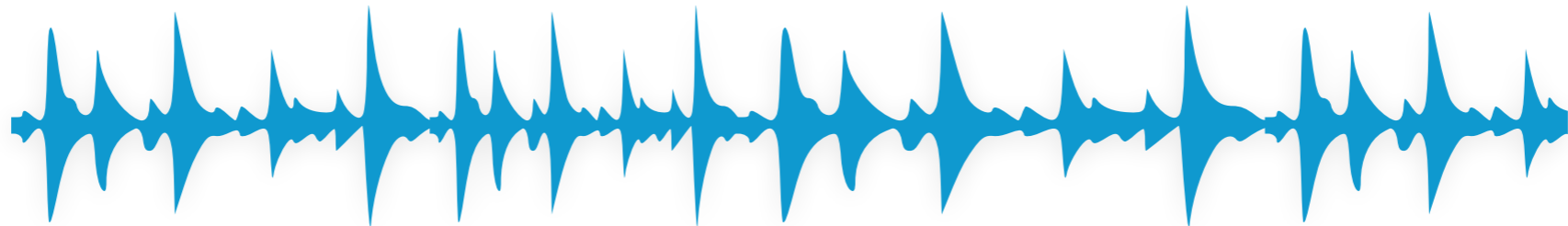
- The “metrical beats” are the main representation we’ll work with for these simple transformations

Beat Tracking

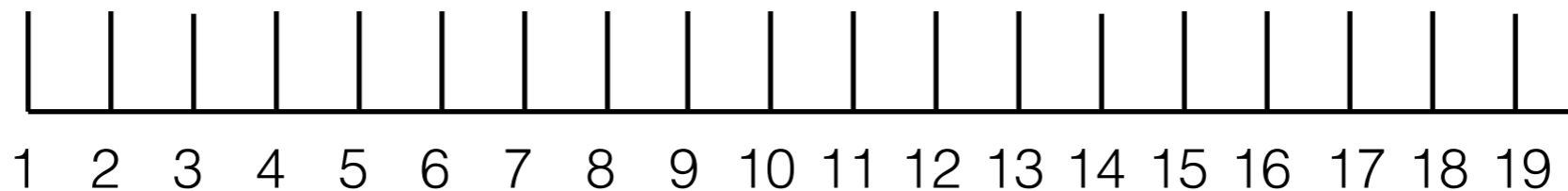
- We'll explore:
 - scrambling the music in two different ways
 - changing the rhythmic structure
 - some automatic remixing

Beat Randomizer

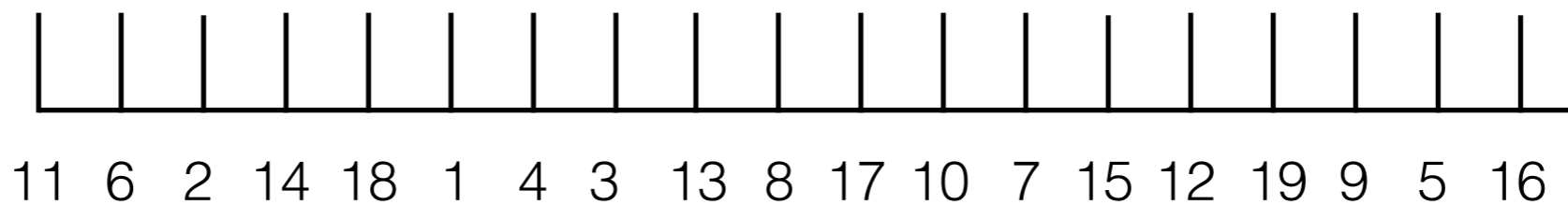
audio
signal



beat
locations



random
beats
locations

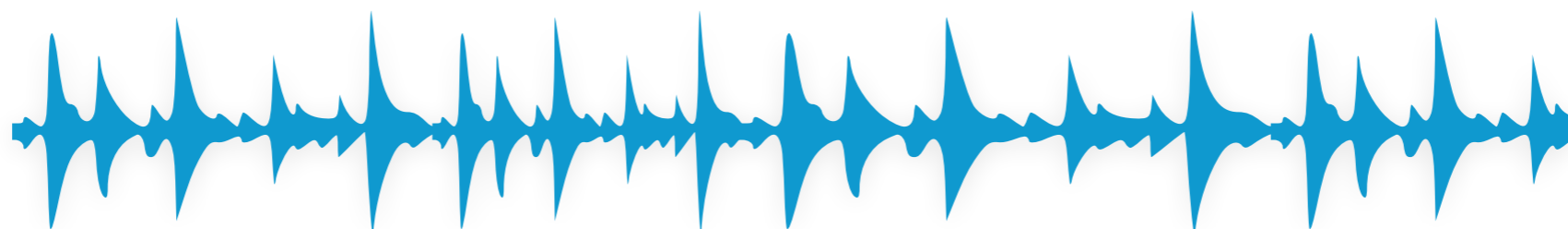


Beat Randomizer with Metre

- 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

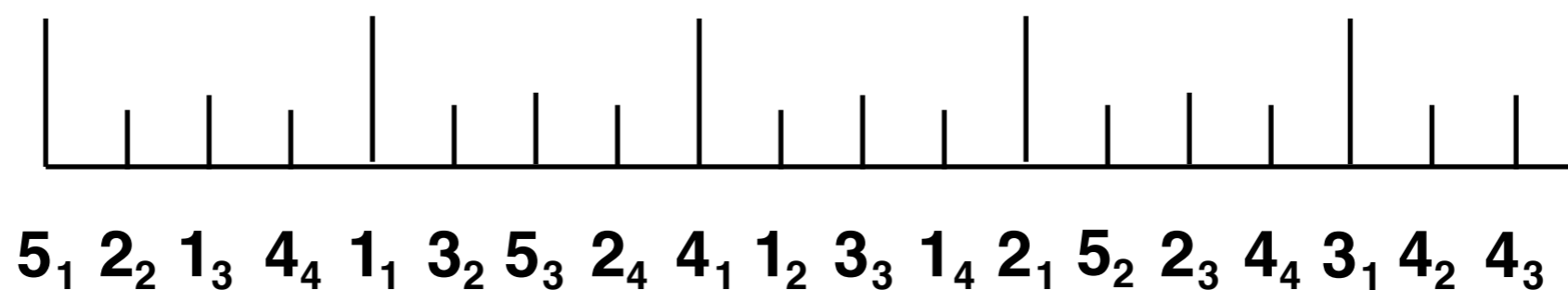
audio
signal



metrical
beats



random
metrical
beats



Beat Randomizer with Metre

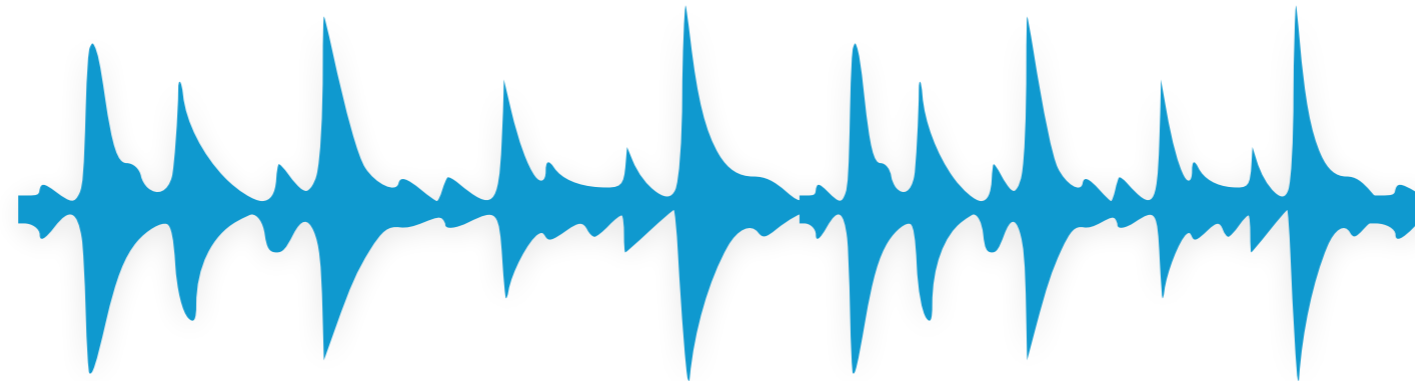
- We can vary number of sub-divisions per beat,
- e.g. 2, 4 or 8

Beat Swinger

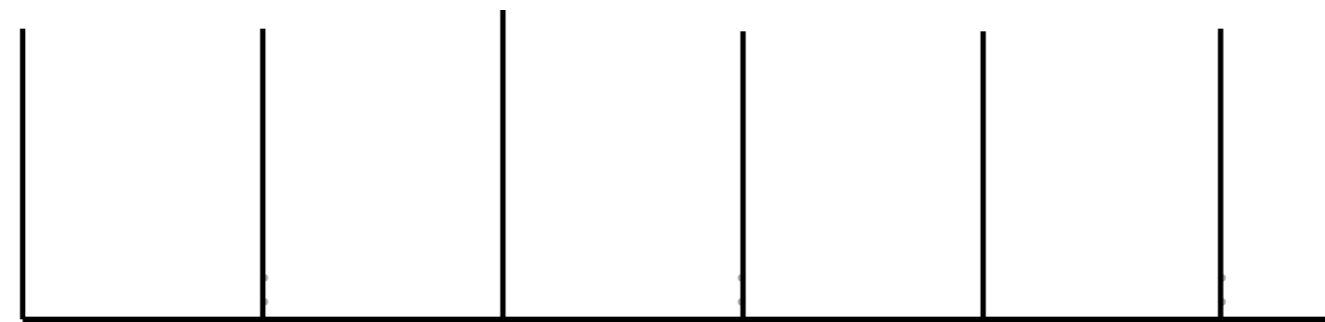
- Given the beat times we can use a **time-stretching** tool to alter the rhythmic structure of the music
- Time-stretching alters the speed of the music without changing the pitch
- To add **swing-feel** we modify the duration of 1/8th notes (i.e. half-beats) in an alternating *long-short* pattern
-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

Beat Swinger

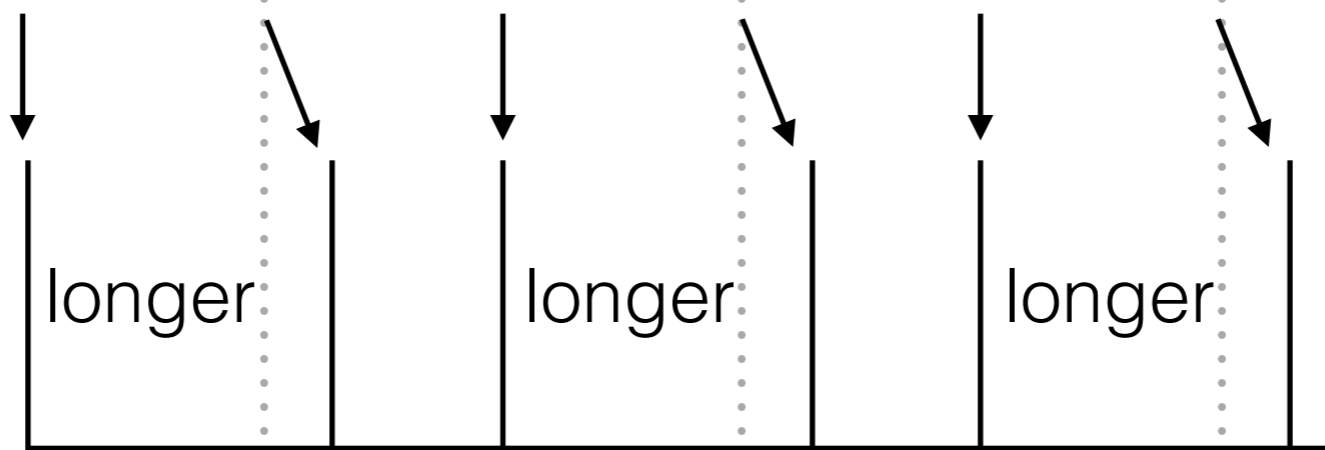
audio
signal



1/8th
note
locations



swung
beats



shorter

shorter

use
time-
stretching

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 mixer

- 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 mixer

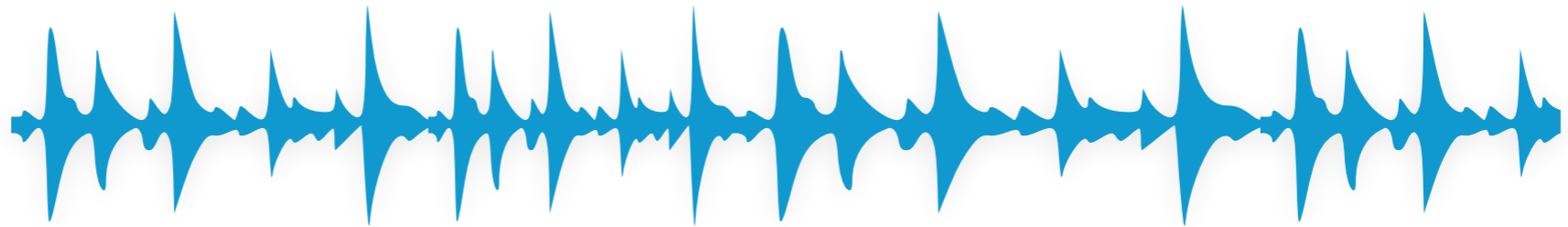
- Given two input music files, we'll follow the “Beat Randomizer with Metre” process for each of them
- But then, also randomly **switch** between which song we playback at each beat
- Making sure that we've **time-stretched** each the beats to be at the same tempo

Beat Remixer

audio signal A



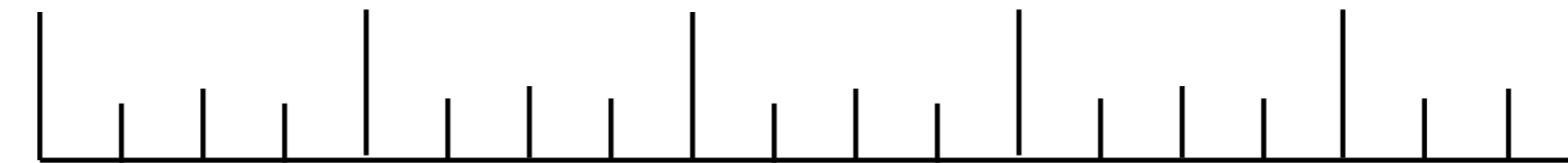
audio signal B



random song

A B A B B A B B A A A B B A B A B B A

random
metrical beats



5₁ 2₂ 1₃ 4₄ 1₁ 3₂ 5₃ 2₄ 4₁ 1₂ 3₃ 1₄ 2₁ 5₂ 2₃ 4₄ 3₁ 4₂ 4₃

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

automashupper

AutoMashUpper by Matthew Davies, Philippe Hamel
Kazuyoshi Yoshii and Masataka Goto (AIST)

load song reset quit save

input - billwithers lovelyday

skip bwd play mix stop skip fwd

mashup visualizer

segmentation level

more mashup | more input
playback balance

mashability controls

key shift range +/- 3
tempo range +/- 30%
harmonic weight 1
rhythmic weight 0
loudness weight 0

selected songs

- 045. 0196_nosleeptillbrooklyn
- 052. 0272_teachmehowtojerk
- 105. 0589_bestieverhad
- 118. 0750_inedyourlove
- 124. 0798_lovelikethis
- 235. DoobaWooba.BassSolo
- 244. Eminem.TheRealSlimShady
- 293. KennyLoggins.DangerZone
- 321. MikiSantamaria.SlapBass
- 414. YvonneElliman.IfICantHaveYou

song library load new library

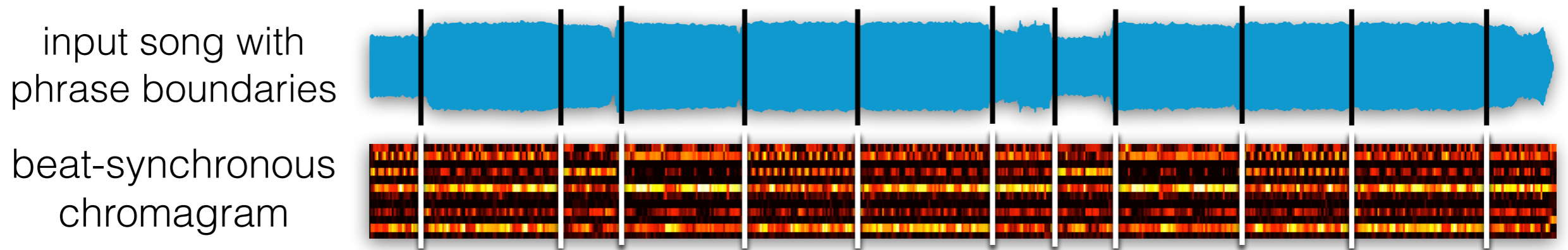
- 001. 0001_12step
- 002. 0003_6foot7foot
- 003. 0012_aroundtheworld
- 004. 0018_bassdownlow
- 005. 0019_beautiful
- 006. 0023_bewareoftheboys
- 007. 0025_blackandtan
- 008. 0026_blackandyellow
- 009. 0030_bodymovin
- 010. 0032_boomboompow
- 011. 0037_breakyourheart
- 012. 0040_bustamove
- 013. 0043_callmemaybe
- 014. 0045_cantgetyou
- 015. 0051_cmonnrideit
- 016. 0054_conceited
- 017. 0056_control
- 018. 0057_crankthat
- 019. 0059_cupidshuffle
- 020. 0062_dance
- 021. 0063_daysgoby
- 022. 0070_dipitlow
- 023. 0078_donttouchme
- 024. 0079_dontyouwantme
- 025. 0084_dropitlikeitshot
- 026. 0087_evacuate
- 027. 0092_fergalicious
- 028. 0094_fireflies
- 029. 0099_forgetyou
- 030. 0102_gangstaluv
- 031. 0106_getitshawty
- 032. 0107_getlow
- 033. 0108_getup
- 034. 0109_geturfreakon
- 035. 0110_girlsandboys
- 036. 0111_girlsonfilm
- 037. 0112_giveitup
- 038. 0135_igottafeeling
- 039. 0139_imcomingout
- 040. 0144_ishotthesheriff
- 041. 0150_justdance
- 042. 0152_kingofthedancehall
- 043. 0168_maneater
- 044. 0179_moveslikejagger

auto mashup! change to selection add to mashup delete from mashup

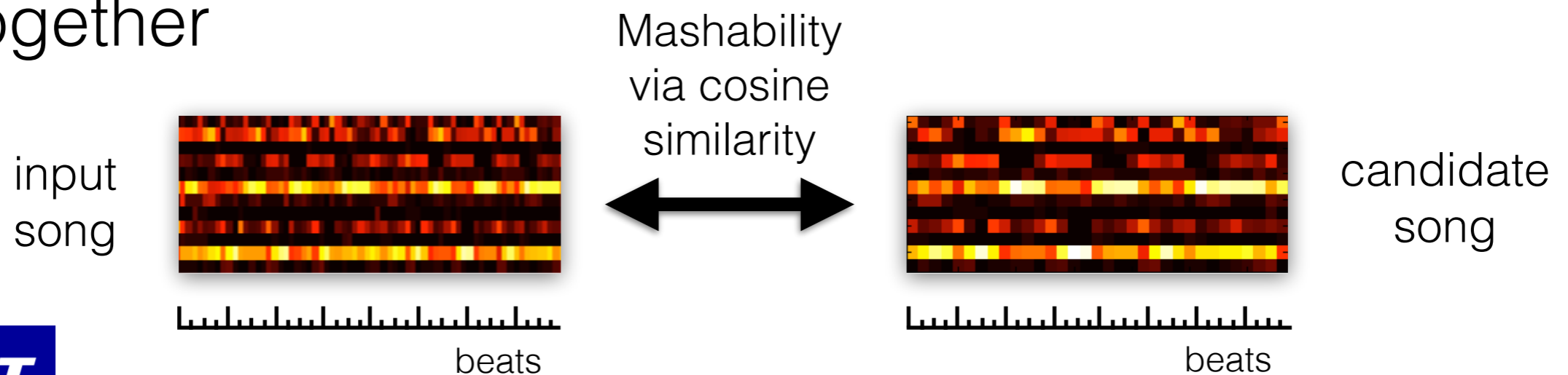
select all pick 10 clear

System Components

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

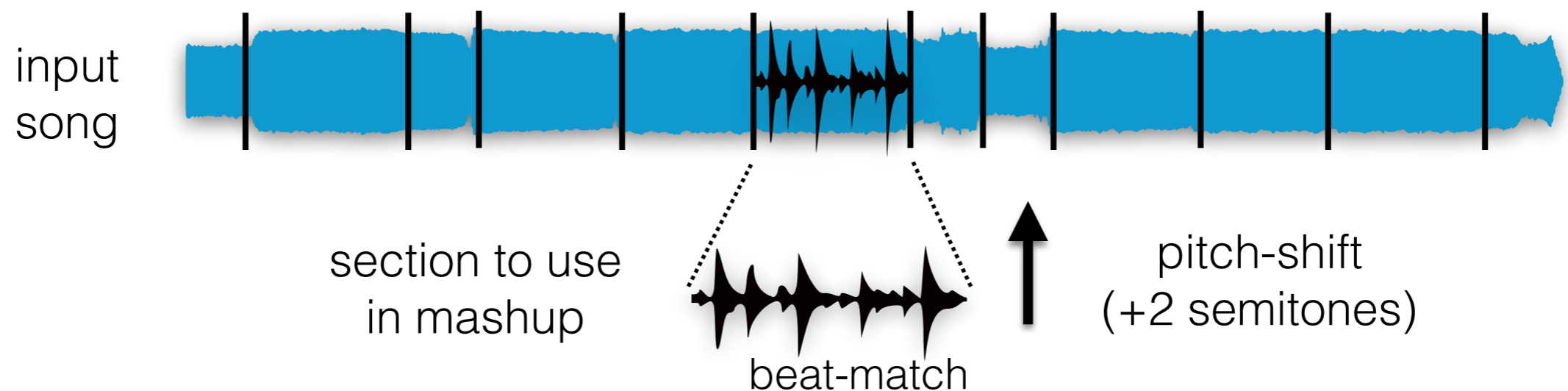


- **“Mashability”** - measure how well two sections fit together

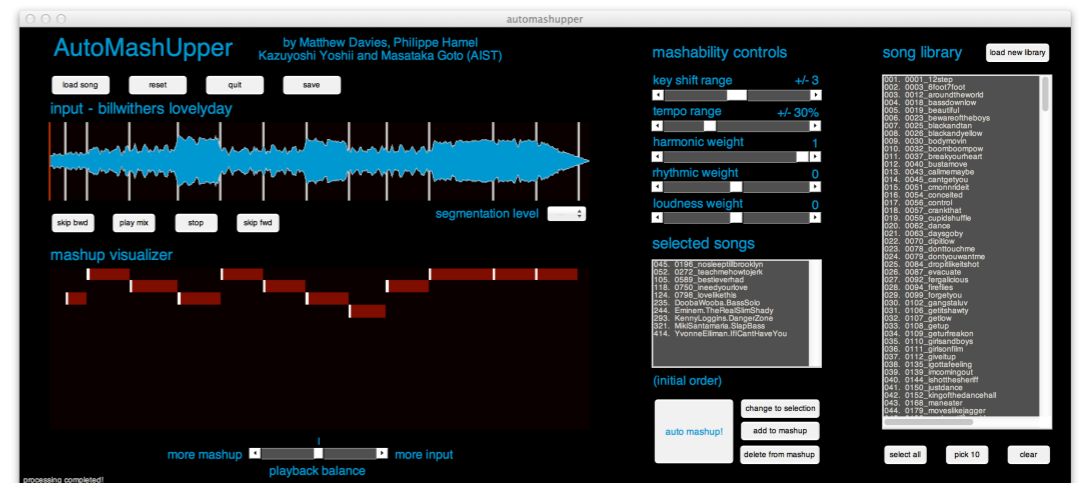


System Components

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

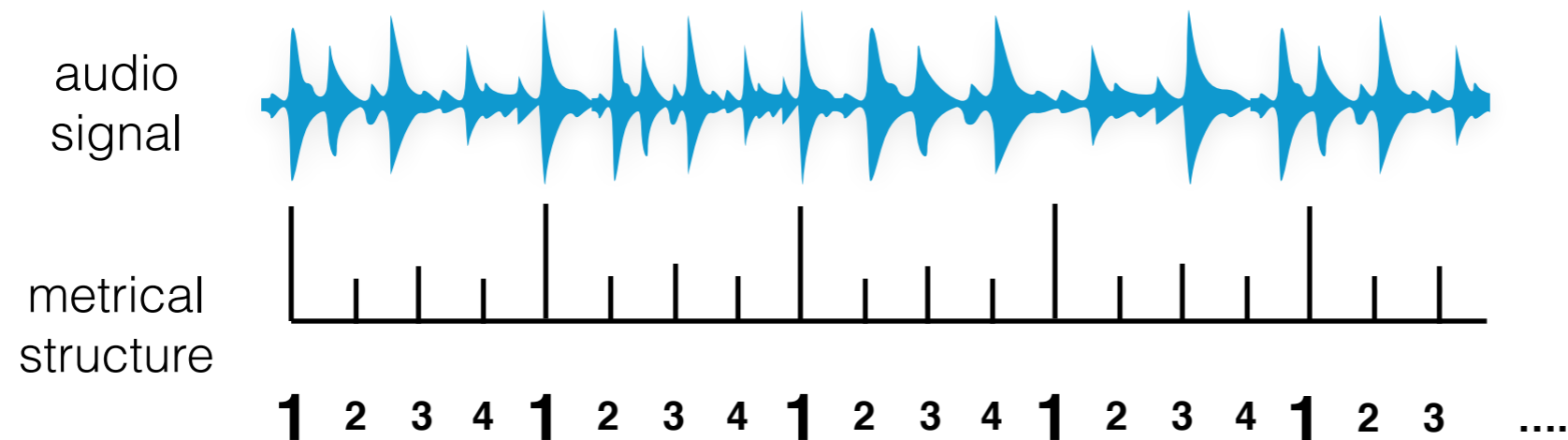


- **User Interface**
allow users to interact with mashup creation process and manipulate result



Beat tracking

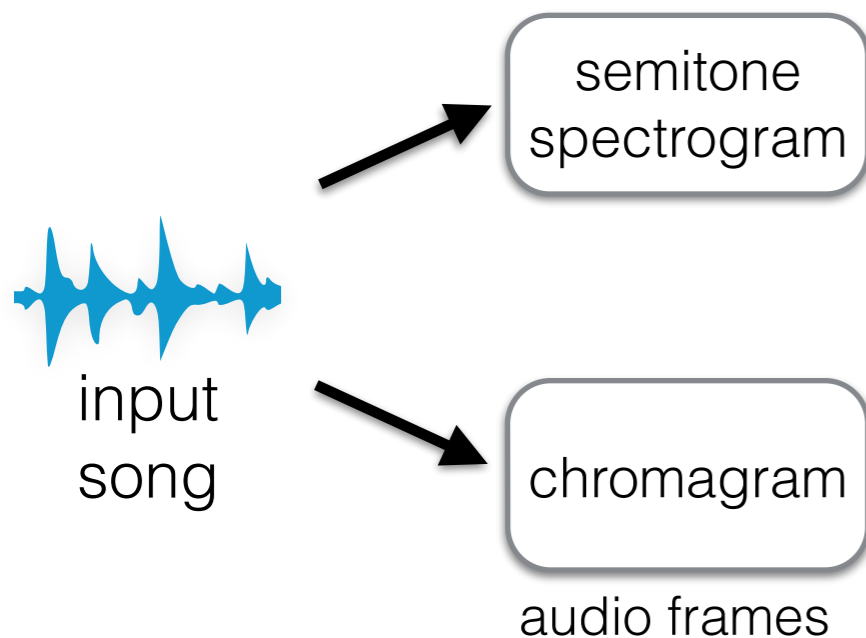
- **Critical** first step in mashup system
 - Enables temporal **synchronisation** between songs
 - Underpins **all** subsequent analysis



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

Beat-synchronous harmonic representations

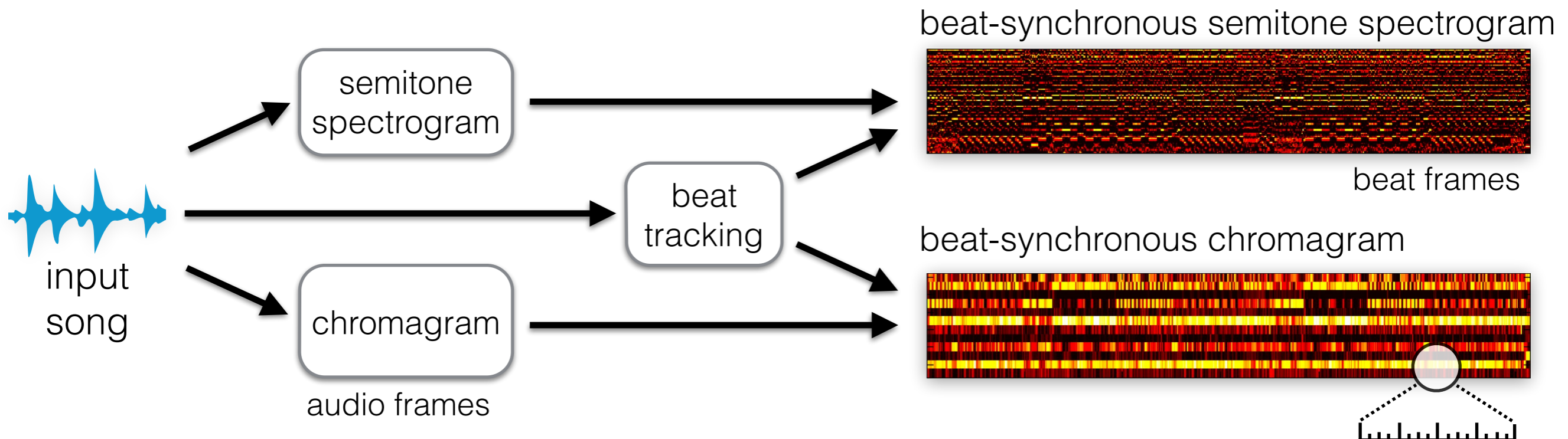
- AutoMashUpper uses **two** harmonic signal representations
 - **semitone spectrogram** and **chromagram**
- Use beat information to make **beat-synchronous** versions



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

Beat-synchronous harmonic representations

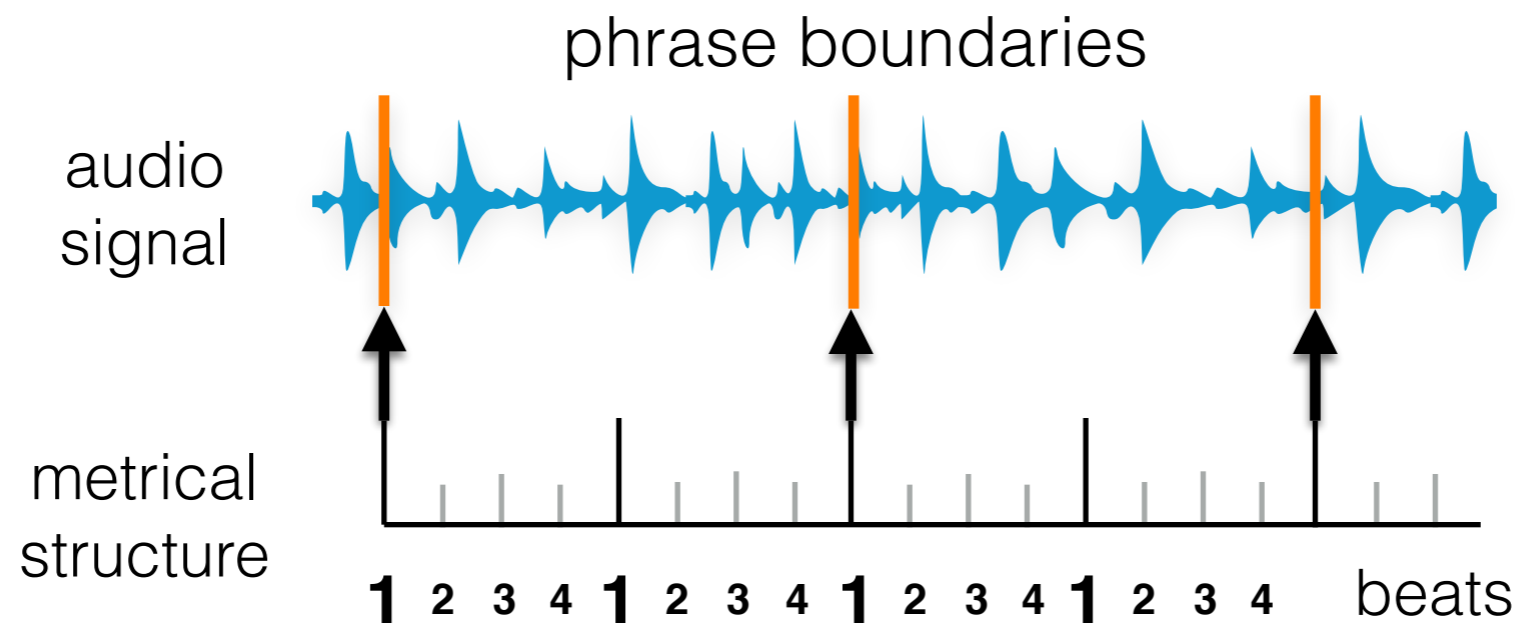
- AutoMashUpper uses **two** harmonic signal representations
 - **semitone spectrogram** and **chromagram**
- Use beat information to make **beat-synchronous** versions



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

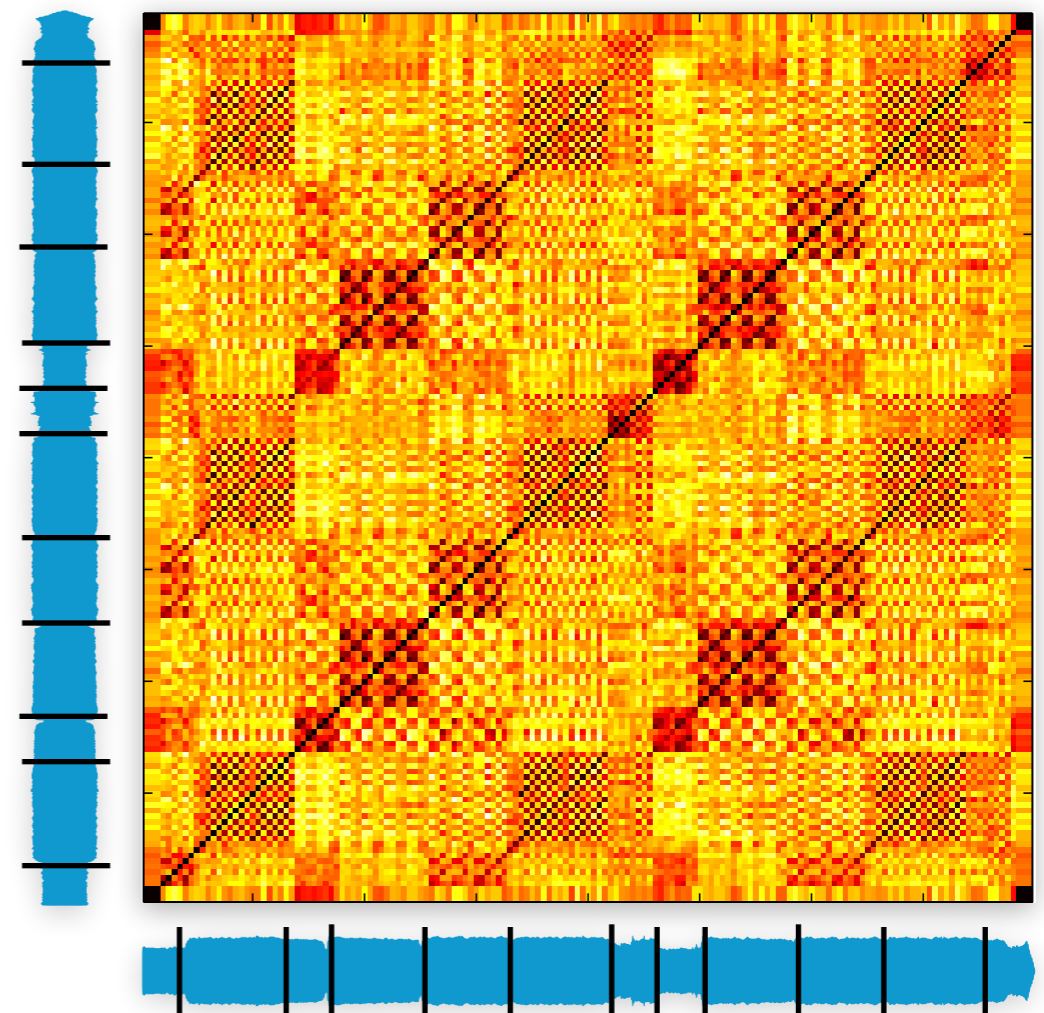
Phrase level segmentation

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



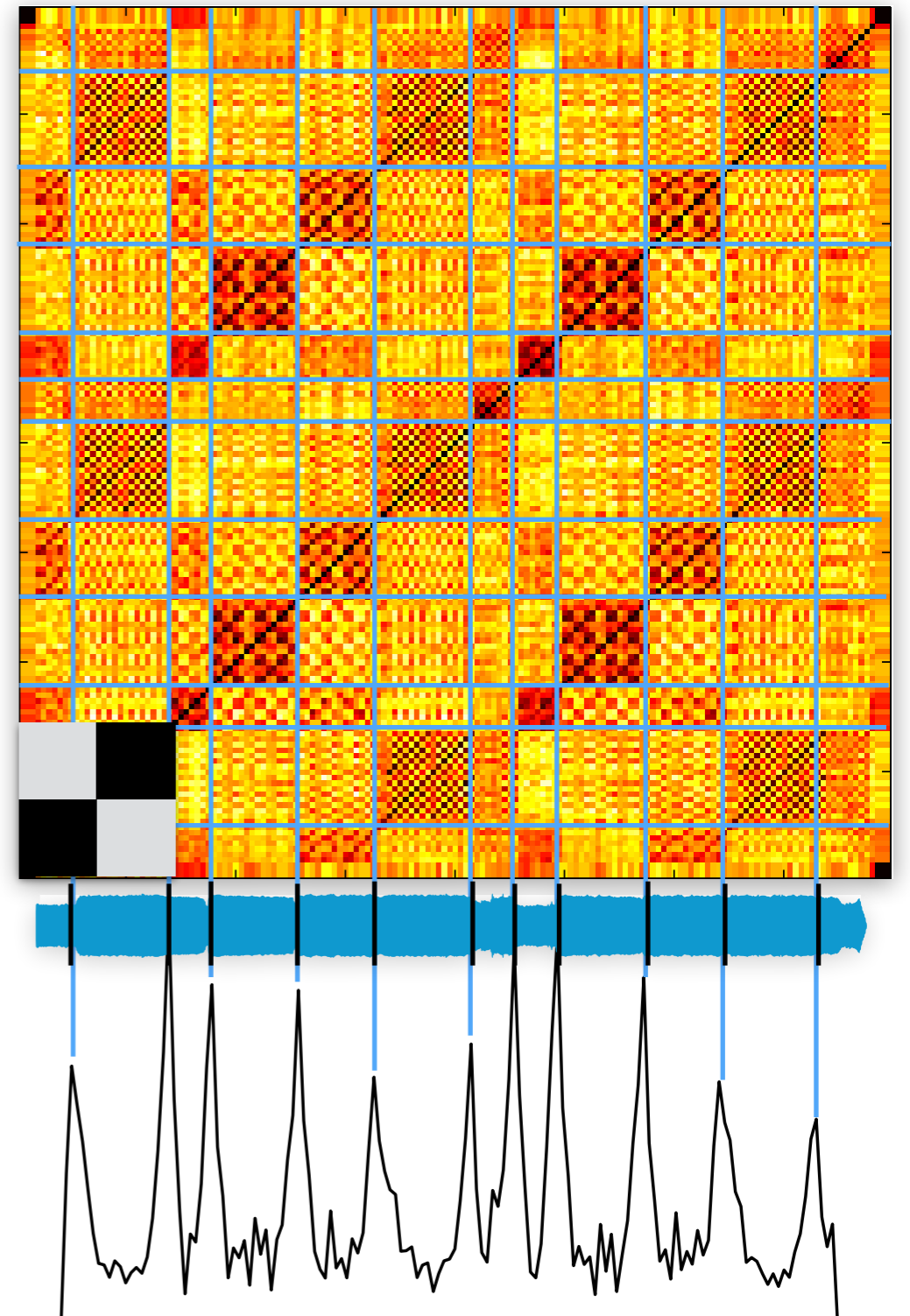
Phrase level segmentation

- 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



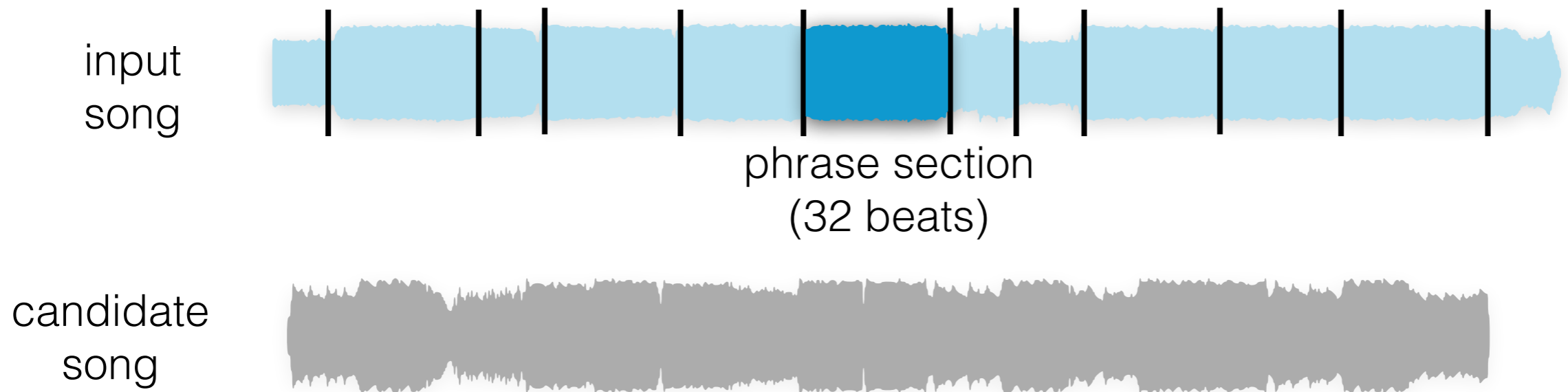
Phrase level segmentation

- 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

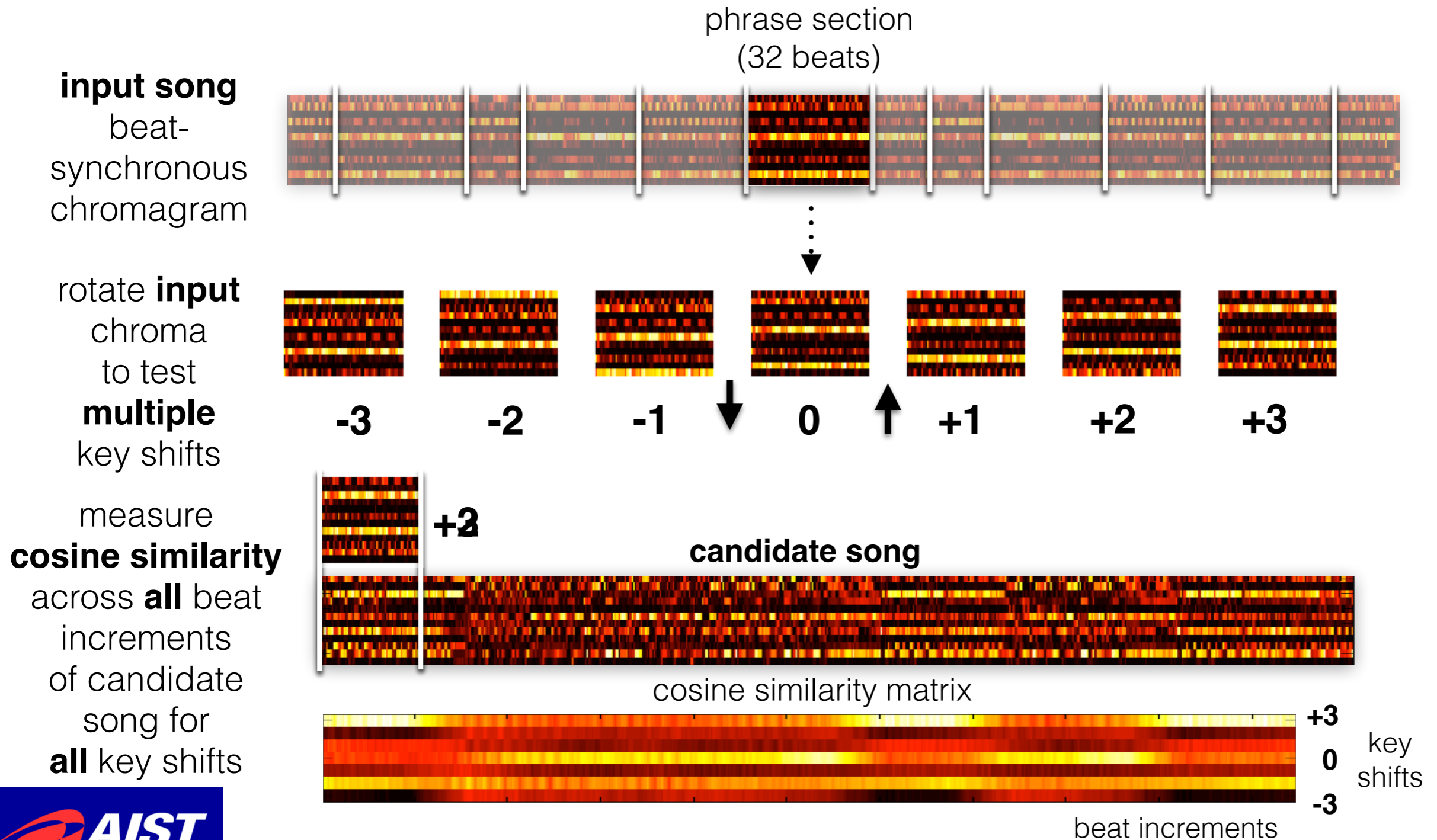
- For each phrase section of input song, search for the “best match” in the user’s song library
- Estimate **mashability** between songs



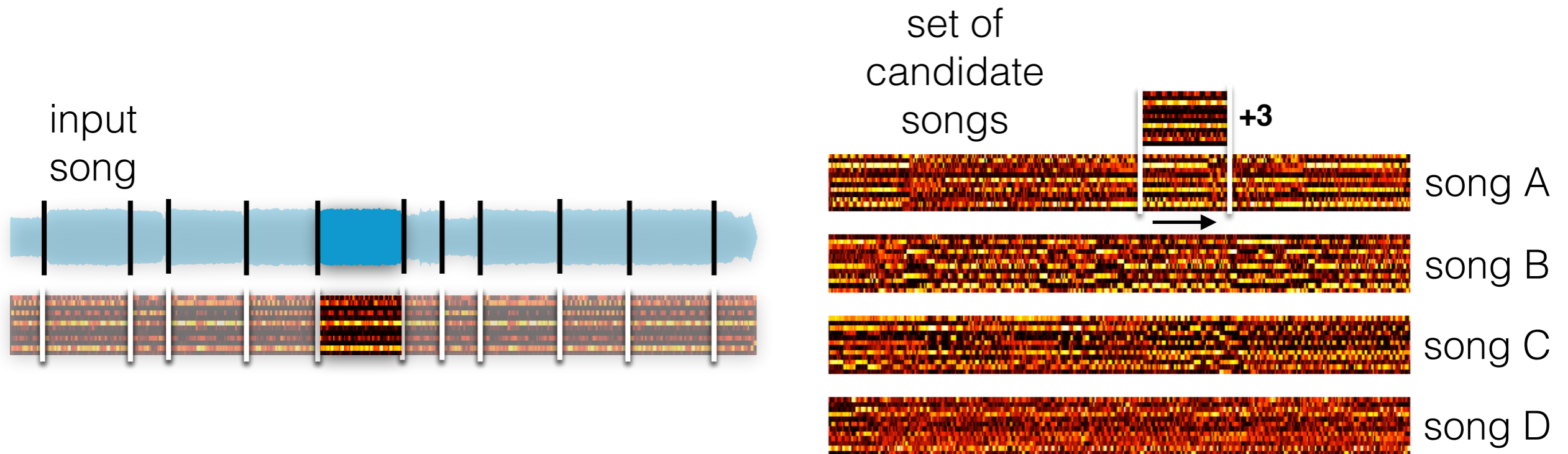
where is best matching 32 beat region in the candidate song?

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

Mashability estimation



Mashability estimation

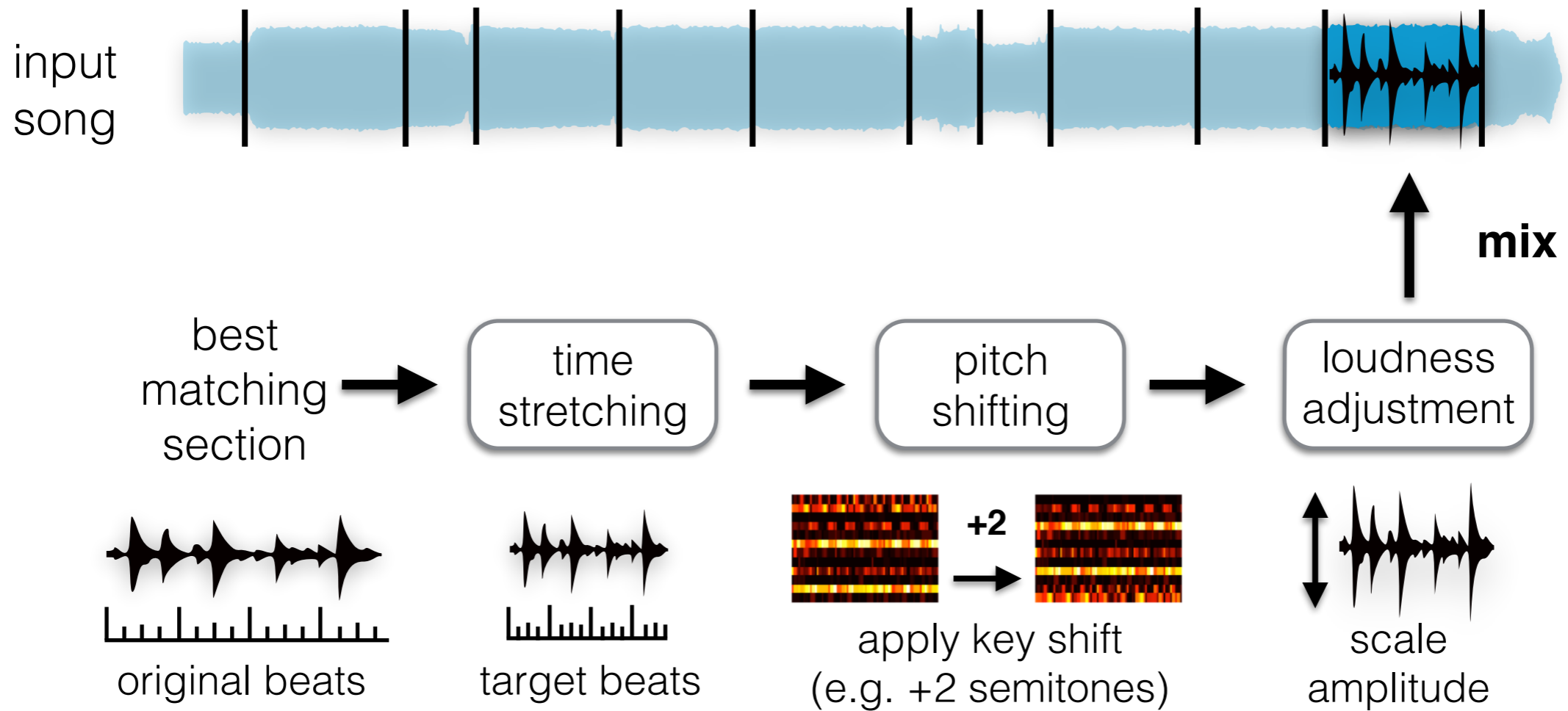


- Repeat mashability estimation across all candidate songs and **rank** maximum mashability score per song
- Search space is **huge**
 - e.g. **7** key shifts x **500** beats = **3500** possible mashups for **one** phrase section matched to **one** song, but computation is acceptable (for up to 500 songs)

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



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

The screenshot shows the AutoMashUpper application window titled "automashupper". The interface is divided into several sections:

- Header:** "AutoMashUpper" logo and credits: "by Matthew Davies, Philippe Hamel, Kazuyoshi Yoshii and Masataka Goto (AIST)".
- Top Controls:** Buttons for "load song", "reset", "quit", and "save".
- Input Section:** "input - 03_perfume" with a waveform visualization and a "segmentation level" slider.
- Playback Controls:** Buttons for "skip bwd", "play mix", "stop", and "skip fwd".
- Mashup Visualizer:** A grid of red blocks representing the mashup structure.
- Playback Balance:** A slider between "more mashup" and "more input".
- Current Status:** "current section : 4 current beat : 138".
- Mashability Controls:** Sliders for "key shift range" (+/- 3), "tempo range" (+/- 30%), "harmonic weight" (1), "rhythmic weight" (0.2), and "loudness weight" (0.2).
- Selected Songs:** A list of songs sorted by mashability, including "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", and "004. 05_perfume".
- Song Library:** A list of 14 songs, including "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", and "014. Perfume.NaturalNikoshite".
- Actions:** Buttons for "auto mashup!", "change to selection", "add to mashup", "delete from mashup", "select all", "pick 10", and "clear".

Artist/Style Mode

The screenshot shows the AutoMashUpper application window titled "automashupper". The interface is divided into several sections:

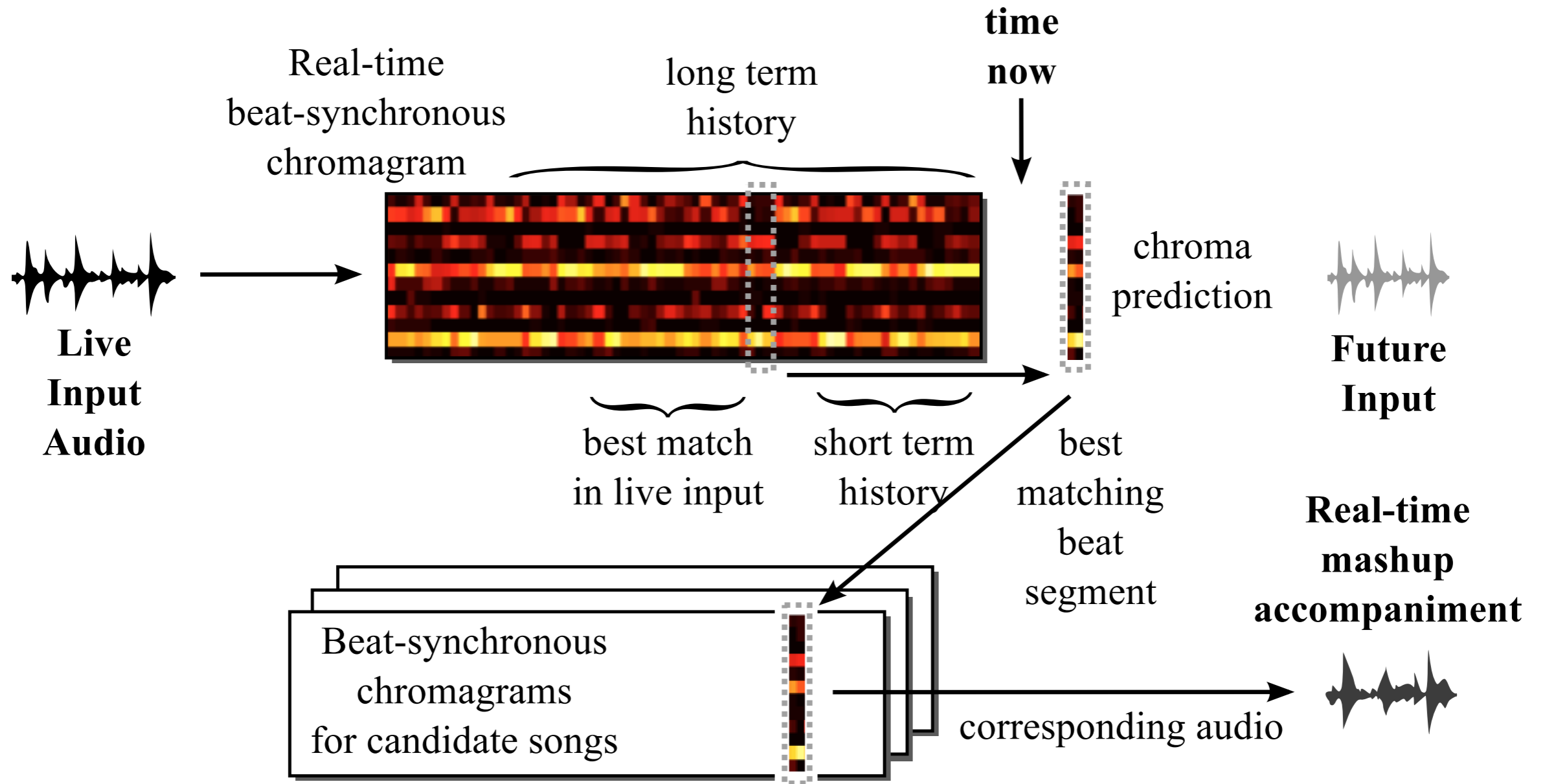
- Header:** "AutoMashUpper" in large blue text, followed by "by Matthew Davies, Philippe Hamel Kazuyoshi Yoshii and Masataka Goto (AIST)".
- Top Controls:** Buttons for "load song", "reset", "quit", and "save".
- Input Section:** "input - georgeduke movinon" with a blue waveform visualization. Below the waveform are buttons for "skip bwd", "play mix", "stop", and "skip fwd". A "segmentation level" slider is also present.
- Mashability Controls:** A panel with sliders for "key shift range" (+/- 3), "tempo range" (+/- 30%), "harmonic weight" (1), "rhythmic weight" (0.2), and "loudness weight" (0.2).
- Song Library:** A list of songs: "001. DaftPunk.AroundTheWorld", "002. DaftPunk.DaFunk", "003. DaftPunk.DigitalLove", "004. DaftPunk.GetLucky", and "005. MikiSantamaria.SlapBass". A "load new library" button is at the top right.
- Selected Songs:** A list of the same five songs, sorted by mashability. Below the list are buttons: "change to selection", "add to mashup", and "delete from mashup".
- Playback Balance:** A slider at the bottom labeled "playback balance" with "more mashup" on the left and "more input" on the right.
- Status:** "current section : 0 current beat : 19" at the bottom left.

Musician Mode

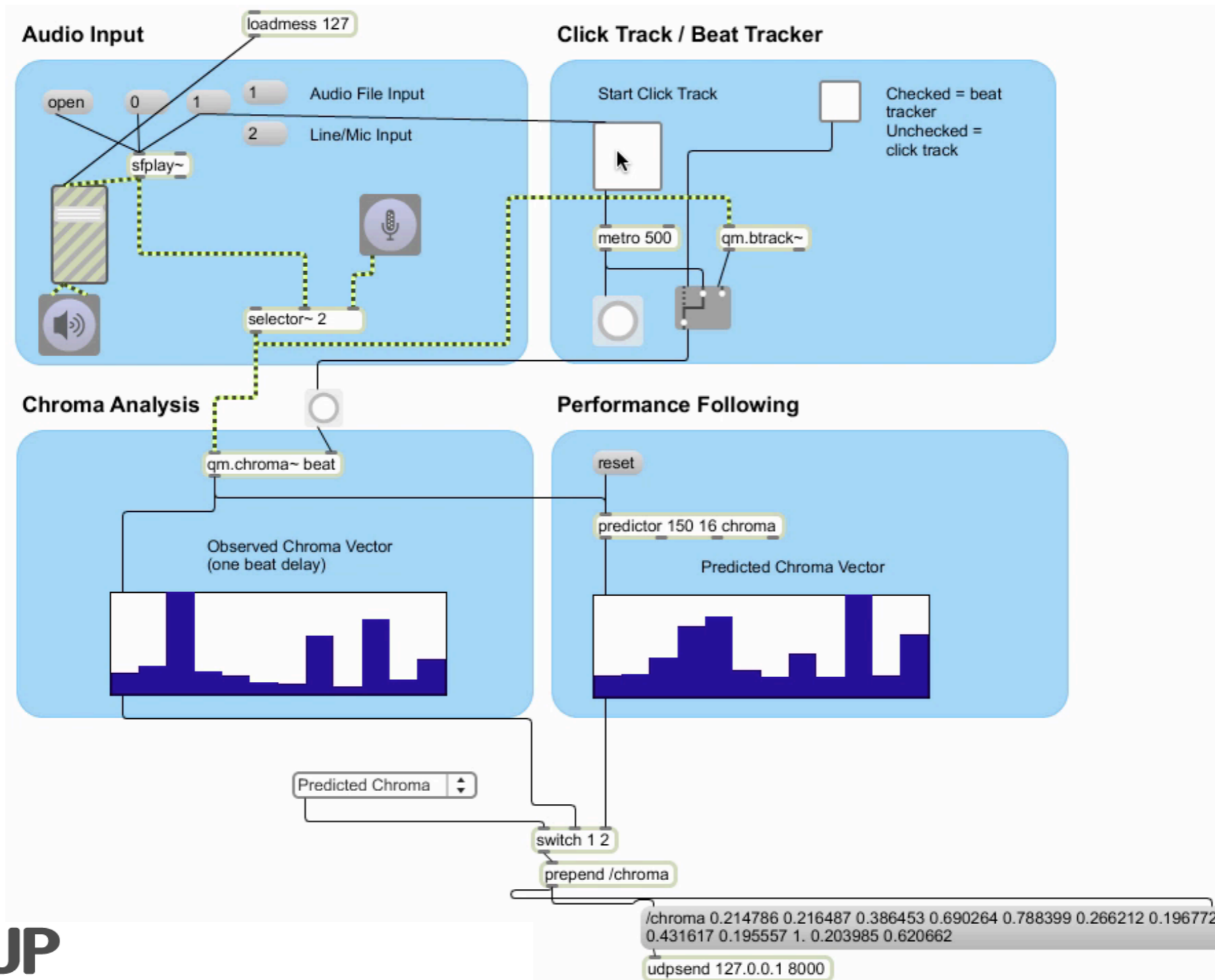
The screenshot shows the AutoMashUpper application window with the following components:

- Header:** "AutoMashUpper" logo and authors "by Matthew Davies, Philippe Hamel, Kazuyoshi Yoshii and Masataka Goto (AIST)".
- Buttons:** "load song", "reset", "quit", "save", "skip bwd", "play mix", "stop", "skip fwd".
- Input:** "input - georgeduke movinon" with a blue waveform visualization.
- Segmentation:** "segmentation level" control with a vertical slider.
- Mashup Visualizer:** A horizontal bar with red segments representing the mashup structure.
- Playback Balance:** "more mashup" and "more input" sliders.
- Current Status:** "current section : 0 current beat : 18".
- Mashability Controls:** Sliders for "key shift range +/- 2", "tempo range +/- 30%", "harmonic weight 1", "rhythmic weight 0.2", and "loudness weight 0.2".
- Selected Songs:** A list box containing "322. Miki Santamaria, SlapBass".
- Song Library:** A scrollable list of songs including "John Coltrane, Giant Steps", "John Mayer, Your Body is a Wonderland", etc.
- Actions:** "change to selection", "add to mashup", "delete from mashup", "select all", "pick 10", "clear".

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