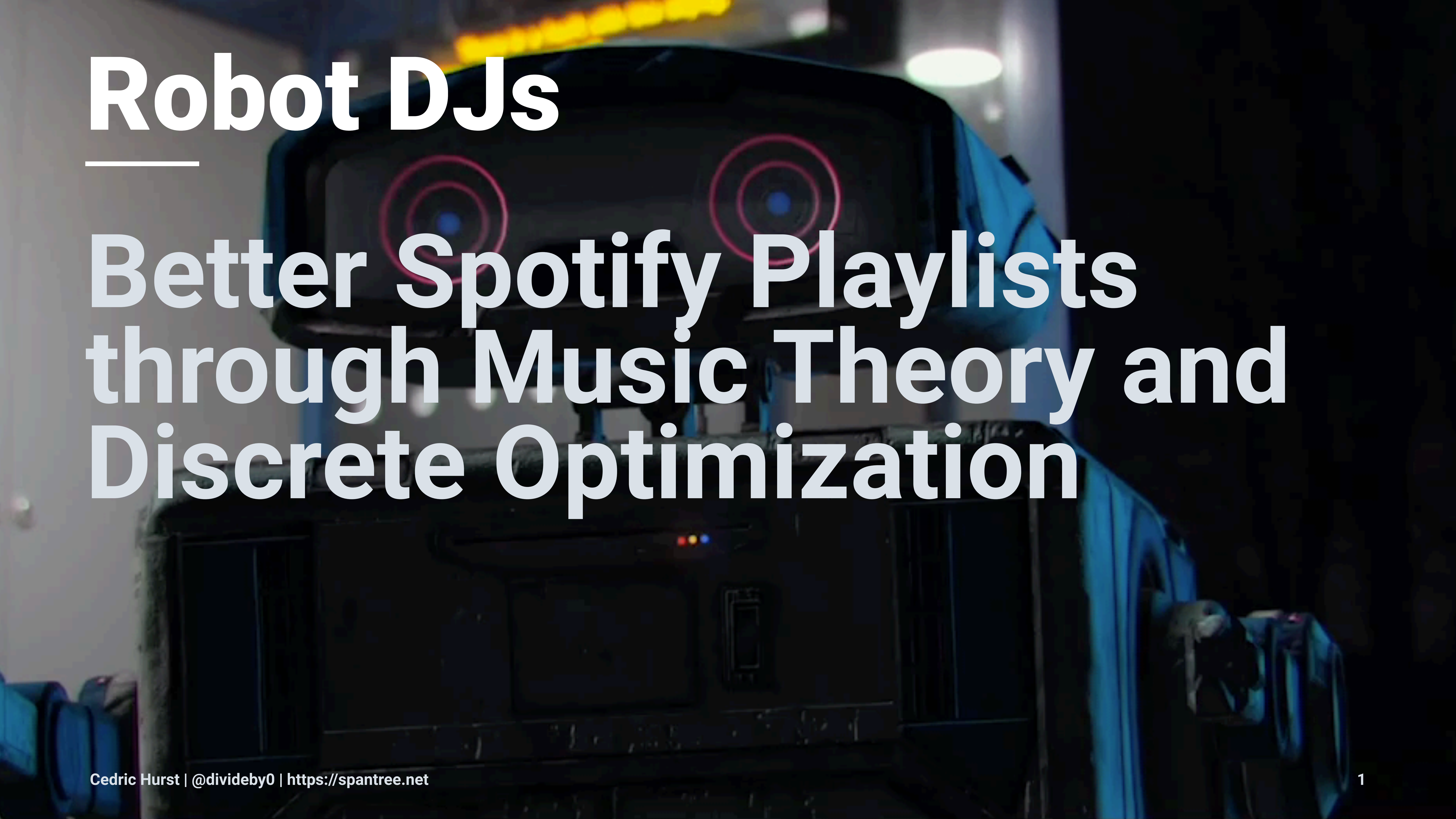


Robot DJs

Better Spotify Playlists through Music Theory and Discrete Optimization

Let's Try an Experiment...

More on that later.

whoami?

|> Consultant

|> Founder

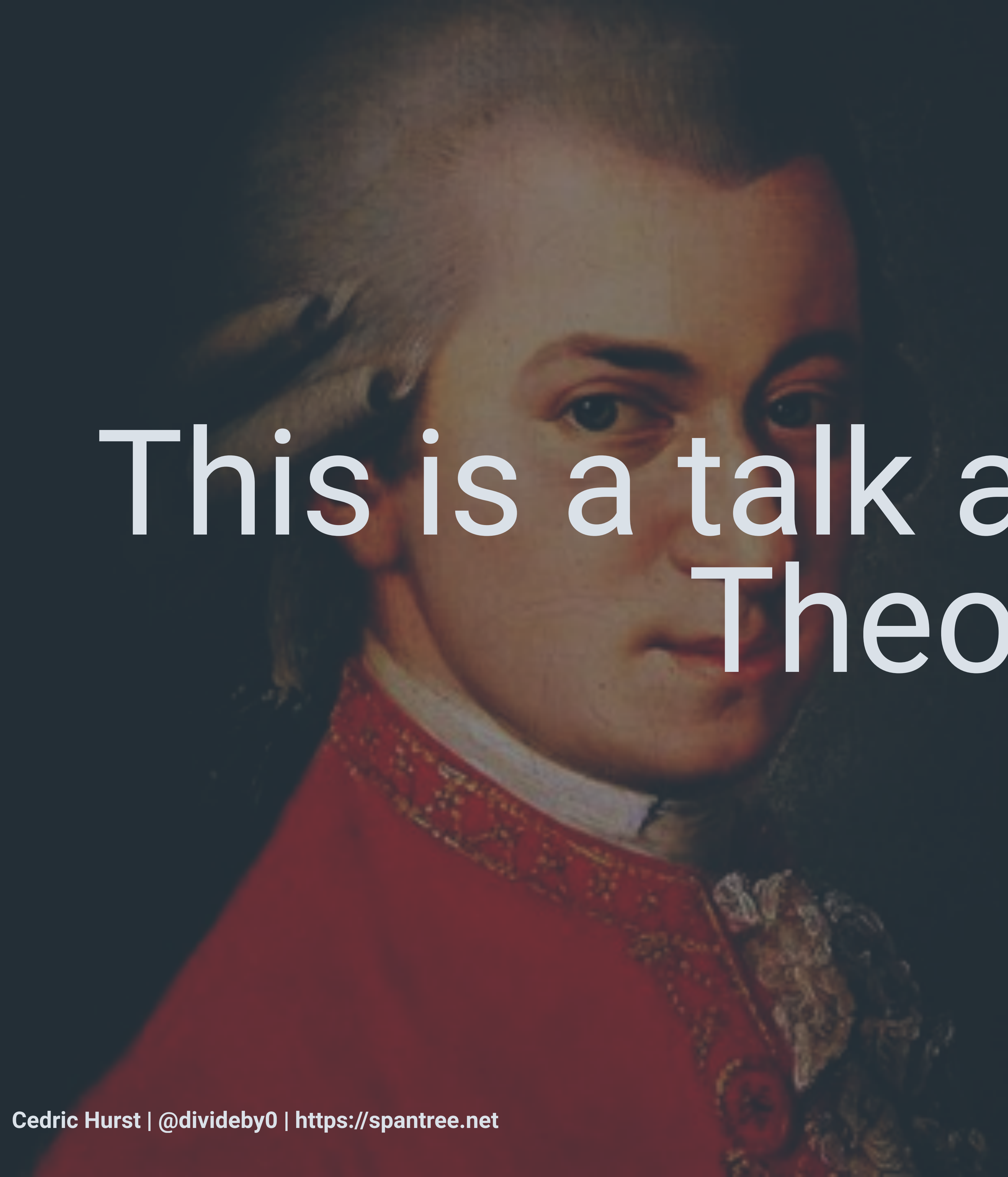
|> Speaker

|> (Co-)Author

Spantree

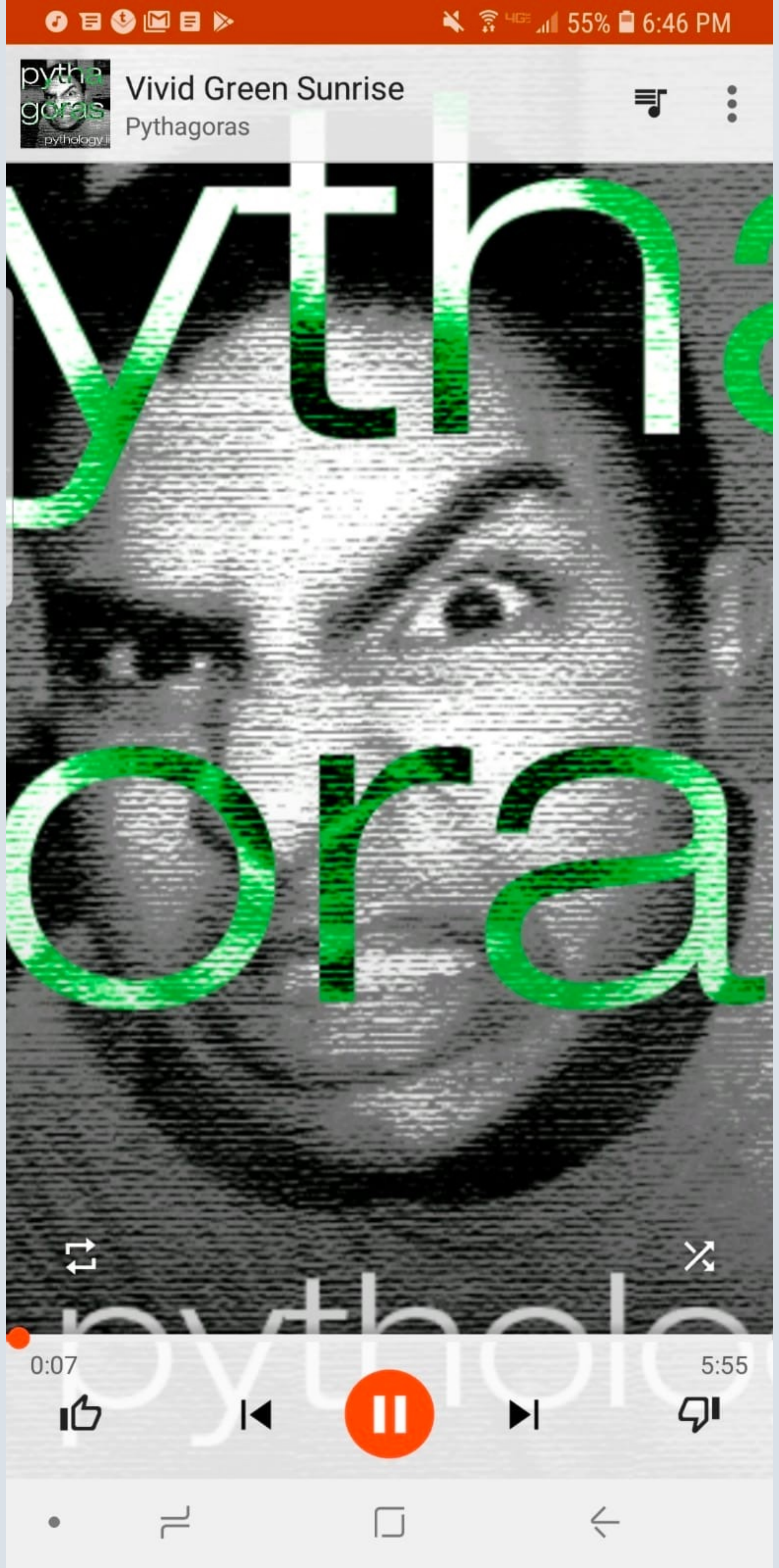
whoelseami

- |> Musician
- |> Composer
- |> Spotify Addict
- |> Recovering DJ



This is a talk about Music Theory.

sargogahtyP



55% 6:46 PM
Vivid Green Sunrise
Pythagoras
ytha
ora
pytholo
0:07
5:55

More on that later…

Remember when I said I was a recovering DJ?

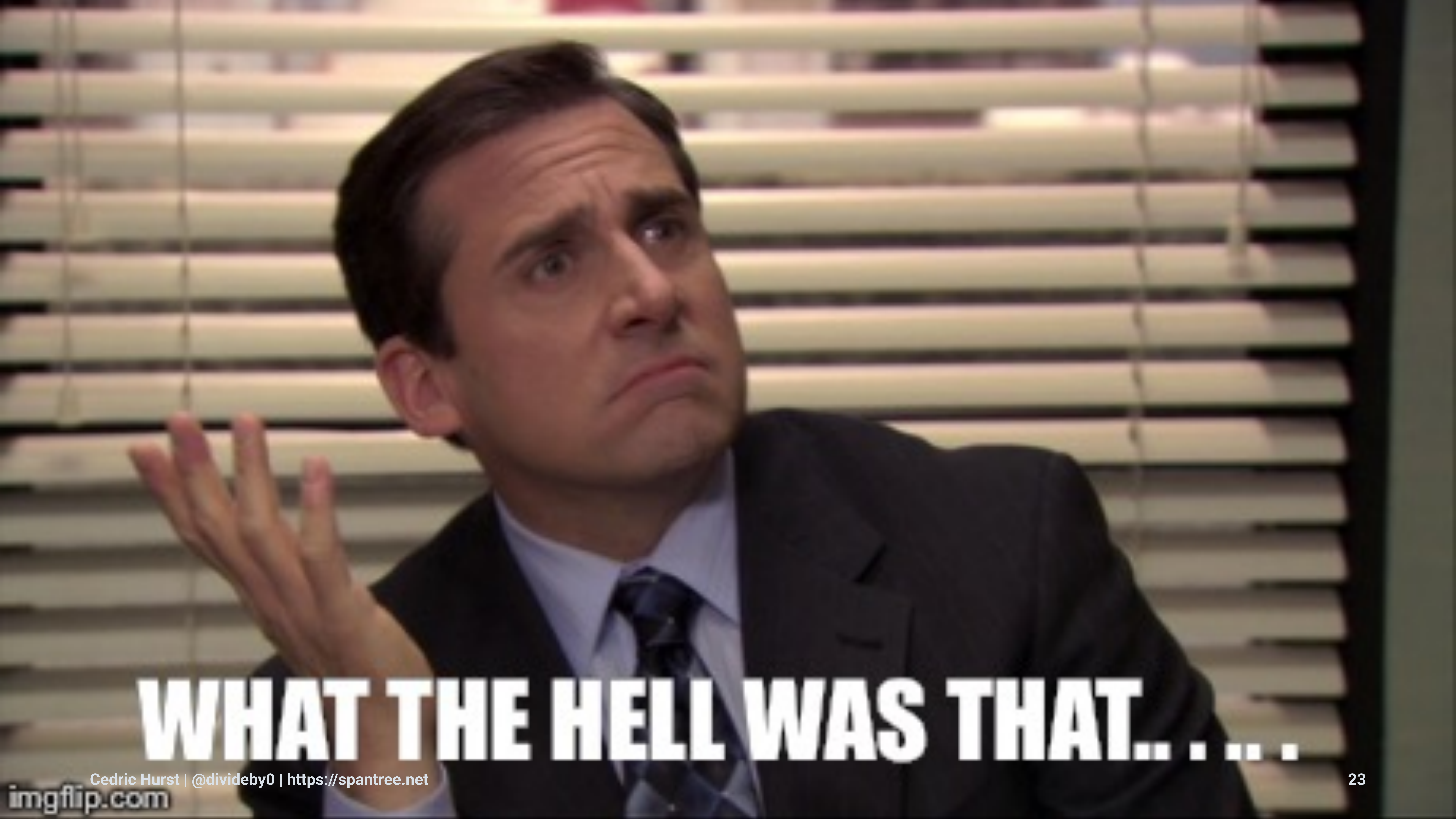
Let's Talk about College.

My Brief Career as a Classical Composer

My Slightly Longer Career as a MIDI Musician

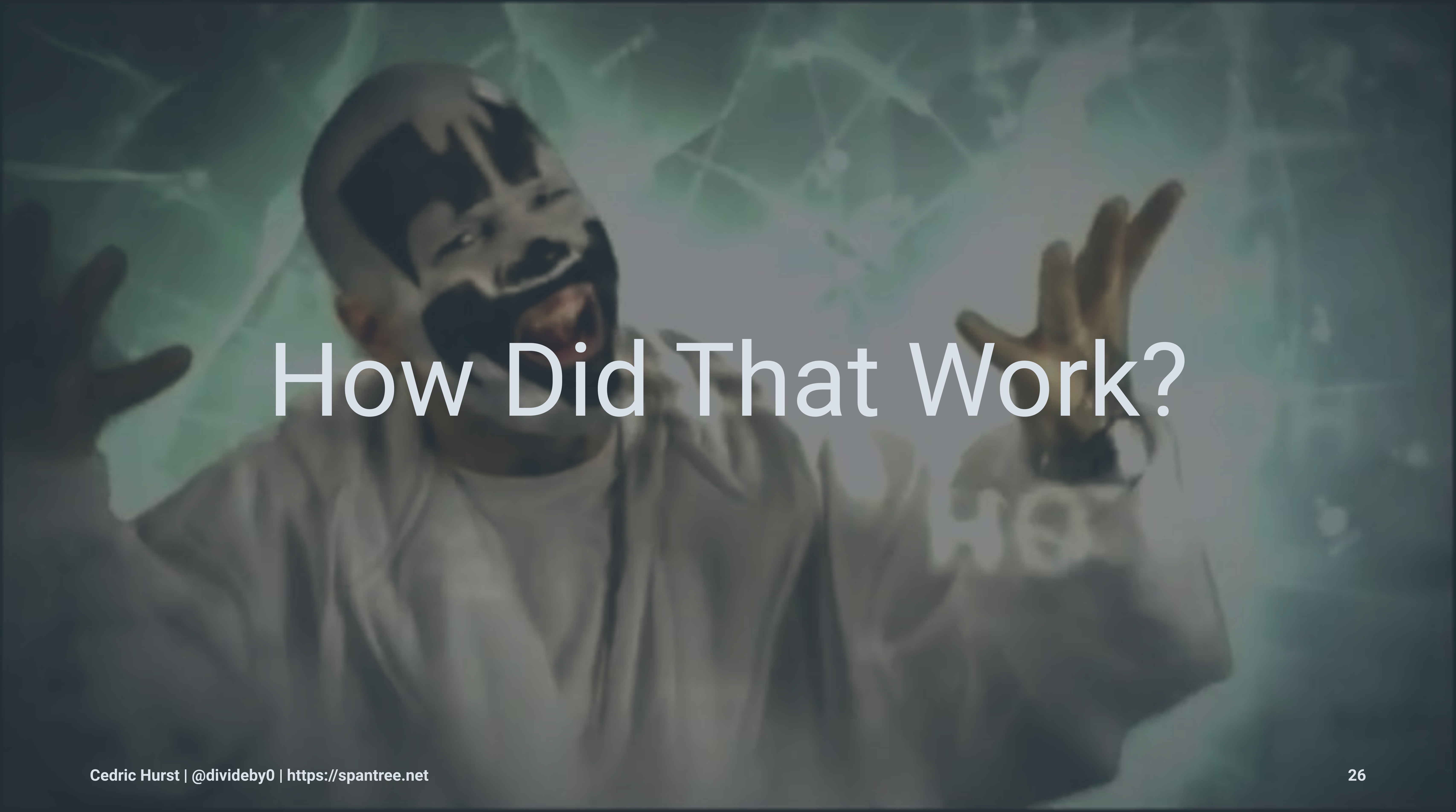
But then College Happened

DivideByZero - Beyonce Rockin' Beats
(circa 2003)


WHAT THE HELL WAS THAT.. . .. .
imgflip.com

Beyonce - Baby Boy

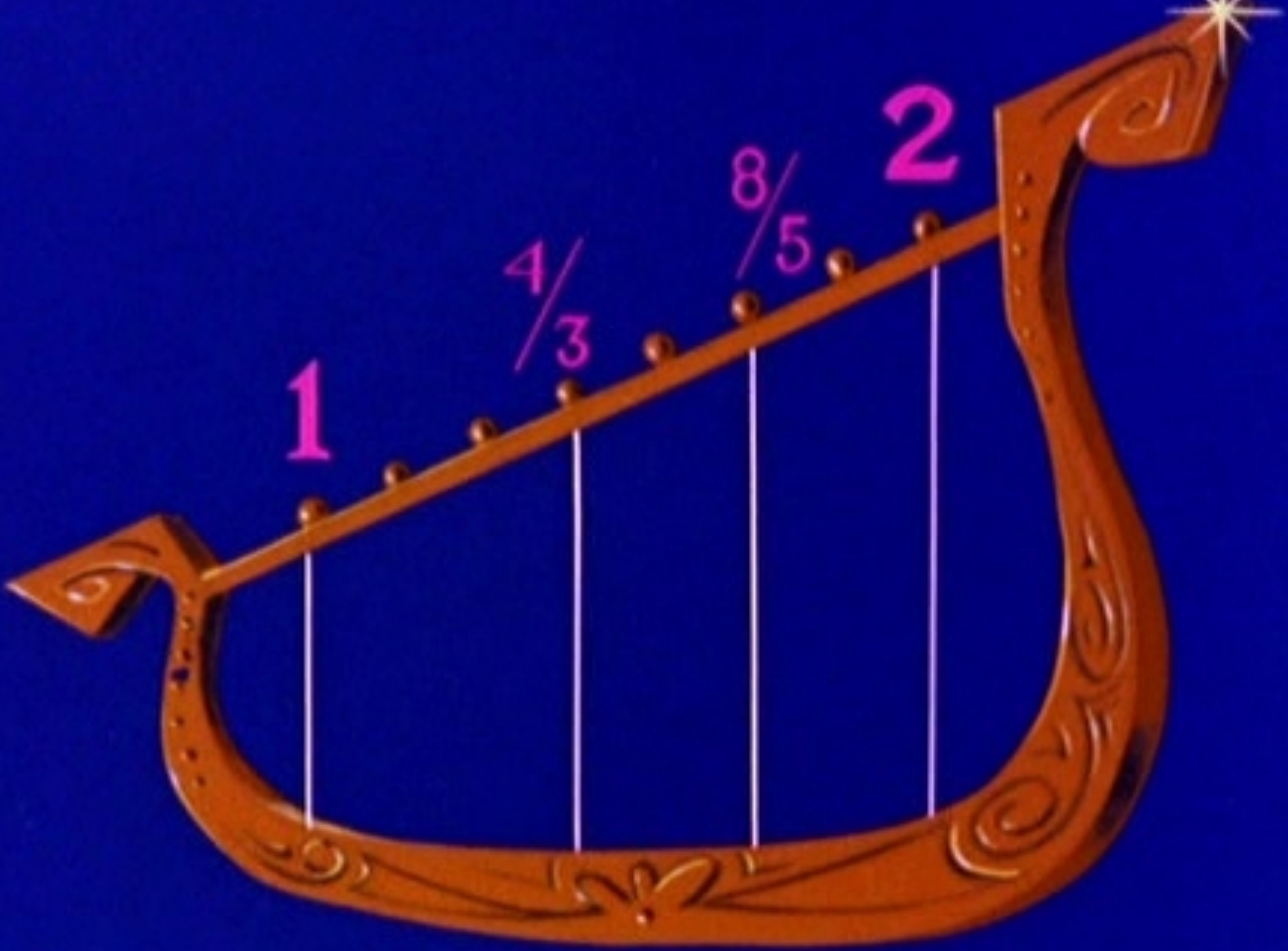
Chemical Brothers - Block Rockin' Beats



How Did That Work?

Harmonics

Remember Pythagoras?


1
4/3
8/5
2

Beyoncé, Sean Paul

Baby Boy (feat. Sean Paul)

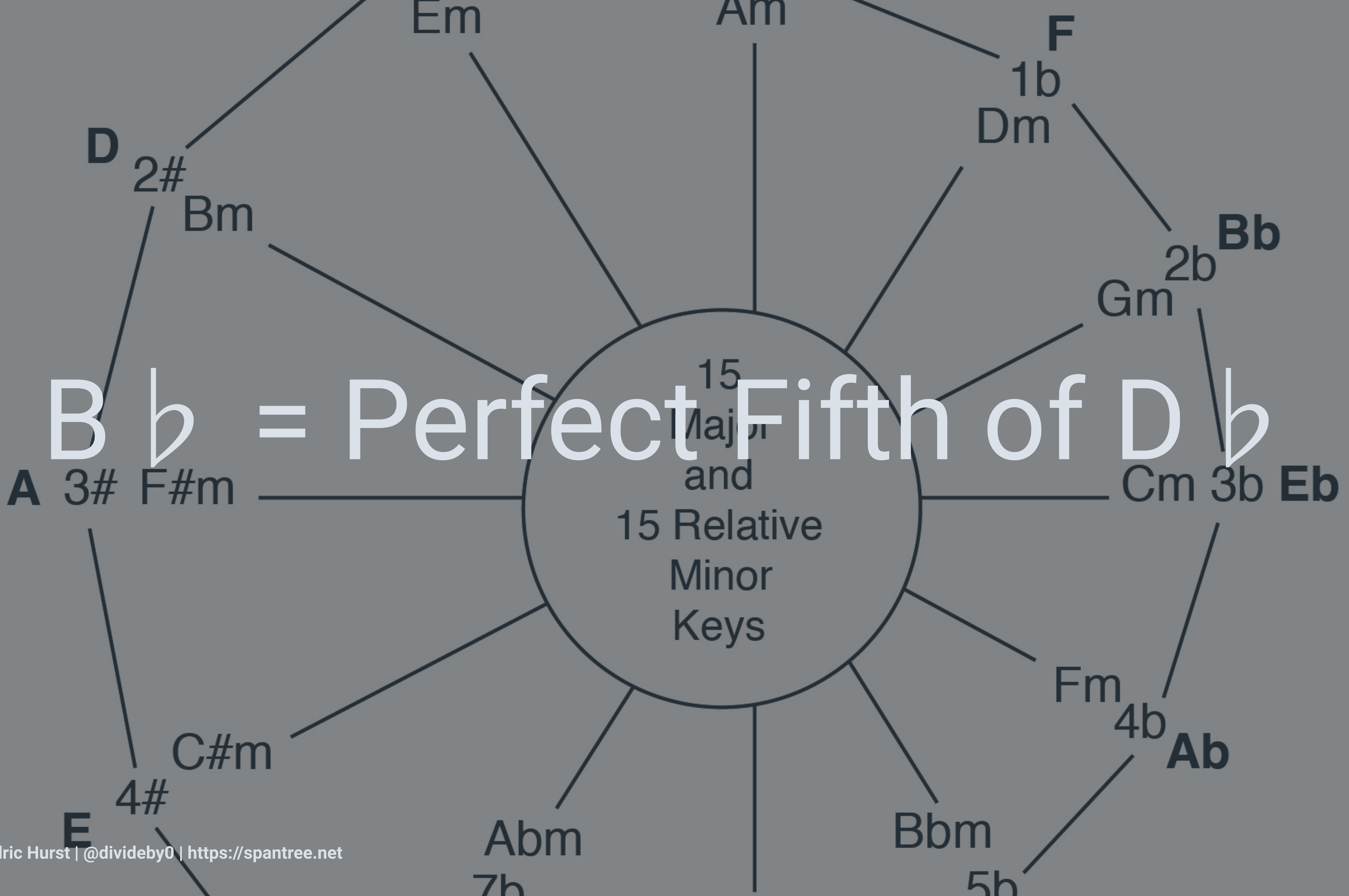
D ♭ Major	3B
Key	**Camelot**
4:04	91
Duration	**BPM**

The Chemical Brothers

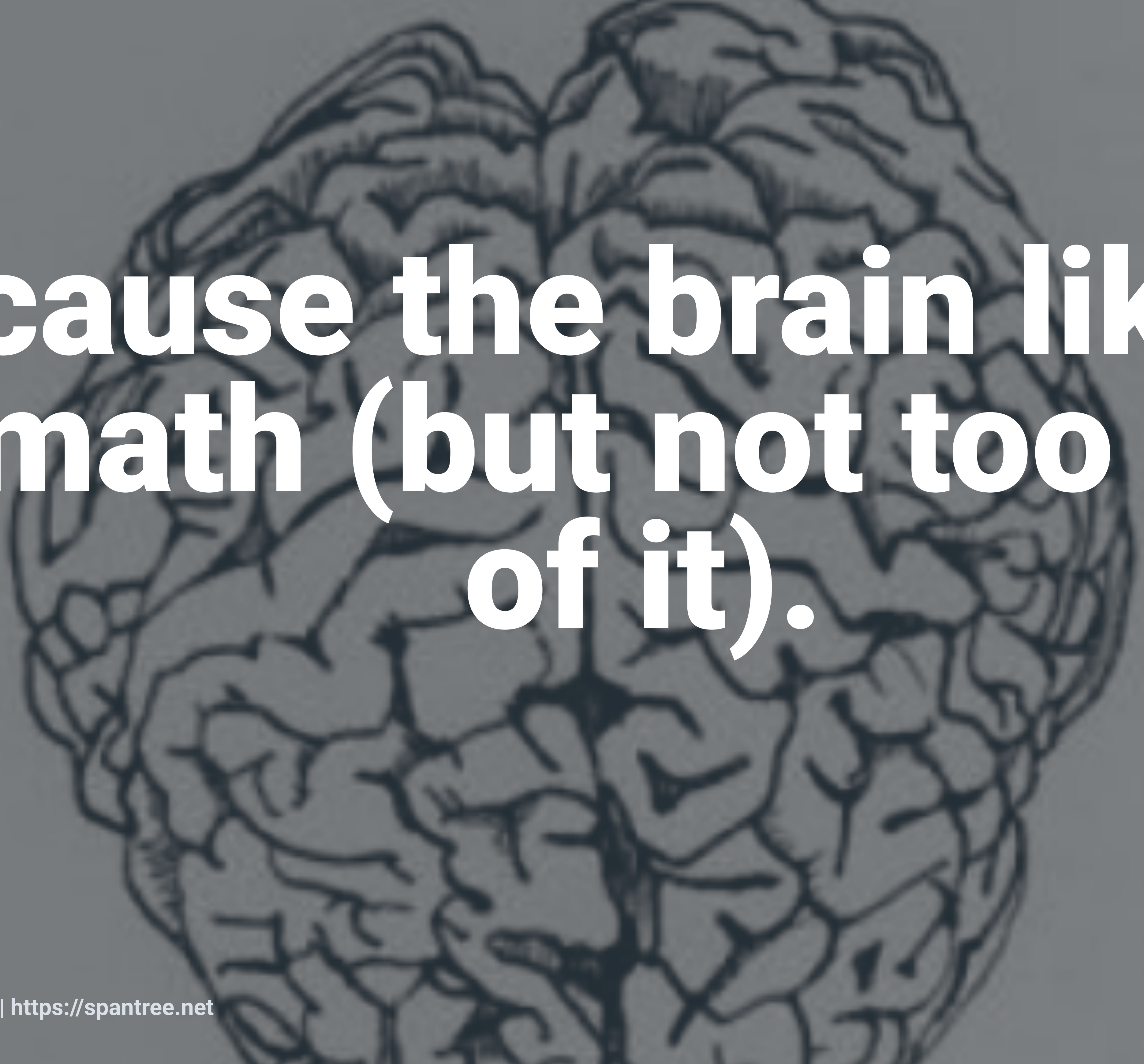
Block Rockin' Beats

B ♭ Minor	3A
Key	**Camelot**
5:14	109
Duration	**BPM**

B ♭ = Perfect Fifth of D ♭

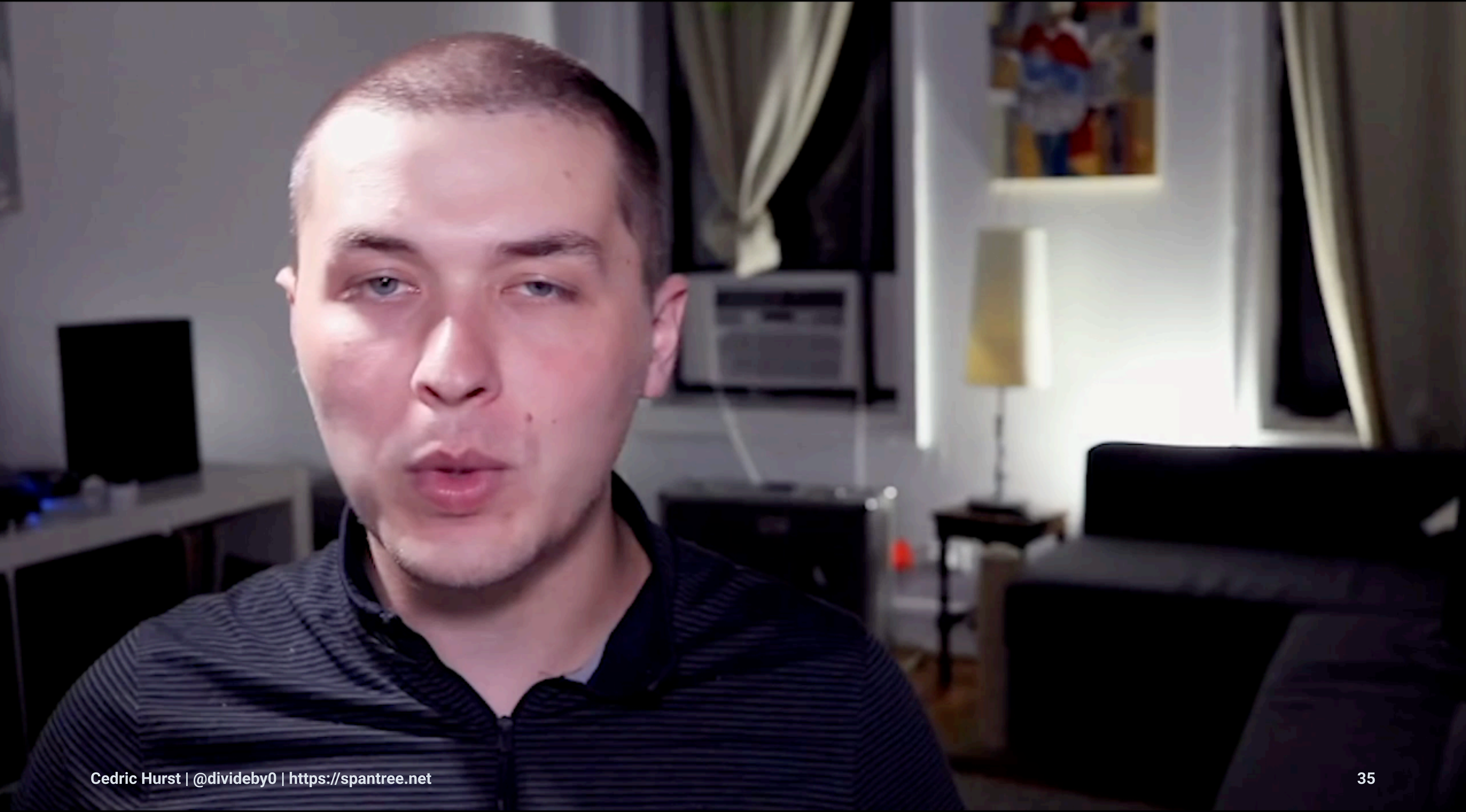


Why does this matter?



Because the brain likes to do math (but not too much of it).


Adam Neely




But what does that have to do with Beyoncé?


Interpreting linear expressions: diamonds
SAMSUNG

Beyoncé, Sean Paul

Baby Boy (feat. Sean Paul)

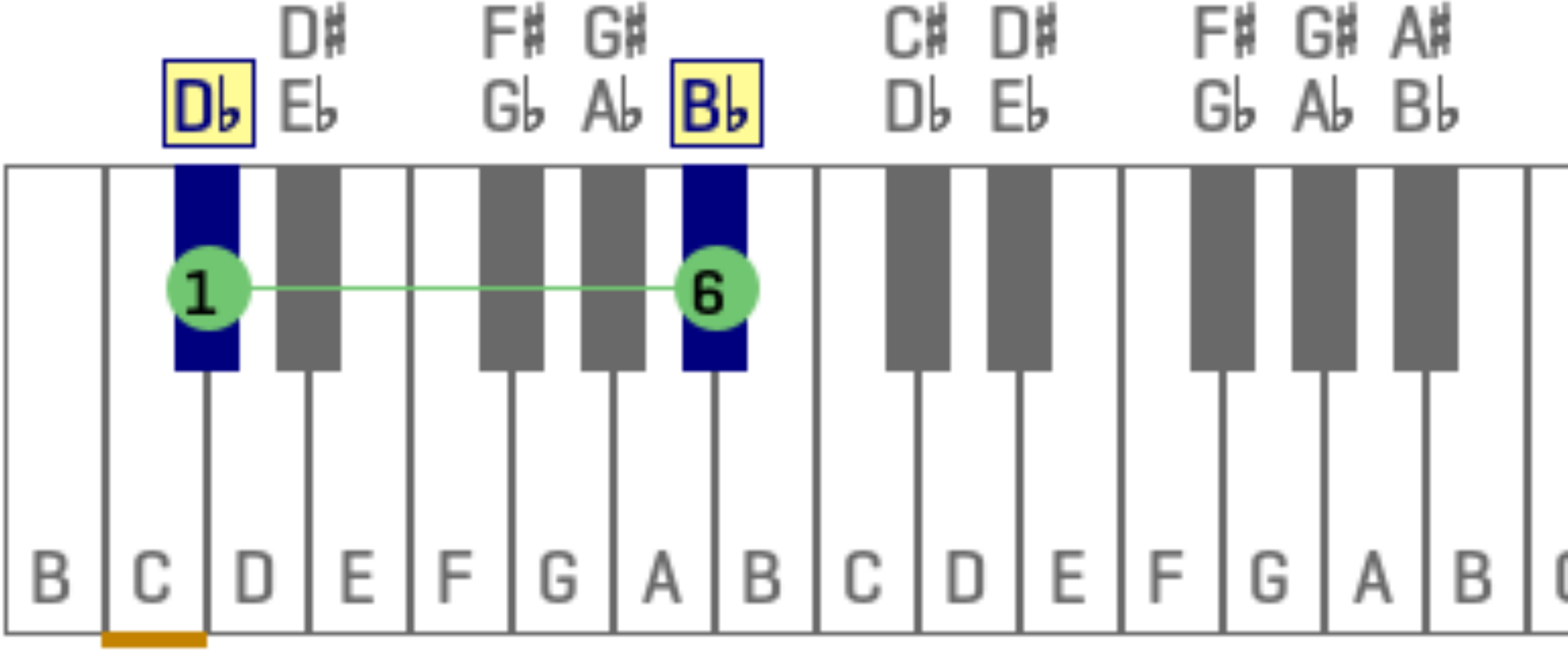
D♭ Major	3B
Key	Camelot
4:04	91
Duration	BPM

The Chemical Brothers

Block Rockin' Beats

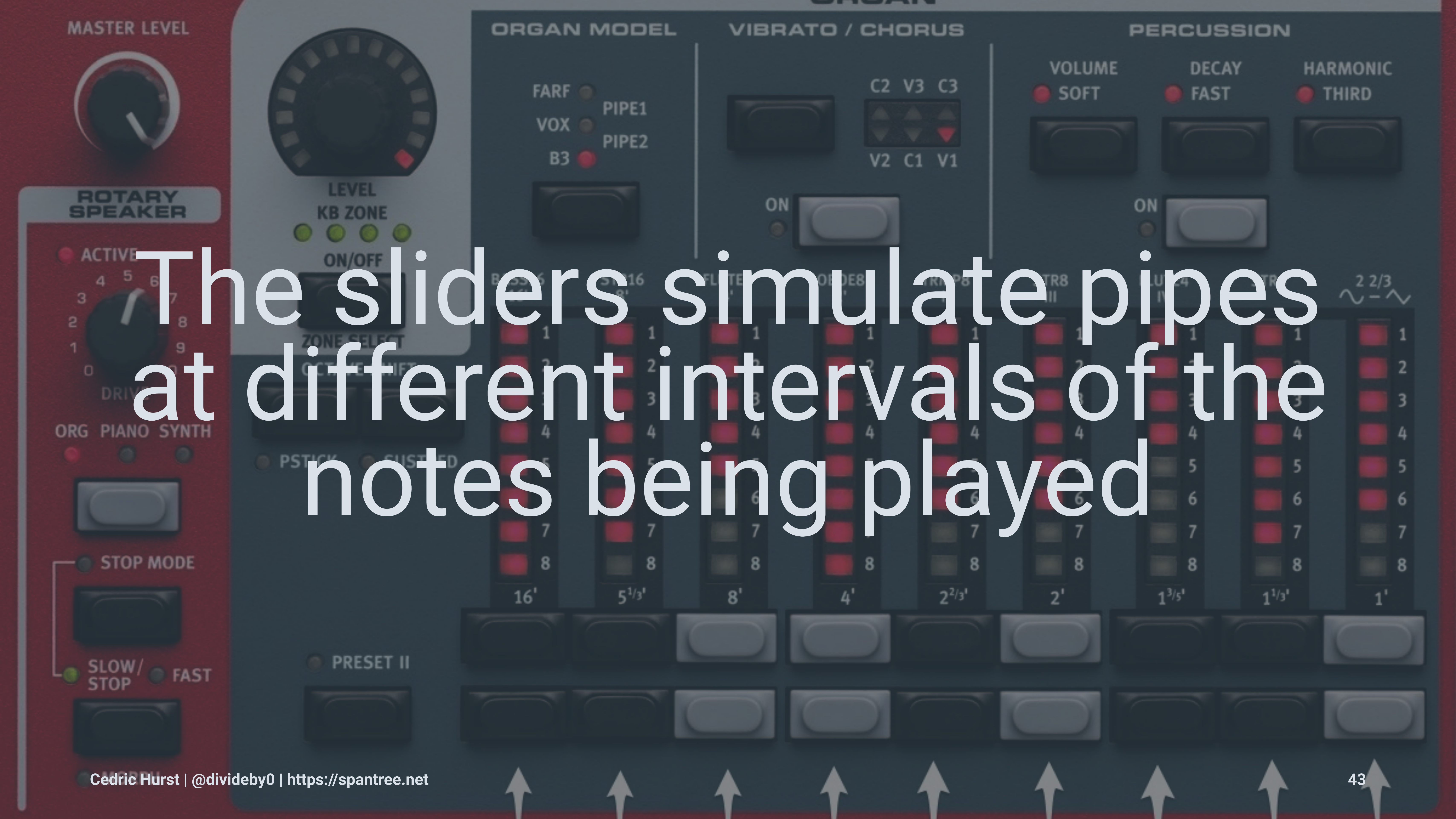
B♭ Minor	3A
Key	Camelot
5:14	109
Duration	BPM

Relative minor of D-flat major is B-flat



Let's See This on the Keyboard

While We're Over There... Remember that Experiment We Did at the Beginning of the Talk?



The sliders simulate pipes at different intervals of the notes being played

This is a Talk about Playlists.

What matters (musically) to a playlist?

|> Tone

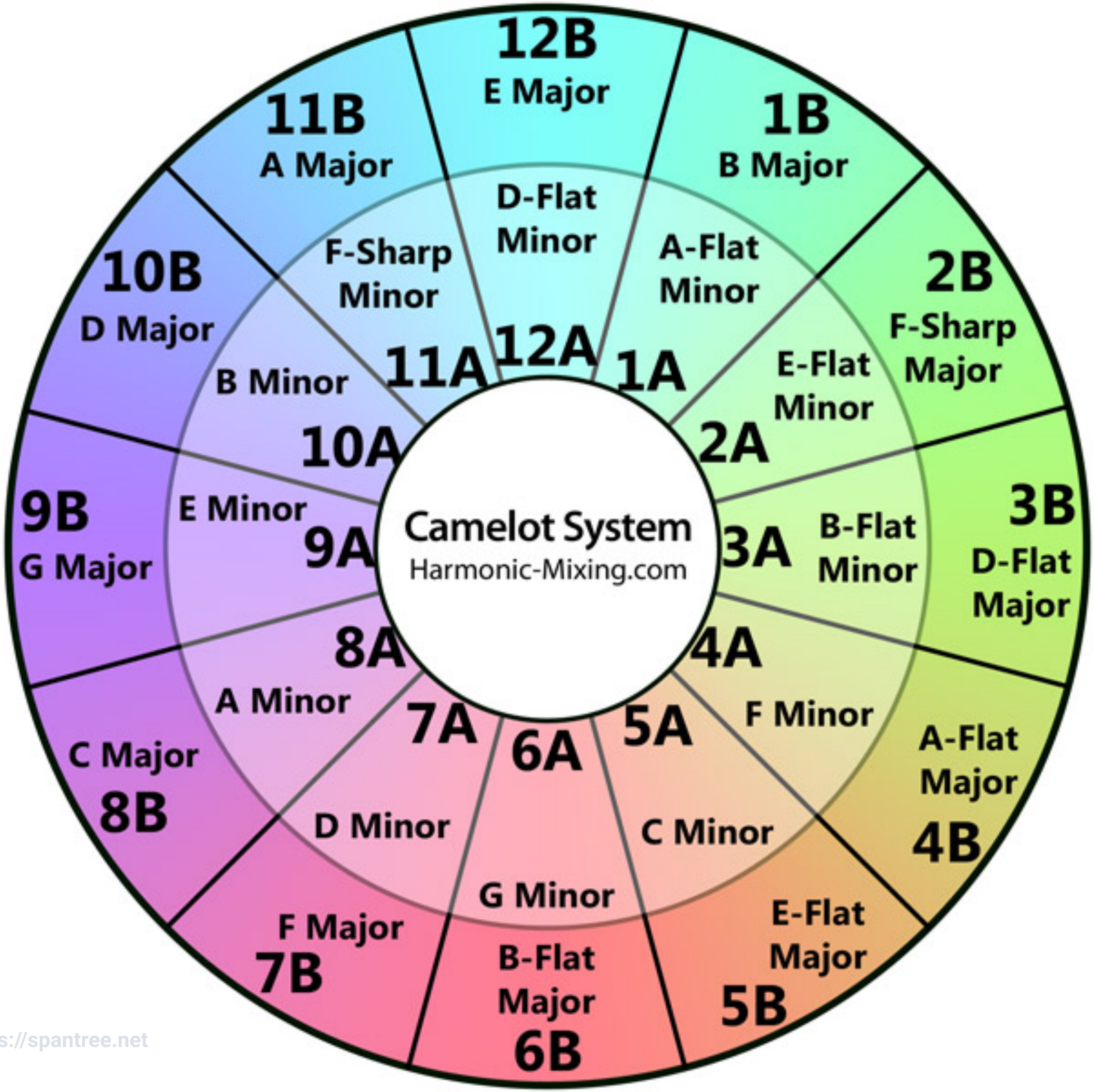
|> Timimg

|> Timbre

Let's talk about tone.

But before we do that, let's talk about modes.

Jacob Collier


12B
E Major
1B
B Major
2B
F-Sharp Major
3B
D-Flat Major
4B
A-Flat Major
5B
E-Flat Major
6B
B-Flat Major
7B
F Major
8B
C Major
9B
G Major
10B
D Major
11B
A Major
12A
D-Flat Minor
1A
A-Flat Minor
2A
E-Flat Minor
3A
B-Flat Minor
4A
F Minor
5A
C Minor
6A
G Minor
7A
D Minor
8A
A Minor
9A
E Minor
10A
B Minor
11A
F-Sharp Minor
Camelot System
Harmonic-Mixing.com

But almost no one **owns** their media anymore.

Today, almost everyone listens to their music in the **cloud**.

This is good news for playlists because they're **pervasive** now.

People are even writing Ph.D theses about it.

JUST PRESS PLAY: THE ROLE OF PLAYLISTS IN DIGITAL AGE MUSIC CONSUMPTION AND DISTRIBUTION

by

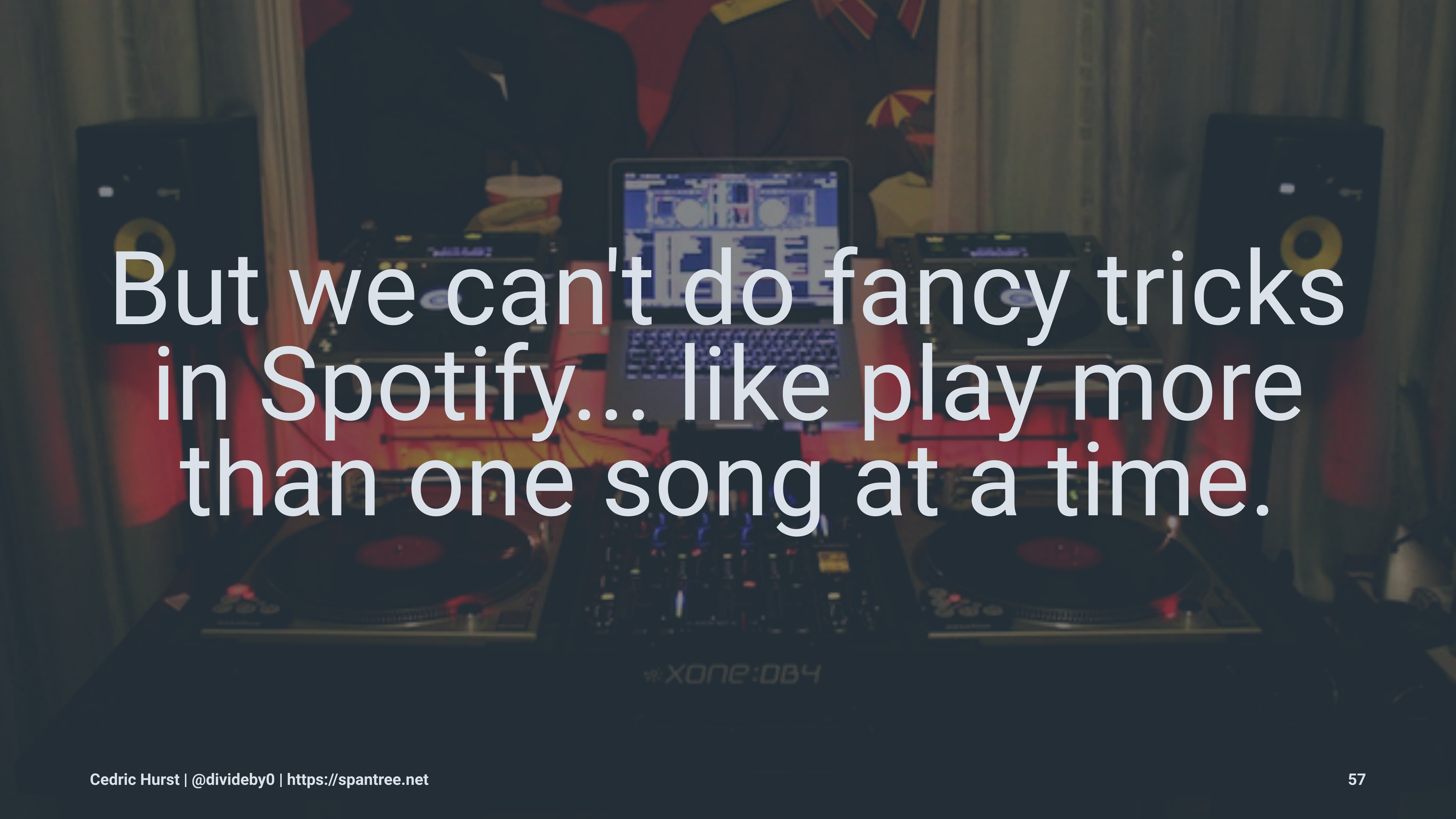
Aidan D. Epstein

Bachelor of Science in Economics, University of Pittsburgh, 2016

Submitted to the Graduate Faculty of

the Kenneth P. Dietrich School of Arts & Sciences in partial fulfillment

of the requirements for the degree of
Master of Arts



But we can't do fancy tricks in Spotify... like play more than one song at a time.

So what can we do?

We can reorder the tracks!

The New York Times

Annoyed by Restaurant Playlists, a Master Musician Made His Own

How Ryuichi Sakamoto assembled the soundtrack for Kajitsu, in Murray Hill, and what it says about the sounds we hear (or should) while we eat.

This is a talk about Discrete Optimization.

Not logged in Talk Contributions Create account Log in

Article Talk Read Edit View history Search Wikipedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction

Help
About Wikipedia
Community portal
Recent changes
Contact page

Optimization problem

In mathematics and computer science, an **optimization problem** is the problem of finding the *best* solution from all feasible solutions. Optimization problems can be divided into two categories depending on whether the variables are continuous or discrete. An optimization problem with discrete variables is known as a discrete optimization. In a discrete optimization problem, we are looking for an object such as an integer, permutation or graph from a countable set. Problems with continuous variables include constrained problems and multimodal problems.

Combinatorial optimization problem [edit]

Main article: Combinatorial optimization

Formally, a combinatorial optimization problem A is a quadruple[citation needed] (I, f, m, g), where

- I is a set of instances;
- given an instance $x \in I$, $f(x)$ is the set of feasible solutions;
- given an instance x and a feasible solution y of x, $m(x, y)$ denotes the measure of y, which is usually a positive real.
- g is the goal function, and is either $\min$ or $\max$.

TL;DR

Discrete

Having a finite number of possibile solutions.

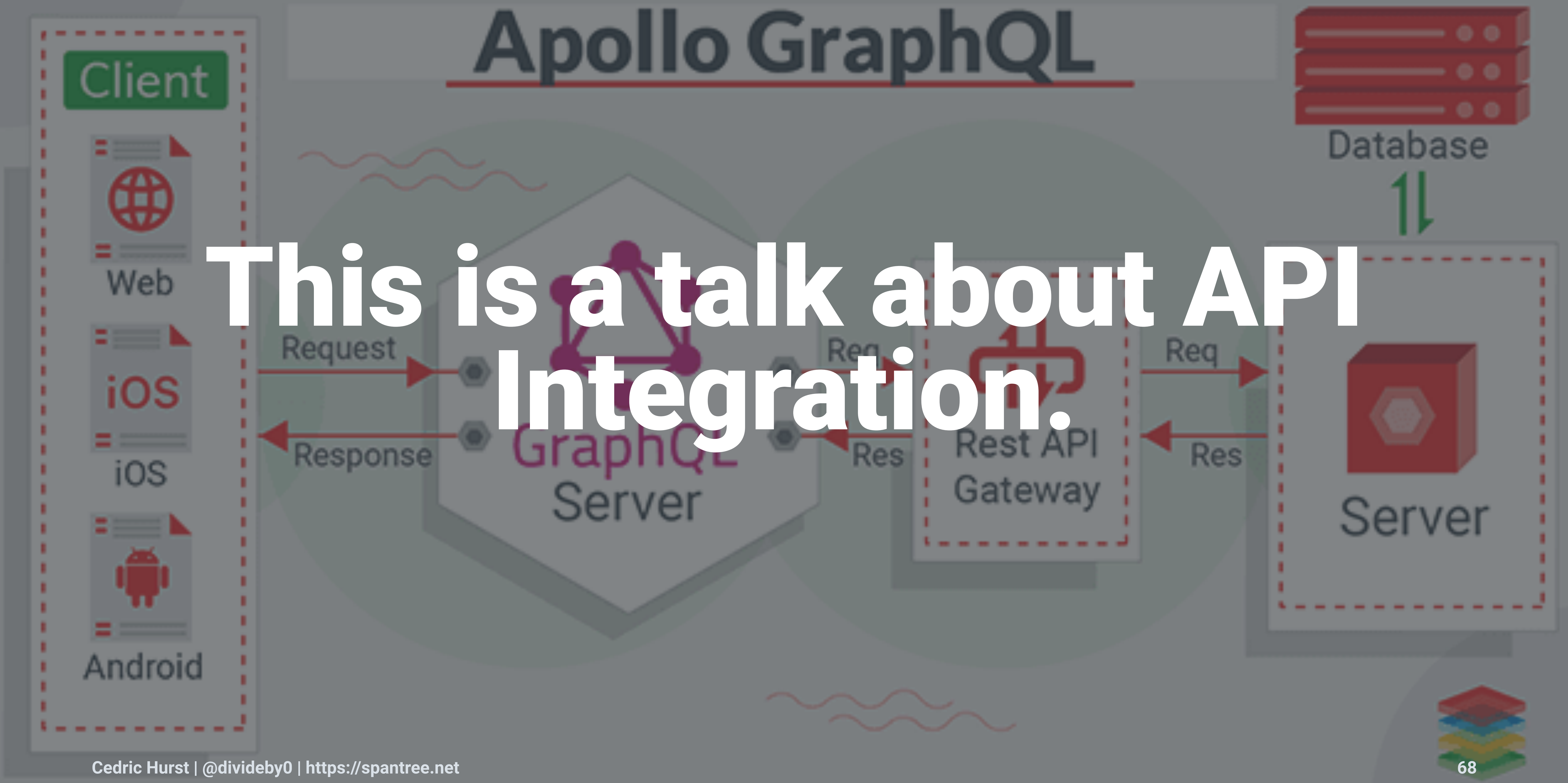
Optimization

Searching those options to find one that closely matches a set of rules.

So what are our rules?

- |> Songs should play exactly once.
- |> Some song transitions should be to songs in the same key.
- |> Other songs transitions should be to adjacent keys on the Camelot Scale.
- |> Songs transitions should avoid drastic tempo or timbre changes.

But the songs are all in the cloud.



This is a talk about API Integration.

Spotify for Developers

DISCOVER DOCS CONSOLE COMMUNITY DASHBOARD USE CASES

WEB API

QUICK START GUIDES LIBRARIES REFERENCE

REFERENCE

- Reference Index
- Search API
- Browse API
- Follow API
- Playlists API
- Library API
- Artists API
- Player API
- Personalization API
- User Profile API
- Albums API
- Tracks API
- Objects Index

Web API Reference BETA

Welcome to the improved reference for the Spotify Web API (beta). **Note this may have some missing information**, even though we try to keep this as accurate as possible.

Have feedback? Let us know on Twitter!

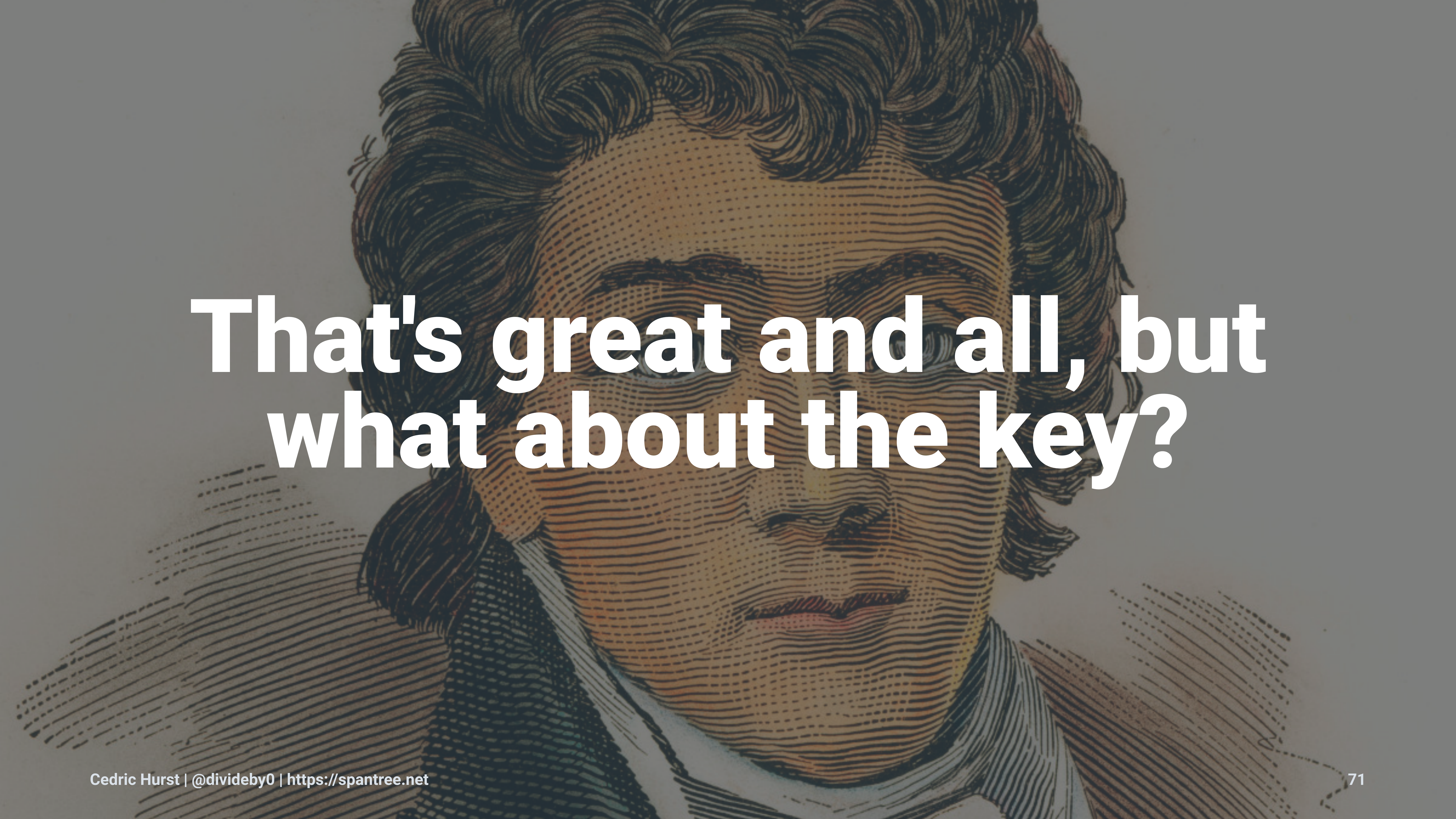
Reference Index

Browse API

- Get All Categories
- Get a Category
- Get a Category's Playlists
- Get Recommendations

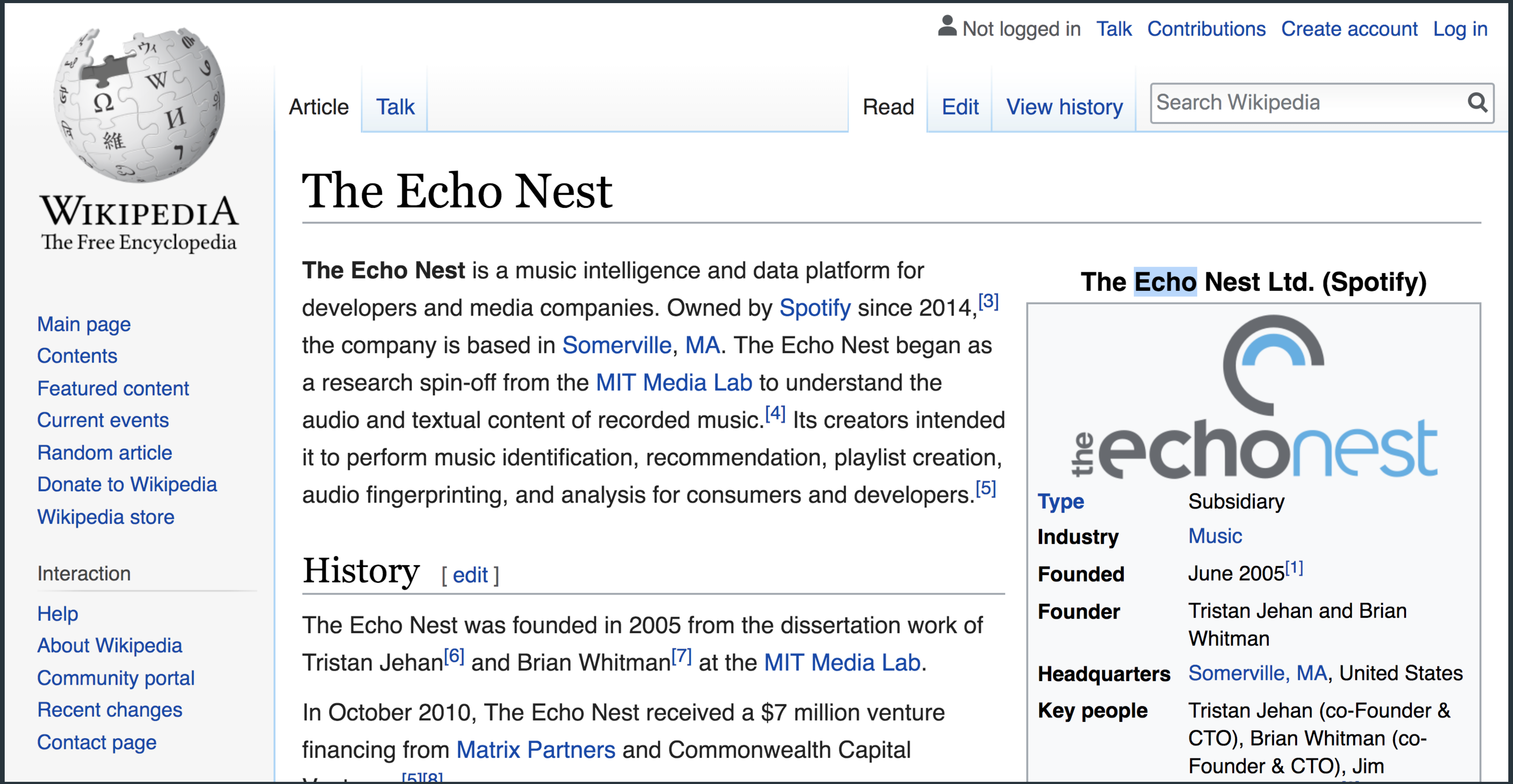
Let's Get a Spotify Playlist.

```
{
  "description": "Having friends over for dinner? Here's the perfect playlist.",
  "tracks": {
    "items": [
      {
        "track": {
          "album": { "name": "Untamed", "release_date": "2015-12-11" },
          "artists": [{ "name": "Cam" }],
          "name": "Burning House",
          "popularity": 64,
        }
      },
      ...
    ]
  }
}
```



That's great and all, but what about the key?

Spotify M&A Team to the Rescue



Not logged in Talk Contributions Create account Log in

Article Talk Read Edit View history Search Wikipedia

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction

Help
About Wikipedia
Community portal
Recent changes
Contact page

The Echo Nest

The Echo Nest is a music intelligence and data platform for developers and media companies. Owned by Spotify since 2014,[3] the company is based in Somerville, MA. The Echo Nest began as a research spin-off from the MIT Media Lab to understand the audio and textual content of recorded music.[4] Its creators intended it to perform music identification, recommendation, playlist creation, audio fingerprinting, and analysis for consumers and developers.[5]

The Echo Nest Ltd. (Spotify)

Type	Subsidiary
Industry	Music
Founded	June 2005[1]
Founder	Tristan Jehan and Brian Whitman
Headquarters	Somerville, MA, United States
Key people	Tristan Jehan (co-Founder & CTO), Brian Whitman (co-Founder & CTO), Jim

History [edit]

The Echo Nest was founded in 2005 from the dissertation work of Tristan Jehan[6] and Brian Whitman[7] at the MIT Media Lab.

In October 2010, The Echo Nest received a $7 million venture financing from Matrix Partners and Commonwealth Capital

Turns out we can get "Audio Features" now too.


```json
{
  "type": "audio_features",
  "id": "11dFghVXANMlKmJXsNCbNl",
  "key": 2,
  "mode": 1,
  "tempo": 114.944,
  "time_signature": 4,
  "loudness": -2.743,
  "danceability": 0.696,
  "energy": 0.905,
  "speechiness": 0.103,
  "acousticness": 0.011,
  "instrumentalness": 0.000905,
  "liveness": 0.302,
  "valence": 0.625
}
```

Key/Mode = D Major


```
{
  "type": "audio_features",
  "id": "11dFghVXANMlKmJXsNCbNl",
  "key": 2,
  "mode": 1,
  "tempo": 114.944,
  "time_signature": 4,
  "loudness": -2.743,
  "danceability": 0.696,
  "energy": 0.905,
  "speechiness": 0.103,
  "acousticness": 0.011,
  "instrumentalness": 0.000905,
  "liveness": 0.302,
  "valence": 0.625
}
```

We also get some other helpful stuff for Optimization.


```
{
  "type": "audio_features",
  "id": "11dFghVXANMlKmJXsNCbNl",
  "key": 2,
  "mode": 1,
  "tempo": 114.944,
  "time_signature": 4,
  "loudness": -2.743,
  "danceability": 0.696,
  "energy": 0.905,
  "speechiness": 0.103,
  "acousticness": 0.011,
  "instrumentalness": 0.000905,
  "liveness": 0.302,
  "valence": 0.625
}
```

Ok, so we have the data. But how do we optimize?

OptaPlanner Home Download Learn Get Help Source Team Services Follow

What is OptaPlanner?

OptaPlanner is **a constraint solver.** It optimizes business resource planning use cases, such as Vehicle Routing, Employee Rostering, Cloud Optimization, Task Assignment, Conference Scheduling, Job Scheduling, Bin Packing and many more. Every organization faces such scheduling puzzles: assign a limited set of *constrained* resources (employees, assets, time and money) to provide products or services. OptaPlanner delivers more efficient plans to improve service quality and reduce costs.

OptaPlanner is **a lightweight, embeddable planning engine**. It enables normal Java™ programmers to solve optimization problems efficiently. It is also compatible with other JVM languages (such as Kotlin and Scala). Constraints apply on plain domain objects and can reuse existing code. There's no need to input them as mathematical equations. Under the hood, OptaPlanner combines sophisticated optimization heuristics and metaheuristics (such as Tabu Search, Simulated Annealing and Late Acceptance) with very efficient score calculation.

OptaPlanner is **open source software**, released under the Apache Software License. It is written in 100% pure Java™, runs on any JVM and is available in the Maven Central repository too.

Download OptaPlanner
7.20.0.Final

Try the examples now:

1. Download the zip and unzip it
2. On Linux/Mac, run `examples/runExamples.sh`
   On Windows, run `examples/runExamples.bat`

Requires Java™ to run.

Read documentation
7.20.0.Final

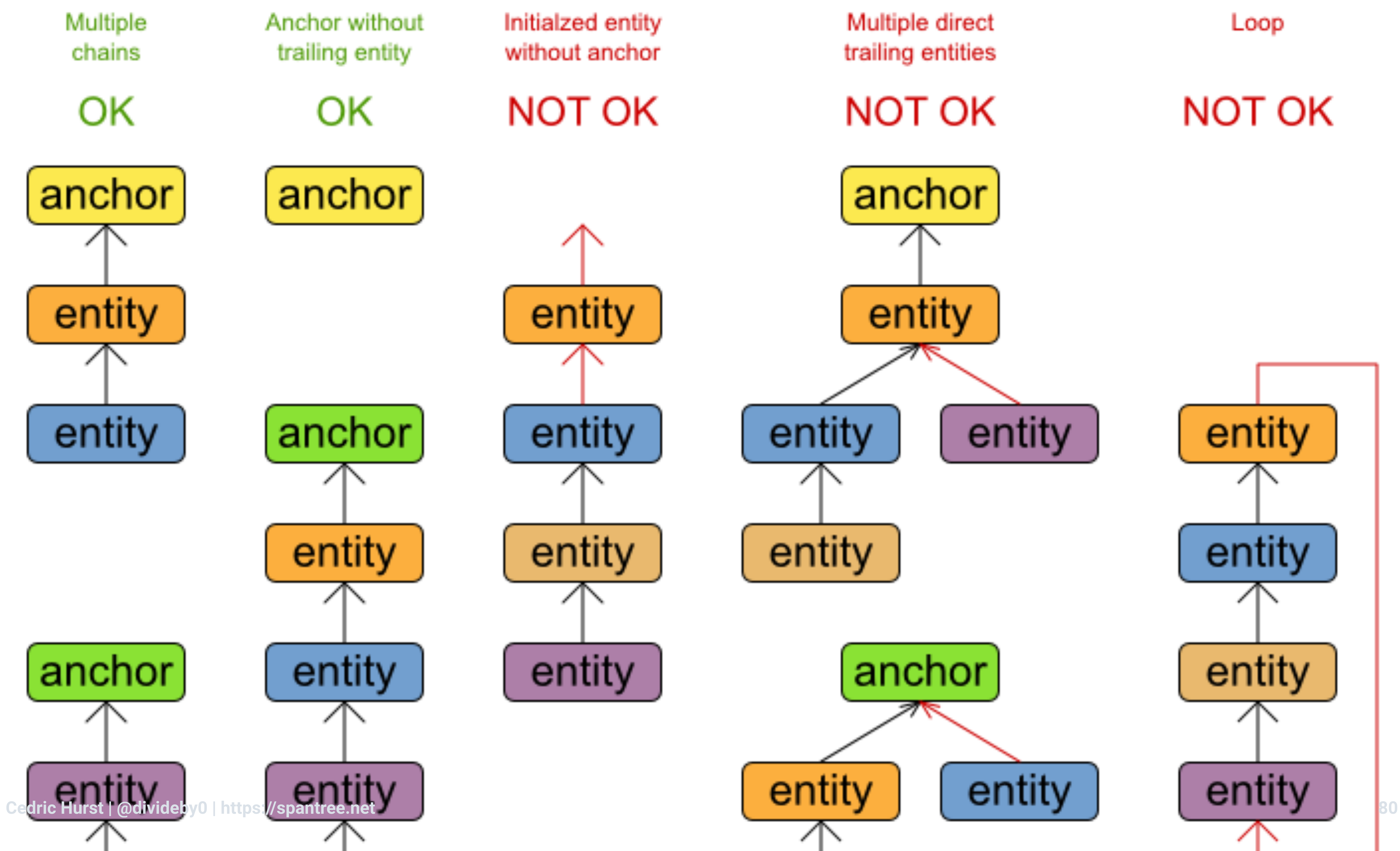
Read the Quick Start chapter.

Let's model our Domain

```kotlin
data class Track(
  @Json(name = "album") var album: Album,
  @Json(name = "artists") var artists: List<Artist>,
  @Json(name = "duration_ms") val durationMs: Int,
  @Json(name = "explicit") val explicit: Boolean,
  @Json(name = "features") val features: AudioFeatures,
  @Json(name = "id") val id: String,
  @Json(name = "name") val name: String,
  ...
)

interface PlaylistTrack {
    @Json(name = "track")
    var track: Track?
}
```

Now let's define an PlanningEntities and Anchors

```
data class FirstPlaylistTrack(
    @Json(name = "track") override var track: Track?
) : PlaylistTrack

@PlanningEntity
data class RestPlaylistTrack(
  @Json(name = "track")
  override var track: Track? = null,

  @PlanningVariable(
    graphType = PlanningVariableGraphType.CHAINED,
    valueRangeProviderRefs = ["firstTrack", "restTracks"]
  )
  var previousTrack: PlaylistTrack? = null,
) : PlaylistTrack
```


Multiple chains
OK
Anchor without trailing entity
OK
Initialzed entity without anchor
NOT OK
Multiple direct trailing entities
NOT OK
Loop
NOT OK
anchor
entity
entity
anchor
entity
anchor
anchor
entity
entity
entity
entity
entity
entity
entity
entity
entity
anchor
entity
entity
entity
anchor
entity
entity
entity
entity
entity
entity

The **FirstPlaylistTrack** remains fixed as the Anchor.

```
data class FirstPlaylistTrack(
    @Json(name = "track") override var track: Track?
) : PlaylistTrack

@PlanningEntity
data class RestPlaylistTrack(
  @Json(name = "track")
  override var track: Track? = null,

  @PlanningVariable(
    graphType = PlanningVariableGraphType.CHAINED,
    valueRangeProviderRefs = ["firstTrack", "restTracks"]
  )
  var previousTrack: PlaylistTrack? = null,
) : PlaylistTrack
```

The**RestPlaylistTracks** have a **previousTrack** value which mutates during planning.

```
data class FirstPlaylistTrack(
    @Json(name = "track") override var track: Track?
) : PlaylistTrack

@PlanningEntity
data class RestPlaylistTrack(
  @Json(name = "track")
  override var track: Track? = null,

  @PlanningVariable(
    graphType = PlanningVariableGraphType.CHAINED,
    valueRangeProviderRefs = ["firstTrack", "restTracks"]
  )
  var previousTrack: PlaylistTrack? = null,
) : PlaylistTrack
```

During Optimization, we experiment with alternative planning variables on each planning entity.

```
data class FirstPlaylistTrack(
    @Json(name = "track") override var track: Track?
) : PlaylistTrack

@PlanningEntity
data class RestPlaylistTrack(
  @Json(name = "track")
  override var track: Track? = null,

  @PlanningVariable(
    graphType = PlanningVariableGraphType.CHAINED,
    valueRangeProviderRefs = ["firstPlaylistTrackRange"]
  )
  var previousTrack: PlaylistTrack? = null,
) : PlaylistTrack
```

Now let's build our Planning Solution

```
@PlanningSolution
data class PlaylistSolution (
  @ProblemFactProperty
  val firstTrack: FirstPlaylistTrack,

  @ValueRangeProvider(id = "firstTrack")
  val firstTrackRange: List<FirstPlaylistTrack> = listOf(firstTrack),

  @ValueRangeProvider(id = "restTracks")
  @PlanningEntityCollectionProperty
  val restTracks: List<RestPlaylistTrack>,

  @ProblemFactCollectionProperty val artists: List<Artist>,
  @ProblemFactCollectionProperty val albums: List<Album>,

  @PlanningScore(bendableHardLevelsSize = 1, bendableSoftLevelsSize = 2)
  var score: BendableBigDecimalScore
)
```

The Planning Solution defines the ranges of possible previousTrack values for our RestPlaylistTracks

```kotlin
@PlanningSolution
data class PlaylistSolution (
  @ProblemFactProperty
  val firstTrack: FirstPlaylistTrack,

  @ValueRangeProvider(id = "firstTrack")
  val firstTrackRange: List<FirstPlaylistTrack> = listOf(firstTrack),

  @ValueRangeProvider(id = "restTracks")
  @PlanningEntityCollectionProperty
  val restTracks: List<RestPlaylistTrack>,

  @ProblemFactCollectionProperty val artists: List<Artist>,
  @ProblemFactCollectionProperty val albums: List<Album>,

  @PlanningScore(bendableHardLevelsSize = 1, bendableSoftLevelsSize = 2)
  var score: BendableBigDecimalScore
)
```

It also defines some **Facts** which can be referenced by our solver but don't mutate.

```
@PlanningSolution
data class PlaylistSolution (
  @ProblemFactProperty
  val firstTrack: FirstPlaylistTrack,

  @ValueRangeProvider(id = "firstTrack")
  val firstTrackRange: List<FirstPlaylistTrack> = listOf(firstTrack),

  @ValueRangeProvider(id = "restTracks")
  @PlanningEntityCollectionProperty
  val restTracks: List<RestPlaylistTrack>,

  @ProblemFactCollectionProperty val artists: List<Artist>,
  @ProblemFactCollectionProperty val albums: List<Album>,

  @PlanningScore(bendableHardLevelsSize = 1, bendableSoftLevelsSize = 2)
  var score: BendableBigDecimalScore
)
```

And finally a score which is used to track the quality of the "working solution" during solving.

```
@PlanningSolution
data class PlaylistSolution (
  @ProblemFactProperty
  val firstTrack: FirstPlaylistTrack,

  @ValueRangeProvider(id = "firstTrack")
  val firstTrackRange: List<FirstPlaylistTrack> = listOf(firstTrack),

  @ValueRangeProvider(id = "restTracks")
  @PlanningEntityCollectionProperty
  val restTracks: List<RestPlaylistTrack>,

  @ProblemFactCollectionProperty val artists: List<Artist>,
  @ProblemFactCollectionProperty val albums: List<Album>,

  @PlanningScore(bendableHardLevelsSize = 1, bendableSoftLevelsSize = 2)
  var score: BendableBigDecimalScore
)
```

Scores have slots which can be used to assign priority.

`[0]hard/[-55/-200]soft`

- |> `[0]hard` = no feasibility issues with the solution
- |> `[-55/-200]soft` = some things are still suboptimal at various degrees of severity

Let's write some rules

Drools Home Download Learn Get Help Source Services KIE

Overview

Drools is a Business Rules Management System (BRMS) solution. It provides a core Business Rules Engine (BRE), a web authoring and rules management application (Drools Workbench), full runtime support for Decision Model and Notation (DMN) models at Conformance level 3 and an Eclipse IDE plugin for core development.

More information can be found on the following links:

- Drools Workbench (web UI for authoring and management)
- Drools Expert (business rules engine)
- Drools Fusion (complex event processing features)
- jBPM (process/workflow integration for rule orchestration/flow)
- OptaPlanner (automated planning)

These projects have community releases from JBoss.org that come without support. Community releases focus on fast paced innovation to give you the latest and greatest, with releases every few months that include both features and fixes. Red Hat JBoss BRMS is our enterprise product for mission critical releases, with a multi year commitment for backport of fixes, based off a sanitised community release of Drools. A range of support packages are available including up to mission critical 24/7, as well as training and consultancy via our Global Professional Services unit. Check Red Hat Process Automation Manager for more details.

Let's make sure we only play a song once

```
rule "Should play each song only once"
    when
        RestPlaylistTrack(
            $t: track,
            previousTrack != null,
            $p: previousTrack
        )
        RestPlaylistTrack(
            track != $t,
            previousTrack == $p
        )
    then
        scoreHolder.addHardConstraintMatch(kcontext, 0, new BigDecimal(-1));
end
```

First we look for a track that has a previous track defined

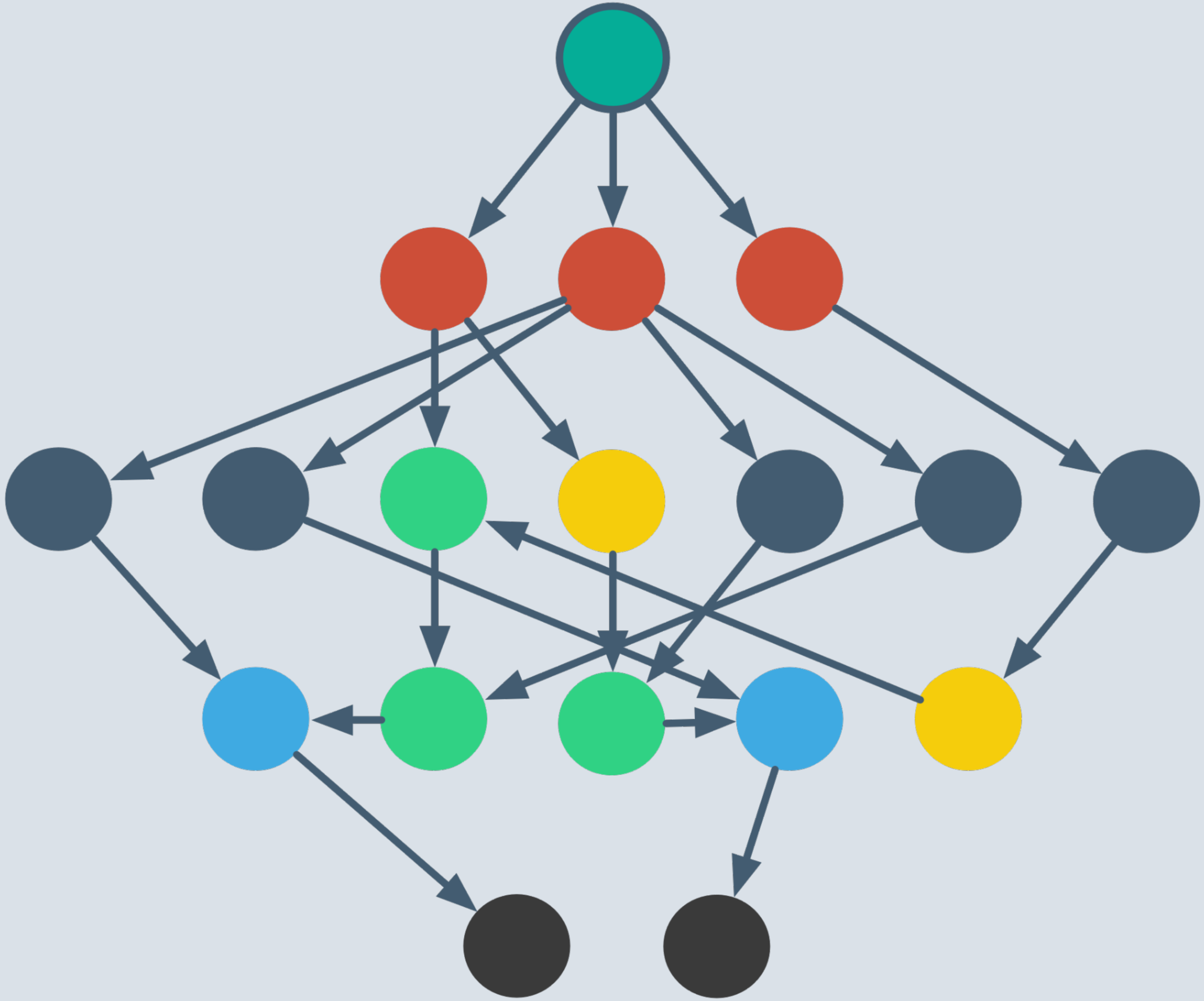
```
rule "Should play each song only once"
    when
        RestPlaylistTrack(
            $t: track,
            previousTrack != null,
            $p: previousTrack
        )
        RestPlaylistTrack(
            track != $t,
            previousTrack == $p
        )
    then
        scoreHolder.addHardConstraintMatch(kcontext, 0, new BigDecimal(-1));
end
```

Then we check to see if there's another track in the solution that points to that same previous track

```
rule "Should play each song only once"
    when
        RestPlaylistTrack(
            $t: track,
            previousTrack != null,
            $p: previousTrack
        )
        RestPlaylistTrack(
            track != $t,
            previousTrack == $p
        )
    then
        scoreHolder.addHardConstraintMatch(kcontext, 0, new BigDecimal(-1));
end
```

Anytime that happens, we add a hard constraint violation with a penalty of -1.

```
rule "Should play each song only once"
    when
        RestPlaylistTrack(
            $t: track,
            previousTrack != null,
            $p: previousTrack
        )
        RestPlaylistTrack(
            track != $t,
            previousTrack == $p
        )
    then
        scoreHolder.addHardConstraintMatch(kcontext, 0, new BigDecimal(-1));
end
```



But what about key rules?

Let's add a shadow variable!

```
@PlanningEntity
data class RestPlaylistTrack(
  @Json(name = "track")
  override var track: Track? = null,

  @PlanningVariable(
    graphType = PlanningVariableGraphType.CHAINED,
    valueRangeProviderRefs = ["firstPlaylistTrackRange", "rest"]
  )
  var previousTrack: PlaylistTrack? = null,

  @CustomShadowVariable(
    variableListenerClass = PreviousTrackUpdatedListener::class,
    sources = [PlanningVariableReference(variableName = "previousTrack")]
  )
  var keyDistance: Int? = null
)
```

And now for the listener...

```kotlin
class PreviousTrackUpdatedListener : VariableListener<RestPlaylistTrack> {
  override fun afterEntityAdded(scoreDirector: ScoreDirector<*>, playlistTrack: RestPlaylistTrack) {
    update(scoreDirector, playlistTrack)
  }


  override fun afterVariableChanged(scoreDirector: ScoreDirector<*>, playlistTrack: RestPlaylistTrack) {
    update(scoreDirector, playlistTrack)
  }

  private fun update(scoreDirector: ScoreDirector<*>, playlistTrack: RestPlaylistTrack) {
    var distance: Int? = null
    playlistTrack.track?.features?.key?.let { thisKey ->
      playlistTrack.previousTrack?.track?.features?.key?.let { previousKey ->
        val noteDistance = Math.abs(
          thisKey.camelotPosition!! - previousKey.camelotPosition!!
        )
        val modeDistance = if (thisKey != previousKey) 1 else 0
        distance = (if (noteDistance < 6) noteDistance else noteDistance - (noteDistance % 6)) + modeDistance
      }
    }
    if(playlistTrack.keyDistance != distance) {
      scoreDirector.beforeVariableChanged(playlistTrack, "keyDistance")
      playlistTrack.keyDistance = distance
      scoreDirector.afterVariableChanged(playlistTrack, "keyDistance")
    }
  }
}
```

This part lets us know whenever stuff happens with the planning entity.

```
class PreviousTrackUpdatedListener : VariableListener<RestPlaylistTrack> {
  override fun afterEntityAdded(scoreDirector: ScoreDirector<*>, playlistTrack: RestPlaylistTrack) {
    update(scoreDirector, playlistTrack)
  }

  override fun afterVariableChanged(scoreDirector: ScoreDirector<*>, playlistTrack: RestPlaylistTrack) {
    update(scoreDirector, playlistTrack)
  }

  private fun update(scoreDirector: ScoreDirector<*>, playlistTrack: RestPlaylistTrack) {
    var distance: Int? = null
    playlistTrack.track?.features?.key?.let { thisKey ->
      playlistTrack.previousTrack?.track?.features?.key?.let { previousKey ->
        val noteDistance = Math.abs(
          thisKey.camelotPosition!! - previousKey.camelotPosition!!
        )
        val modeDistance = if (thisKey != previousKey) 1 else 0
        distance = (if (noteDistance < 6) noteDistance else noteDistance - (noteDistance % 6)) + modeDistance
      }
    }
    if(playlistTrack.keyDistance != distance) {
      scoreDirector.beforeVariableChanged(playlistTrack, "keyDistance")
      playlistTrack.keyDistance = distance
      scoreDirector.afterVariableChanged(playlistTrack, "keyDistance")
    }
  }
}
```

This is the fancy business logic we need to execute to get the camelot distance (because its a circle).

```
class PreviousTrackUpdatedListener : VariableListener<RestPlaylistTrack> {
  private fun update(scoreDirector: ScoreDirector<*>, playlistTrack: RestPlaylistTrack) {
    var distance: Int? = null
    playlistTrack.track?.features?.key?.let { thisKey ->
      playlistTrack.previousTrack?.track?.features?.key?.let { previousKey ->
        val noteDistance = Math.abs(
          thisKey.camelotPosition!! - previousKey.camelotPosition!!
        )
        val modeDistance = if (thisKey != previousKey) 1 else 0
        distance = (if (noteDistance < 6) noteDistance else noteDistance - (noteDistance % 6)) + modeDistance
      }
    }
    if(playlistTrack.keyDistance != distance) {
      scoreDirector.beforeVariableChanged(playlistTrack, "keyDistance")
      playlistTrack.keyDistance = distance
      scoreDirector.afterVariableChanged(playlistTrack, "keyDistance")
    }
  }
}
```

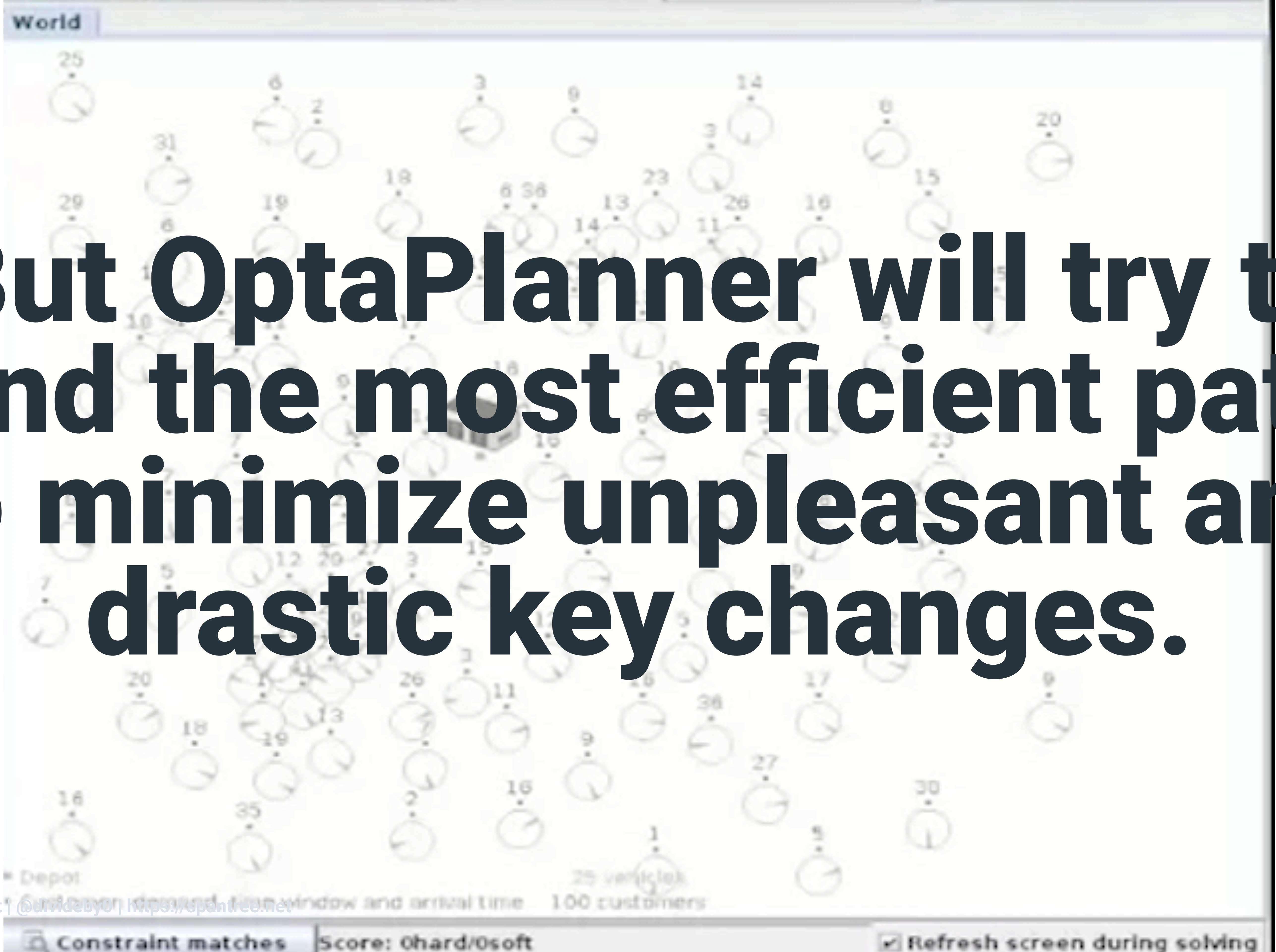
This part tells OptaPlanner we've changed somethingbrew install asciinema2gif.

```
class PreviousTrackUpdatedListener : VariableListener<RestPlaylistTrack> {
  private fun update(scoreDirector: ScoreDirector<*>, playlistTrack: RestPlaylistTrack) {
    var distance: Int? = null
    playlistTrack.track?.features?.key?.let { thisKey ->
      playlistTrack.previousTrack?.track?.features?.key?.let { previousKey ->
        val noteDistance = Math.abs(
          thisKey.camelotPosition!! - previousKey.camelotPosition!!
        )
        val modeDistance = if (thisKey != previousKey) 1 else 0
        distance = (if (noteDistance < 6) noteDistance else noteDistance - (noteDistance % 6)) + modeDistance
      }
    }
    if(playlistTrack.keyDistance != distance) {
      scoreDirector.beforeVariableChanged(playlistTrack, "keyDistance")
      playlistTrack.keyDistance = distance
      scoreDirector.afterVariableChanged(playlistTrack, "keyDistance")
    }
  }
}
```

So now that we've done all the heavy lifting in Kotlintown, our key rule is dead simple.

```
rule "Key distance should be kept to a minimum"
    when
       RestPlaylistTrack(
           keyDistance != null,
           keyDistance > 0,
           $kd: keyDistance
       )
    then
        scoreHolder.addSoftConstraintMatch(kcontext, 0, new BigDecimal(-$kd*$kd));
end
```

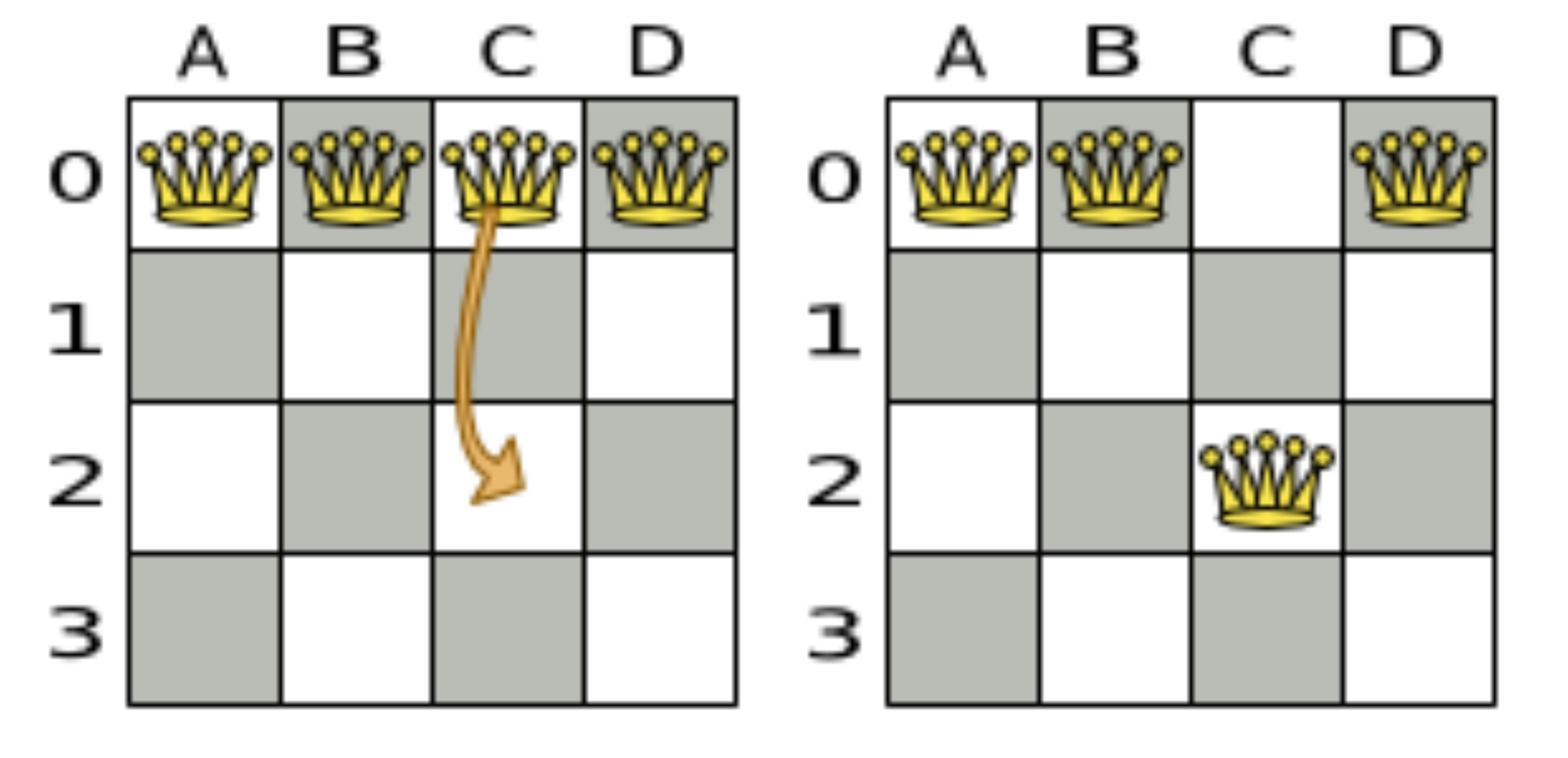
Since most playlists contain songs in more than one key, the score will never be zero.



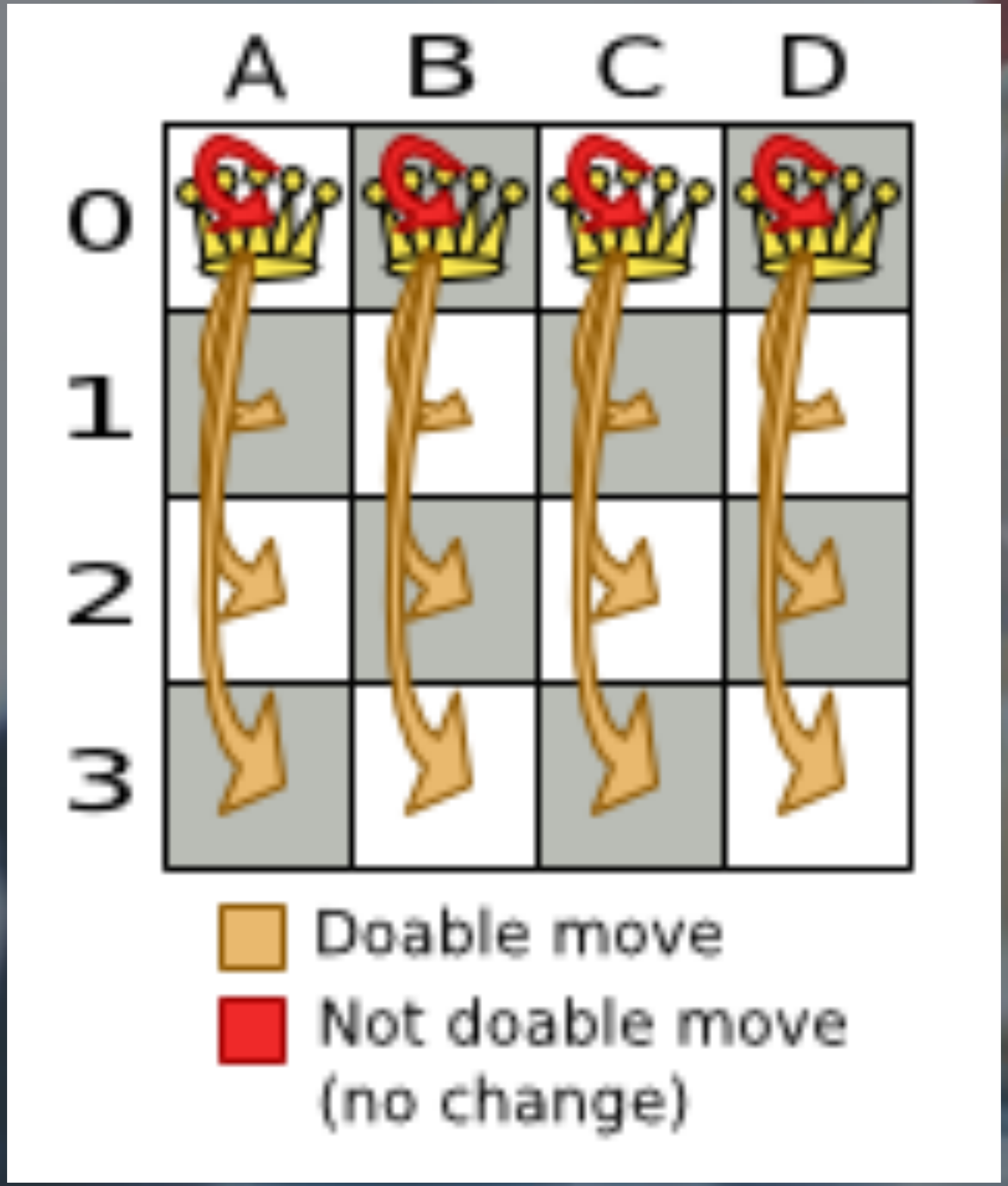
But OptaPlanner will try to find the most efficient path to minimize unpleasant and drastic key changes.

How does it do that?

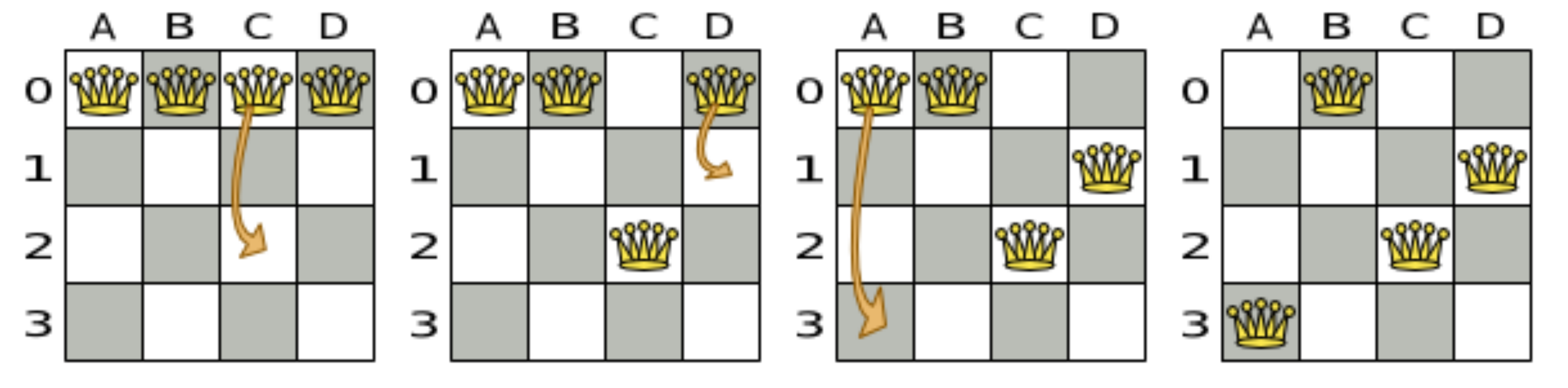
QC0 -> C2



Let's examine all possible moves.

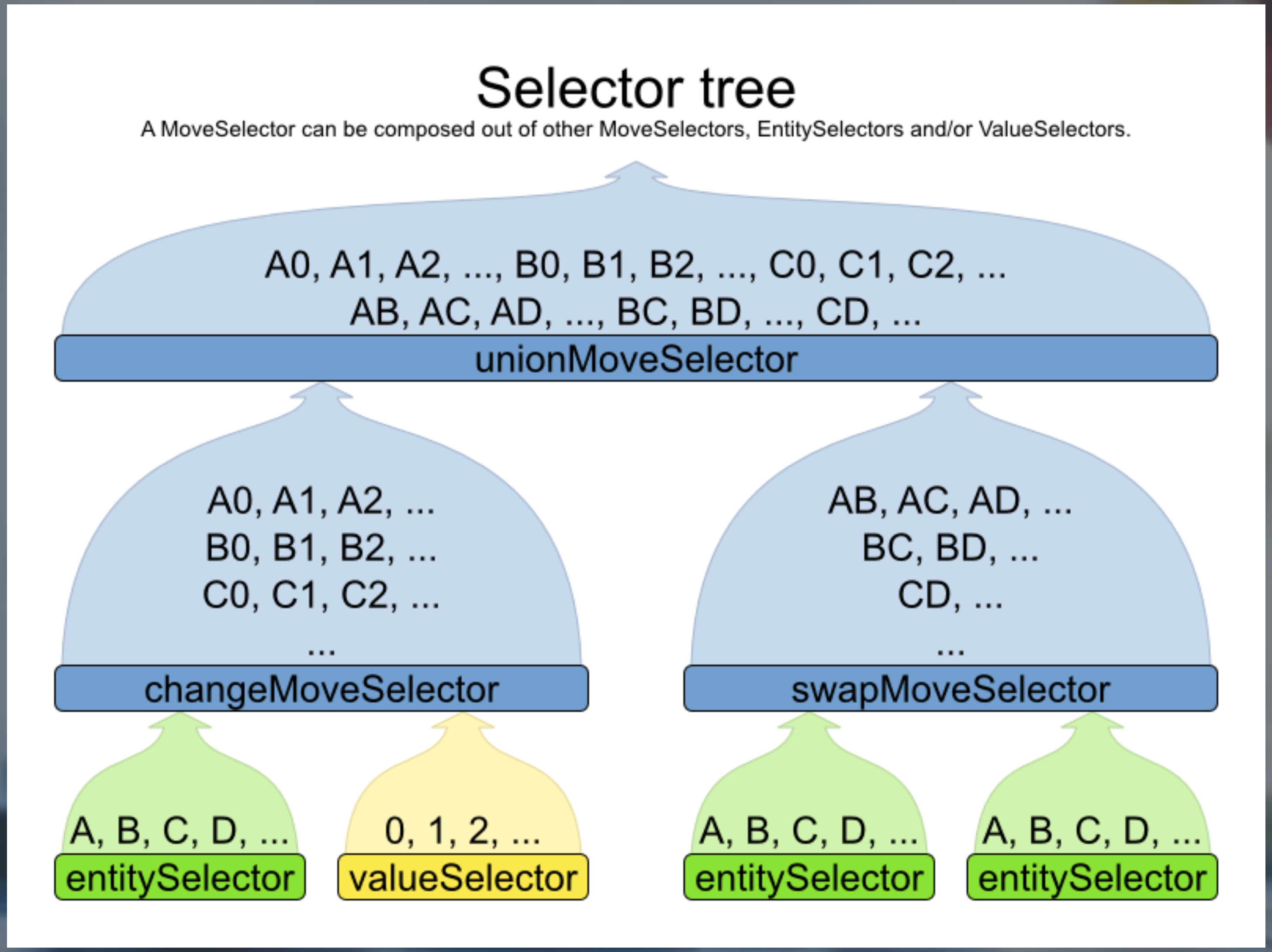


The sequence of moves impact the next viable move.



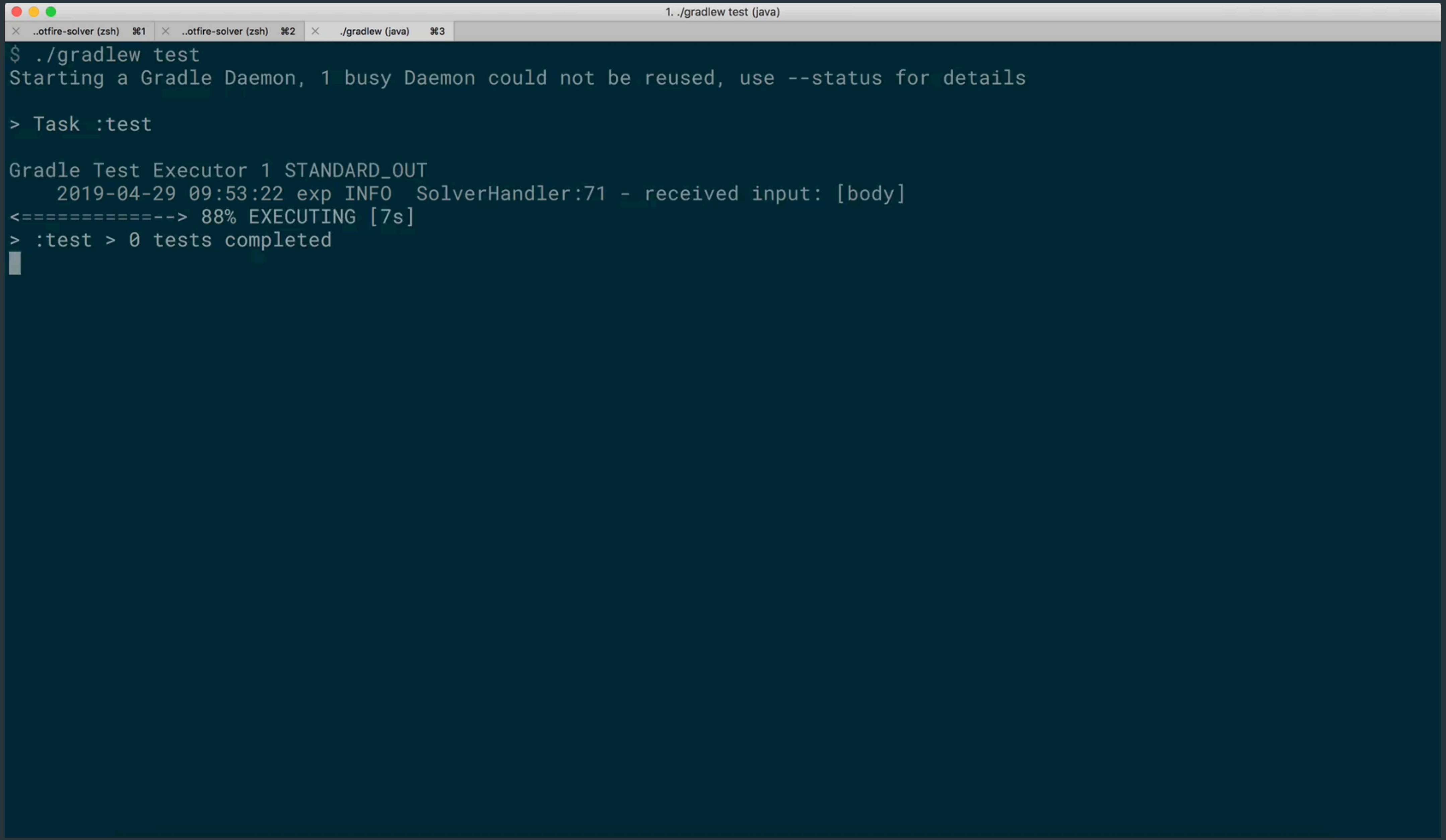
OptaPlanner calculates a random series of moves...

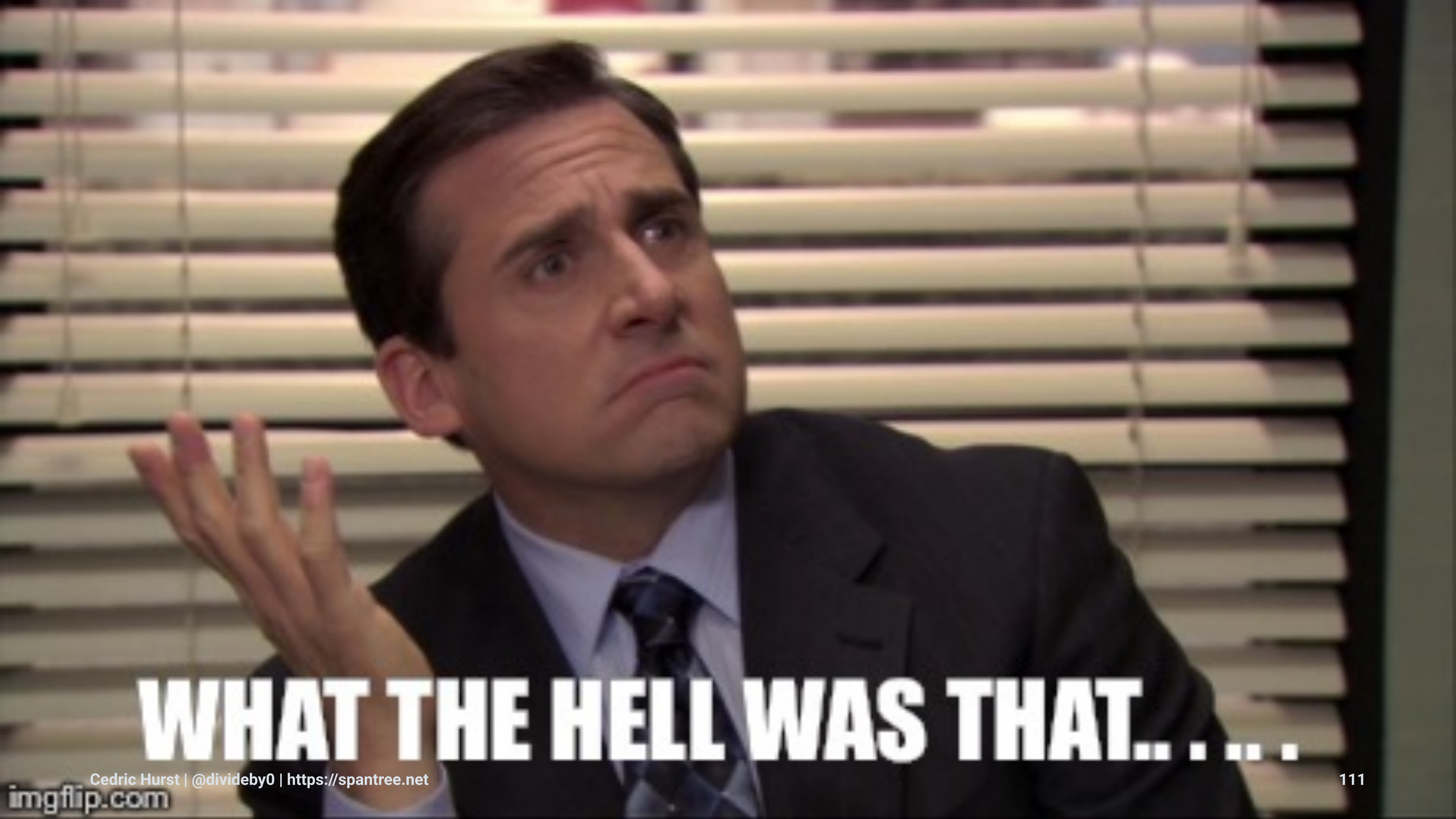
Selecting a subset that (eventually) make the score **better**.



So how does this apply to playlists?

Let's watch a solver in action!




WHAT THE HELL WAS THAT.. . .. .
imgflip.com

Let's dissect a log line.

```
CH step (850),
time spent (1534),
score ([0]hard/[-17456/-3092]soft),
selected move count (1),
picked move (
  Oneohtrix Point Never - RayCats (8A) {
    null -> St. Vincent - Strange Mercy (7B)
  }
)
```

First we start with an empty playlist...

...and run a **Construction Heuristic** to select previous tracks

```
CH step (0),
time spent (257),
score (-850init/[-850]hard/[-9/0]soft),
selected move count (1),
picked move (
  Kaki King - I Never Said I Love You (9A) {
    null -> St. Vincent - Strange Mercy (7B)
  }
)
```

We do this one track at a time...

and pick the track that makes the overall **solution score better**.

```
CH step (0),
time spent (257),
score (-850init/[-850]hard/[-9/0]soft),
selected move count (1),
picked move (
  Kaki King - I Never Said I Love You (9A) {
    null -> St. Vincent - Strange Mercy (9A)
  }
)
```

This step decided to transition from St Vincent to Kaki King

this makes sense because they're in nearby keys.

```
CH step (0),
time spent (257),
score (-850init/[-850]hard/[-9/0]soft),
selected move count (1),
picked move (
  Kaki King - I Never Said I Love You (9A) {
    null -> St. Vincent - Strange Mercy (7A)
  }
)
```

In this phase, we can't undo or reconsider moves.

Once you've decided on a transition, you're stuck with it for a while.

```
CH step (850),
time spent (1300),
score ([0]hard/[-17456/-3092]soft),
selected move count (1),
picked move (
  Oneohtrix Point Never - RayCats (8A) {
    null -> St. Vincent - Strange Mercy (7B)
  }
)
```

If this is all we do, we'll quickly hit a **Local Optimum**.

What if that Kaki King track was the only track that worked after Unknown Mortal Orchestra?

Let's do a Local Search!

```
LS step (251),
time spent (534861),
score ([0]hard/[-4142/-1222]soft),
new best score ([0]hard/[-4142/-1222]soft),
accepted/selected move count (29/221),
picked move (
  Passion Pit - Little Secrets (7B) {
    Fleet Foxes - Bedouin Dress (9B)
  } <-tailChainSwap-> Steve Reich - Mallet Quartet: III. Fast (3A) {
    Ali Farka Touré - Yer Bounda Fara (5B)
  }
)
```

After a (long) while, we end up with an optimized solution.

```
Solving ended: time spent (601024),
best score ([0]hard/[-3475/-1088]soft),
score calculation speed (112/sec),
phase total (2),
environment mode (REPRODUCIBLE)
```

For most playlists, we'll never get a perfect solution.

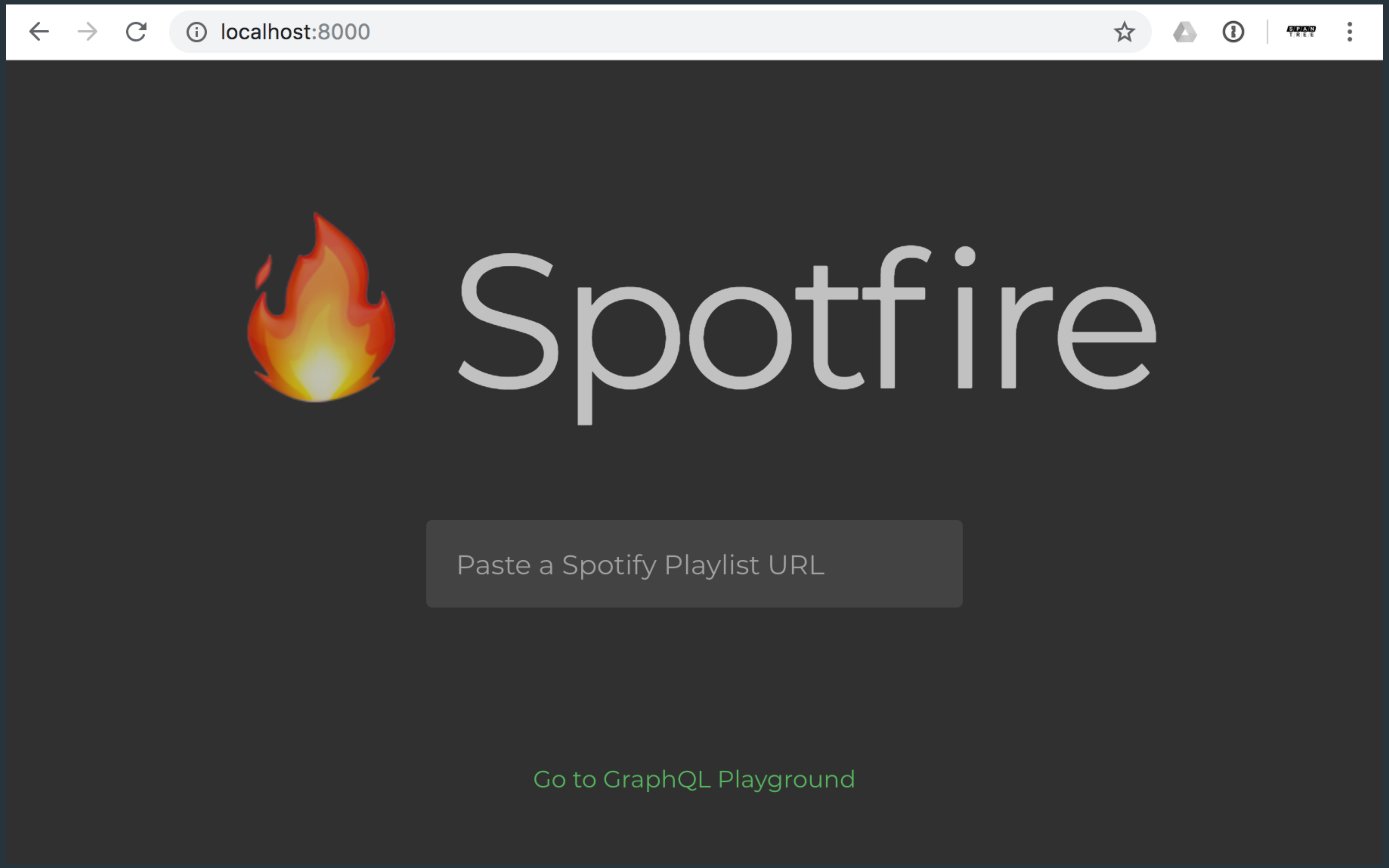
But OptaPlanner can tell us what's suboptimal.

This is thanks to Drools and its awesome rule algorithm.

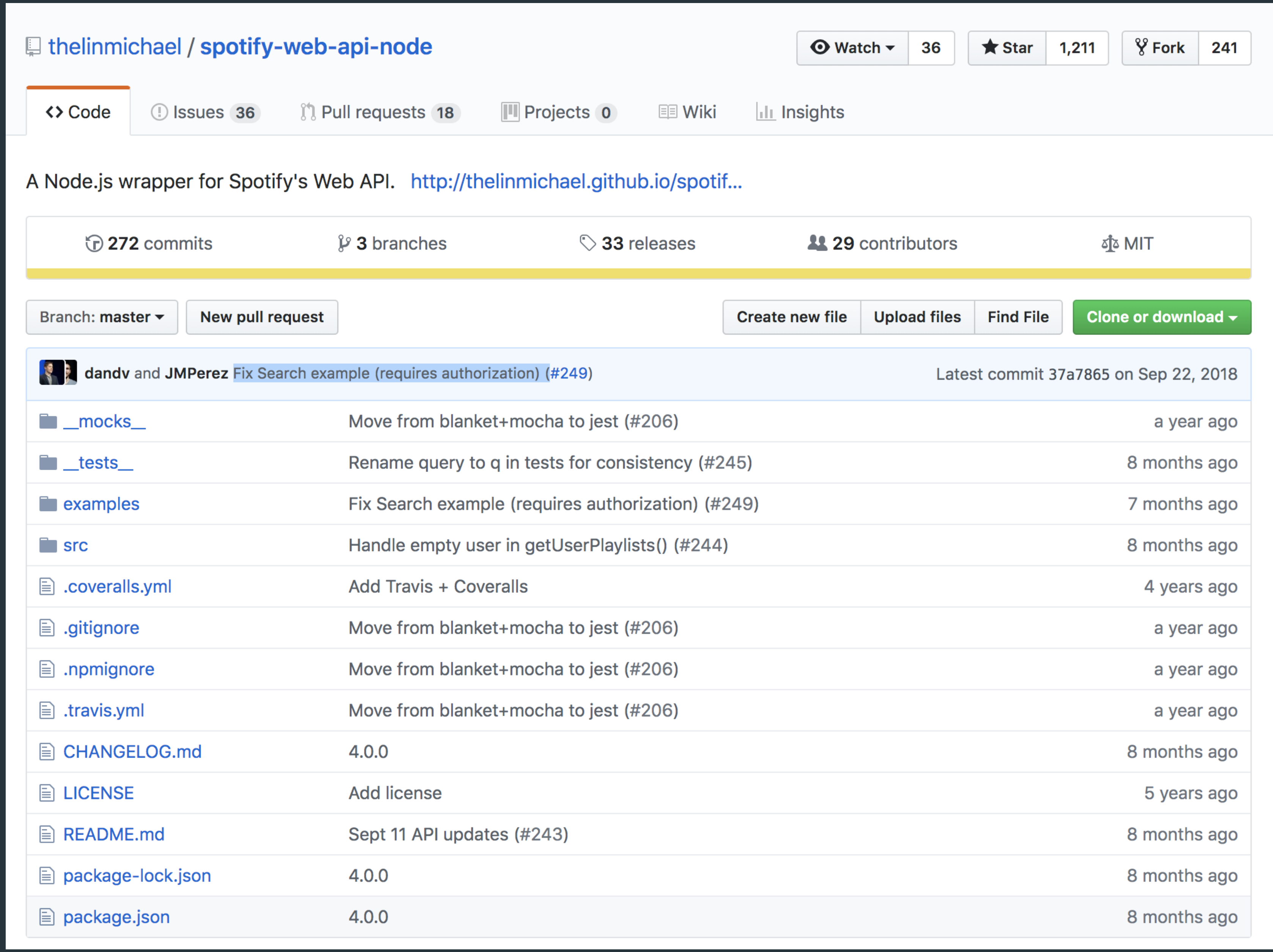
```
ConstraintViolationReporter:28
Key distance should be kept to a minimum ->
  violations: 625,
  score impact: [-2174/0]soft

- Violation 0, score impact: ([0]hard/[-49/0]soft)
  - Max Tundra - Will Get Fooled Again (1B) -> Wilco - Impossible Germany (9A)
- Violation 1, score impact: ([0]hard/[-25/0]soft
  - Erykah Badu + DRAM - WiFi (5A) -> Broken Social Scene - Halfway Home (9B)
```

So Let's See It in Action!

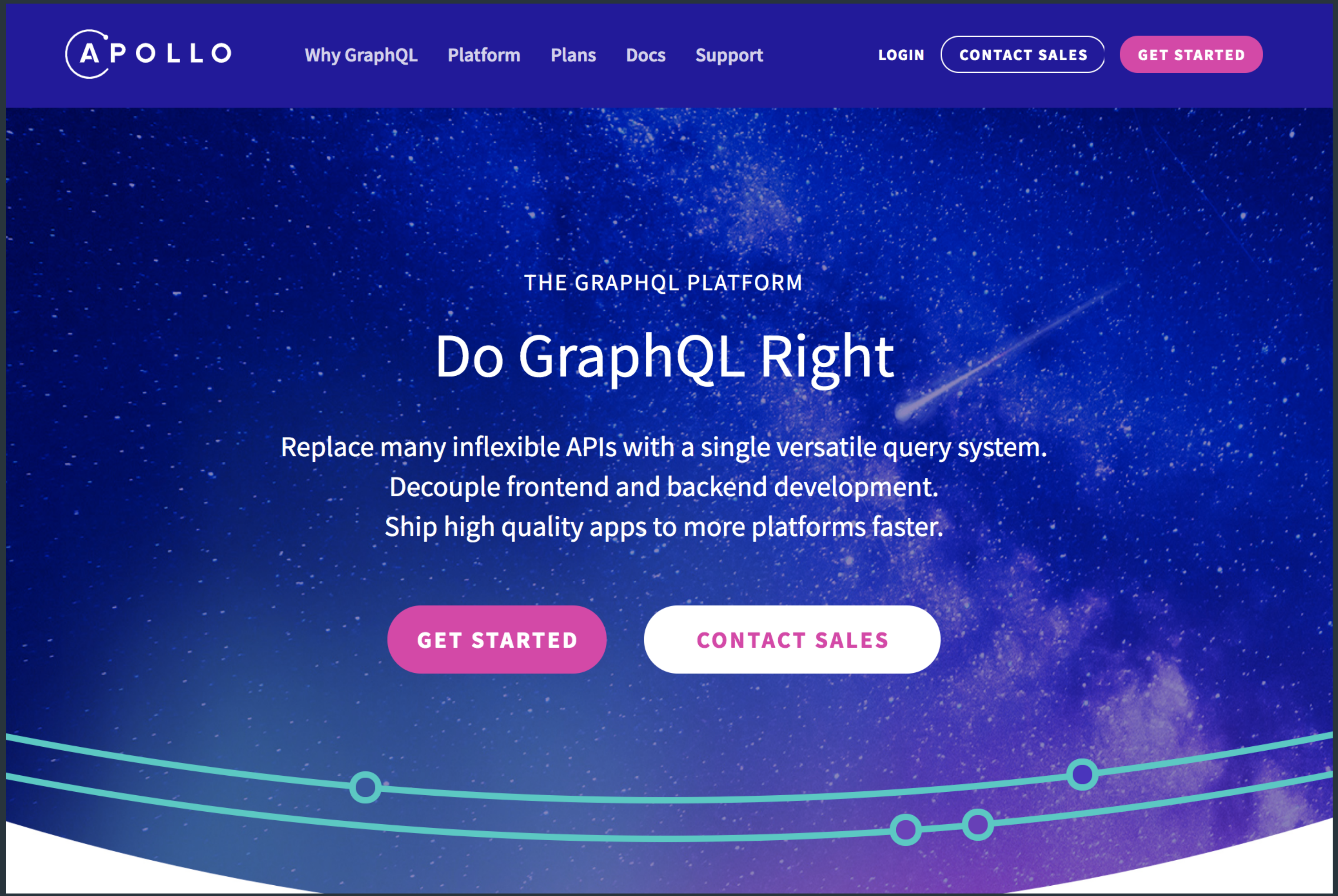
This is a talk about failure.



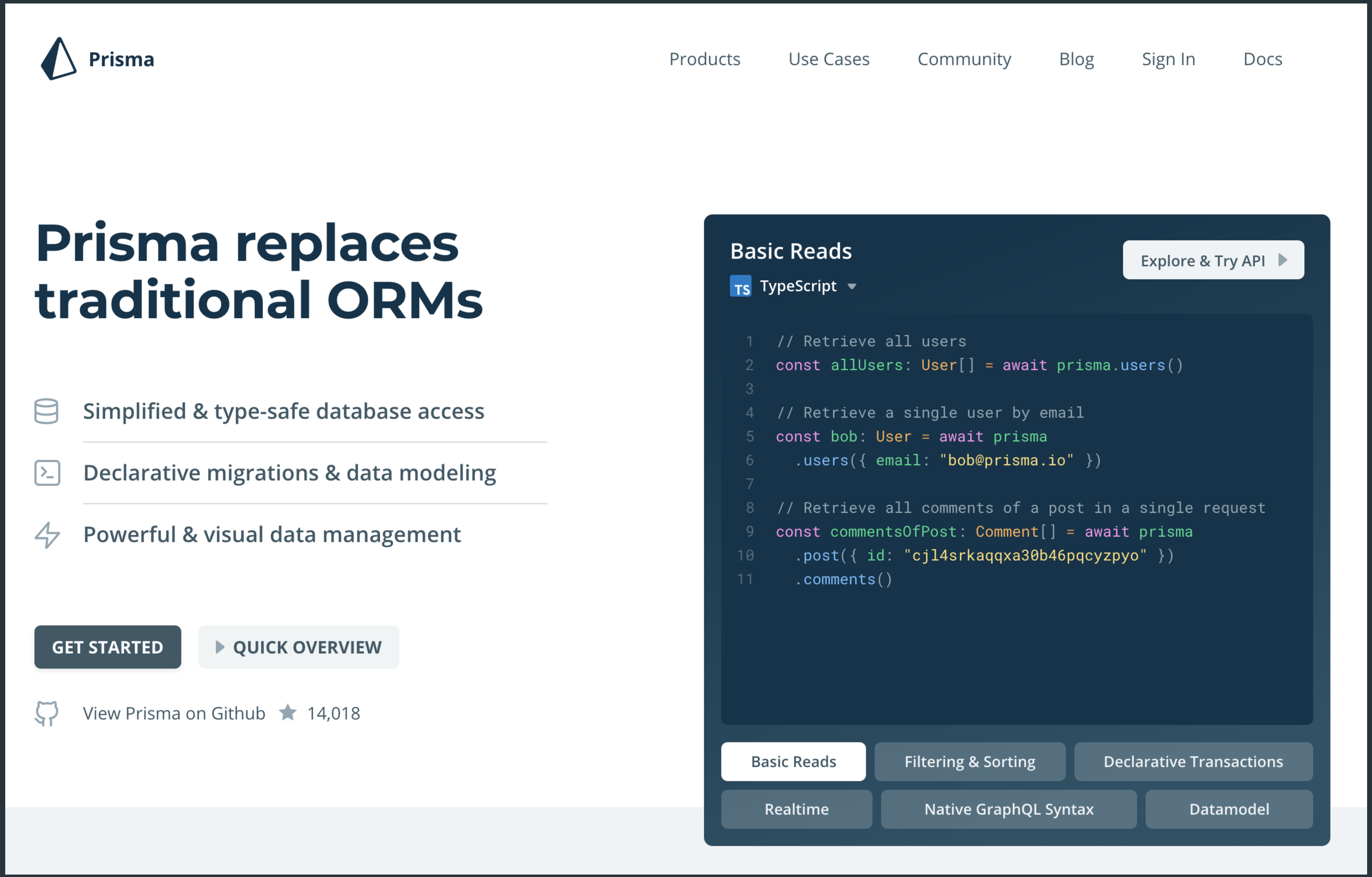
Let's Pull Some Data On-Demand from Spotify.



Let's Slap Some GraphQL on Top.



Let's Persist Some Data.



Let's Grab a Playlist.

PRETTIFY HISTORY http://localhost:4001/

```
query {
  playlist(
    uri: "https://open.spotify.com/user/jcoll
  ) {
    playlist_id
    collaborative
    latest_snapshot {
      playlist_tracks {
        added_at
        track {
          artists { name }
          name
          duration_ms
          features {
            tempo
            key { label }
          }
        }
      }
    }
  }
}
```


```
{
  "data": {
    "playlist": {
      "playlist_id": "4kdrAVuPxDdfwOnchhexK1",
      "collaborative": false,
      "latest_snapshot": {
        "playlist_tracks": [
          {
            "added_at": "2017-05-14T13:36:20.000Z",
            "track": {
              "artists": [
                {
                  "name": "Sam Amidon"
                }
              ],
              "name": "Roll On John",
              "duration_ms": 172690,
              "features": {
                "tempo": 78.66,
                "key": {
                  "label": "F♯Maj"
                }
              }
```

Now Let's Optimize.

PRETTIFY HISTORY http://localhost:4001/ COPY CURL

```graphql
mutation {
  optimizePlaylist(
    playlist_id: "4kdrAVuPxDdfwOn...",
    snapshot_id: "NTc5LDI4YmRkNWI..."
  ) {
    id
    extract_path
    status
  }
}
```

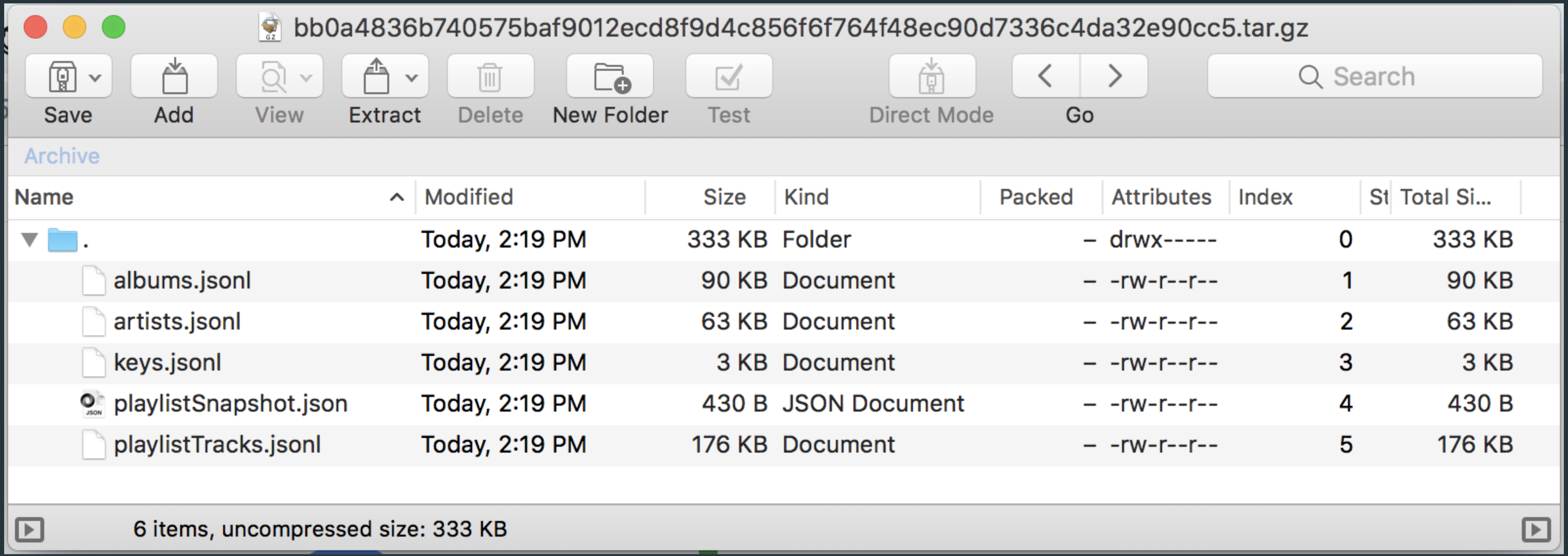

```json
{
  "data": {
    "optimizePlaylist": {
      "id": "cjv2qy5gv001w0783091fxnzc",
      "extract_path":
"https://s3.amazonaws.com/spotfire-extracts/bb0a4836b740575baf9012ecd8f9d4c856f6f764f48ec90d7336c4da32e90cc5.tar.gz",
      "status": "INITIALIZED"
    }
  },
  "extensions": {}
}
```


SCHEMA

What's that extract path?


```
{
  "data": {
    "optimizePlaylist": {
      "id": "cjv2qy5gv001w0783091fxnzc",
      "extract_path":
      "https://s3.amazonaws.com/spotfire-extracts/bb0a483632e90cc5.tar.gz",
      "status": "INITIALIZED"
    }
  },
  "extensions": {}
}
```

Let's take a peek at that file.



bb0a4836b740575baf9012ecd8f9d4c856f6f764f48ec90d7336c4da32e90cc5.tar.gz

Save Add View Extract Delete New Folder Test Direct Mode Go Search

Archive

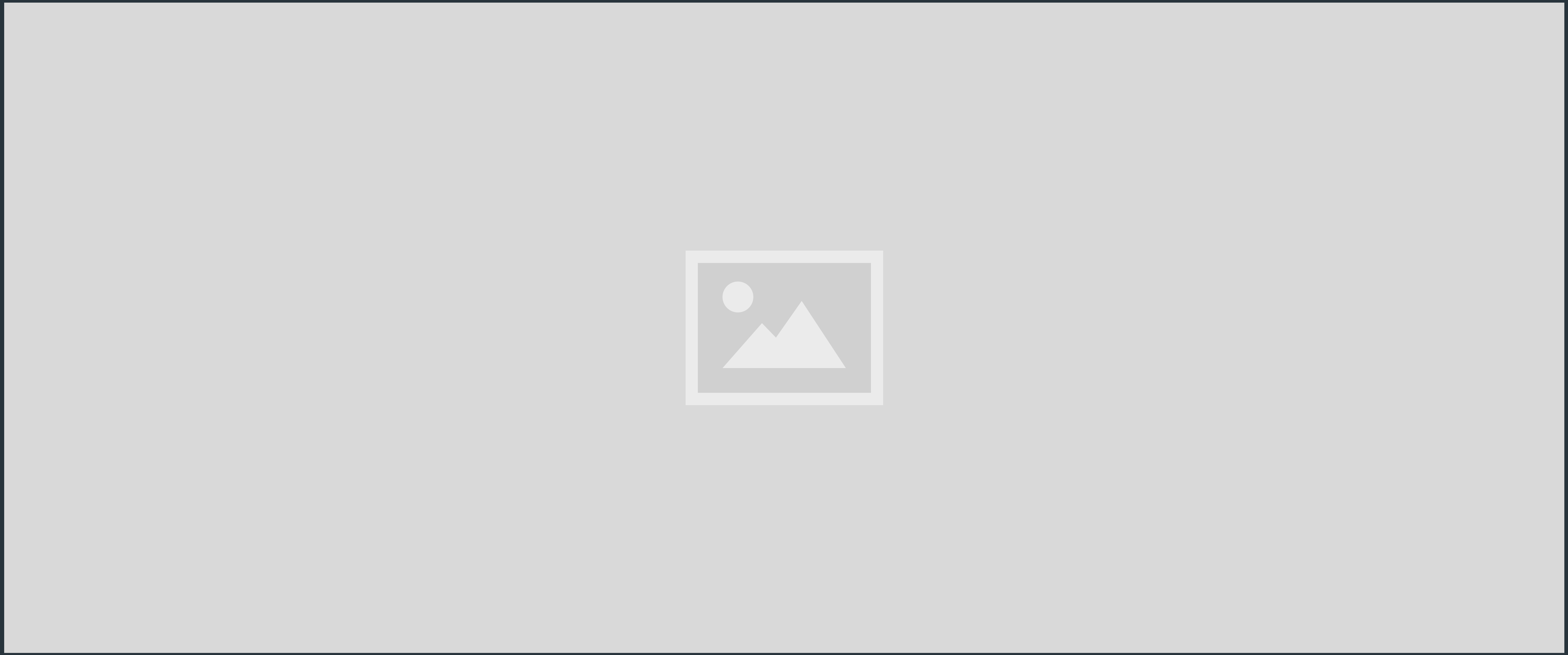
Name	Modified	Size	Kind	Packed	Attributes	Index	St	Total Si...
.	Today, 2:19 PM	333 KB	Folder	–	drwx-----	0		333 KB
albums.jsonl	Today, 2:19 PM	90 KB	Document	–	-rw-r--r--	1		90 KB
artists.jsonl	Today, 2:19 PM	63 KB	Document	–	-rw-r--r--	2		63 KB
keys.jsonl	Today, 2:19 PM	3 KB	Document	–	-rw-r--r--	3		3 KB
playlistSnapshot.json	Today, 2:19 PM	430 B	JSON Document	–	-rw-r--r--	4		430 B
playlistTracks.jsonl	Today, 2:19 PM	176 KB	Document	–	-rw-r--r--	5		176 KB

6 items, uncompressed size: 333 KB

Now comes the part that's not hooked up yet.

```
serverless invoke --function solver -p "src/test/payloads/collier.json"
2019-04-29 20:28:04 DEBUG DefaultLocalSearchPhase:133 -     LS step (2), time spent (2918), score ([0]hard/[-6367/-1093]soft), new best score ([0]hard/[-6367/-1093]soft), ...
2019-04-29 20:28:05 DEBUG DefaultLocalSearchPhase:133 -     LS step (3), time spent (3090), score ([0]hard/[-6274/-1082]soft), new best score ([0]hard/[-6274/-1082]soft), ...
```

Coming Soon



Thanks

Spantree (esp Mari, Eli + Justin)
GOTO Conference
Spotify + The Echo Nest Teams
* OptaPlanner, Apollo and Prisma teams for great open source tools