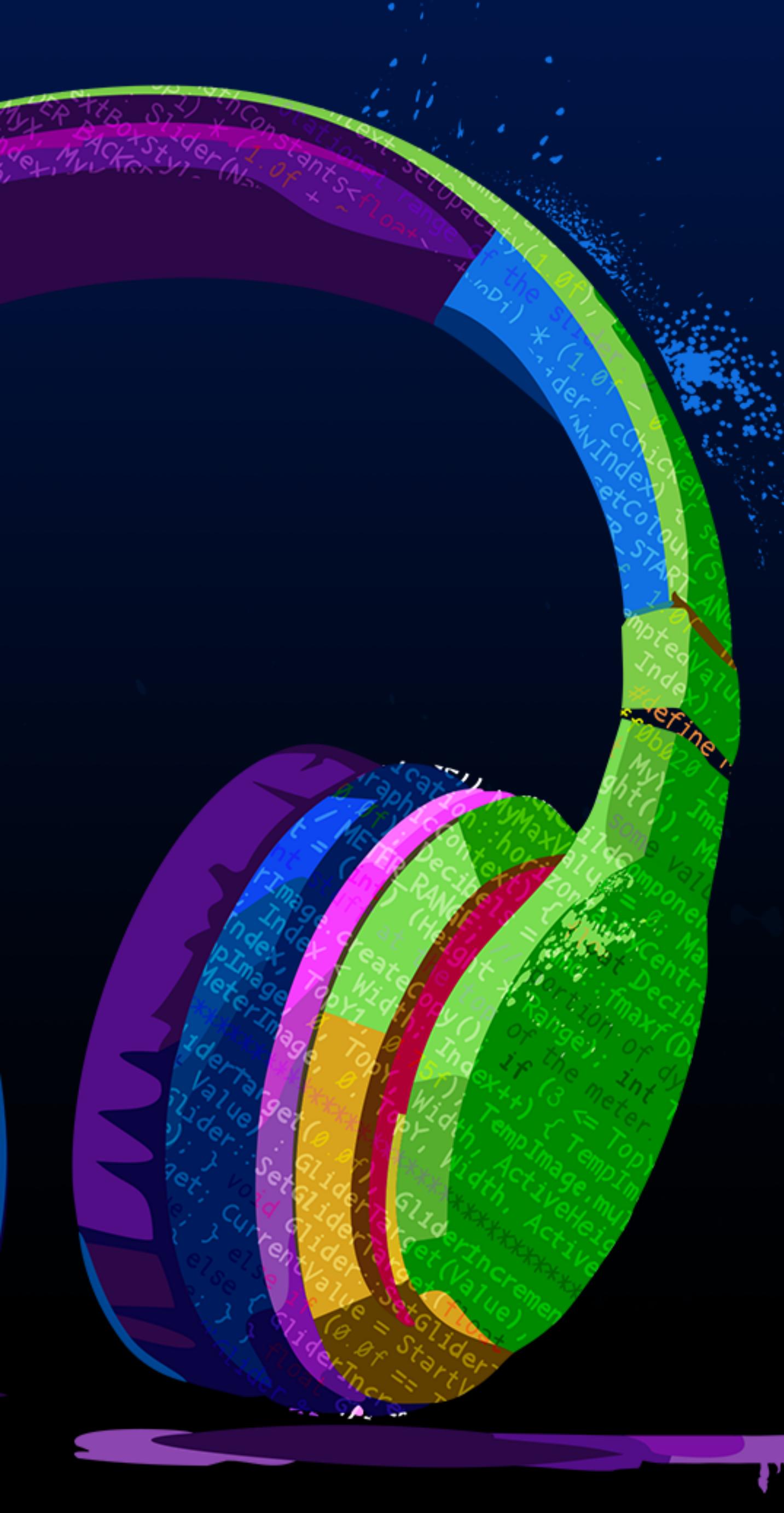




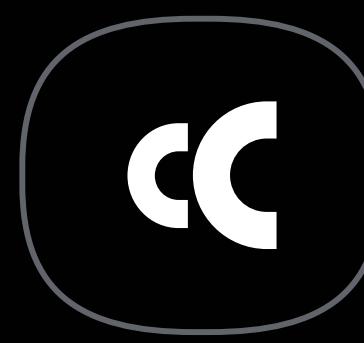
# HIGH PERFORMANCE WEBVIEW USER INTERFACES

## *GETTING THE MOST OUT OF THE 2D CANVAS AND DOM*

**ARTHUR CARABOTT**



# OUTPUT



# OUTPUT CREATOR

**Render Duration:**  
0.00 ms (0.00 - 0.00 ms)  
0 fps ( 0 - 0 fps)

**Actions:**  
Scrolling, Dragging, Zooming

**Visible Regions:**  
1710 (max: 1710)

**Screen:**  
2560 x 1440 @ 2x

**Browse All**

**Vocals Collection**

**Bleeding Heart**

- Strong Effort Vox No No 01
- Strong Effort Vox Broken ...
- Strong Effort Vox Yeah Ho...
- Strong Effort Vox The Har...
- Strong Effort Vox Hoh Wh...
- Strong Effort Vox Broken ...
- Strong Effort Vox Pretty Li...
- Strong Effort Vox Uhh Uh...
- Strong Effort Vox Twisted ...
- Strong Effort Vox Yeah Ye...
- Strong Effort Vox Wuh Oh...
- Strong Effort Vox We Coul...
- Strong Effort Vox Try 01
- Strong Effort Vox The Har...
- Strong Effort Vox Yeah Oo...

**Keep It Casual**

**Sweet and Low**

**Bottle Service**

**MacBook Pro**  
13-inch, M1, 2020  
Chip Apple M1  
Memory 8 GB  
Startup disk Macintosh HD  
Serial number C02F01P0Q05H  
macOS Sonoma 14.5

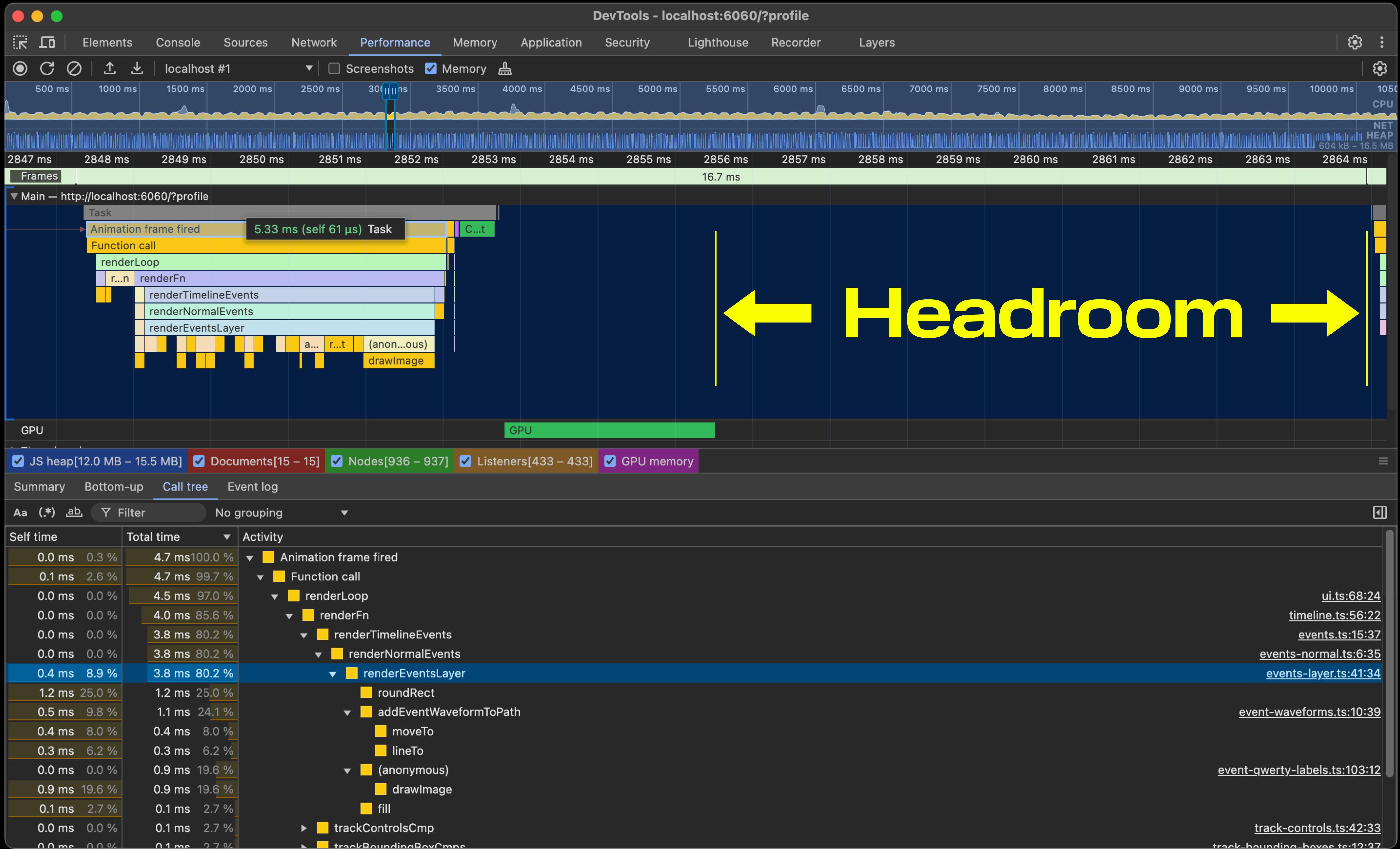
More Info...

Regulatory Certification  
TM and © 1983-2024 Apple Inc.  
All Rights Reserved.

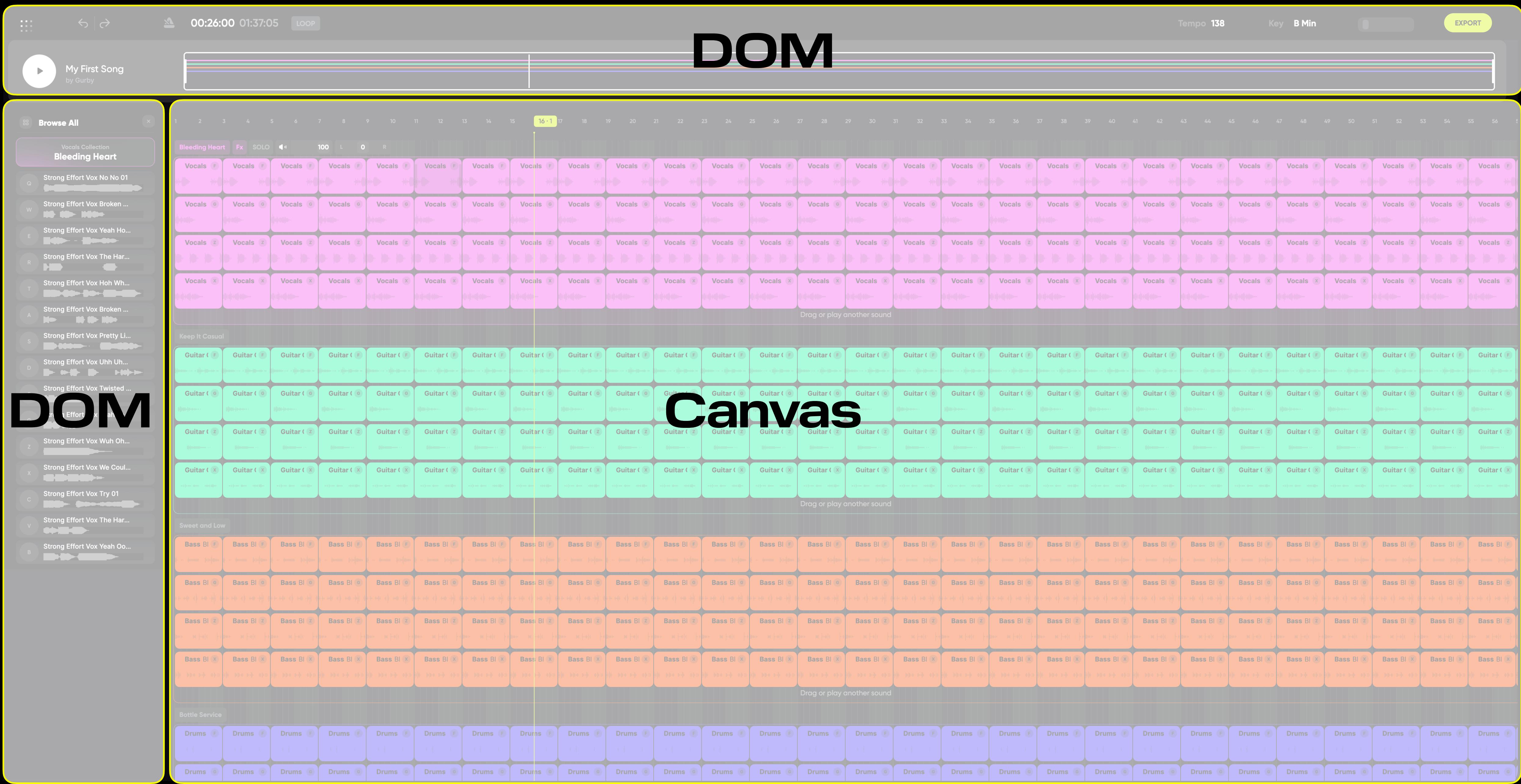
# Render Duration:

5.00 ms (1.90 - 8.60 ms)  
200 fps ( 116 - 526 fps)









# OUTPUT

# DOM

```
<section>
  <button>Toggle Metronome</button>
  <label>Tempo: </label>
  <input type="range" min="1" max="999" />
</section>
```

# HTML

```
<section>  
  
  <button>  
    |   Toggle Metronome  
  </button>  
  
  <label>  
    |   Tempo:  
  </label>  
  
  <input  
    |   type="range"  
    |   min="1"  
    |   max="999"  
  />  
  
</section>
```

# JavaScript

```
const topEl = document.createElement("section");  
  
{  
  const metronomeEl = document.createElement("button");  
  metronomeEl.textContent = "Toggle Metronome";  
  topEl.appendChild(metronomeEl);  
}  
  
{  
  const tempoLabelEl = document.createElement("label");  
  tempoLabelEl.textContent = "Tempo:";  
  
  const tempoEl = document.createElement("input");  
  tempoEl.type = "range";  
  tempoEl.min = "1";  
  tempoEl.max = "999";  
  topEl.appendChild(tempoLabelEl);  
}
```

# Canvas

```
ctx.beginPath();
ctx.roundRect( 10, 10, 400, 200, 10);
ctx.roundRect(400, 220, 200, 100, 10);

ctx.fillStyle = "rgb(43, 156, 212)";
ctx.fill();
```

"When performance matters,  
Canvas beats DOM"

**Me**  
This talk, today

1. Canvas
2. DOM
3. Layering

**OUTPUT**

<canvas>

2D



# 2D

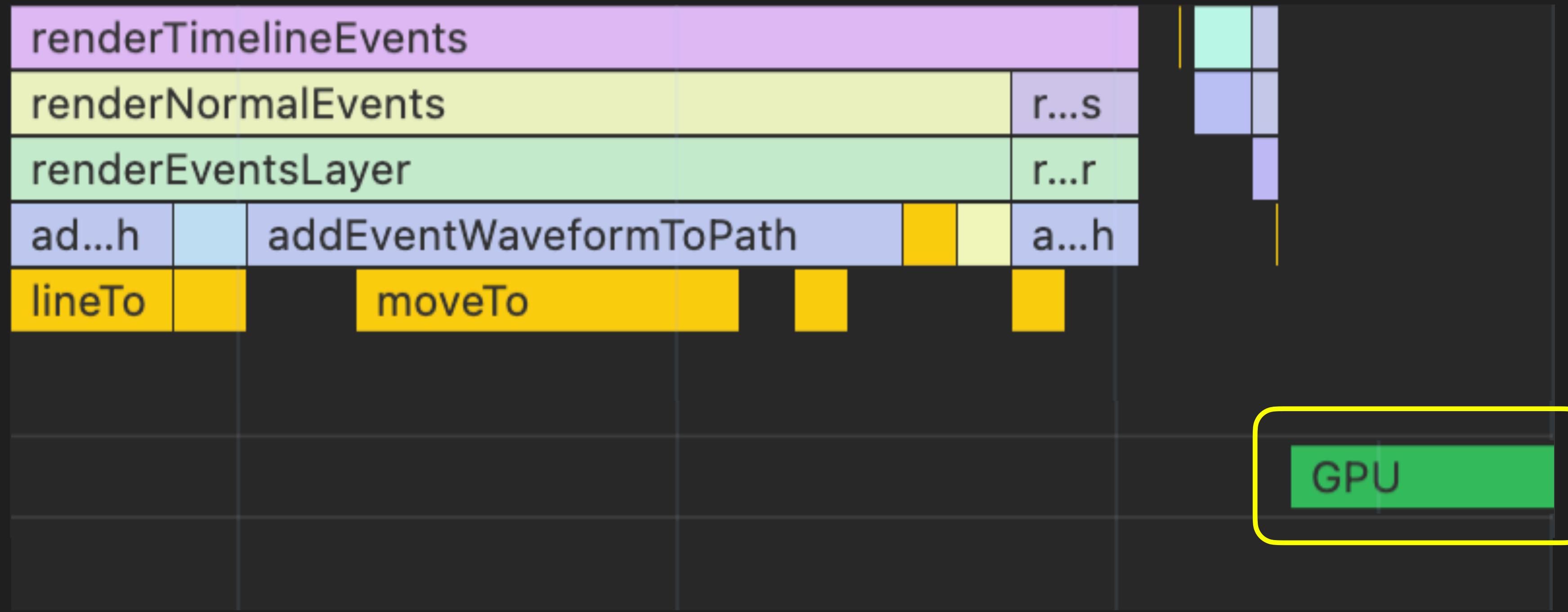
```
ctx.beginPath();
ctx.roundRect( 10, 10, 400, 200, 10);
ctx.roundRect(400, 220, 200, 100, 10);

ctx.fillStyle = "rgb(43, 156, 212)";
ctx.fill();
```



OUTPUT

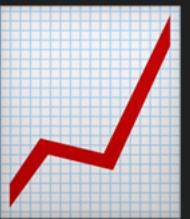
# 2D



## Hardware Accelerated

# Tradeoffs:

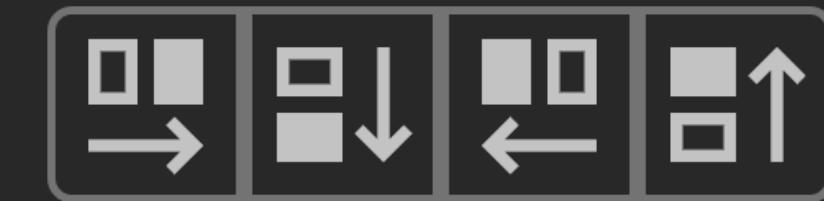
Performance  
Control



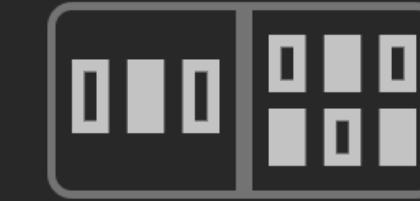
Layout  
Animations  
Interactions  
Accessibility



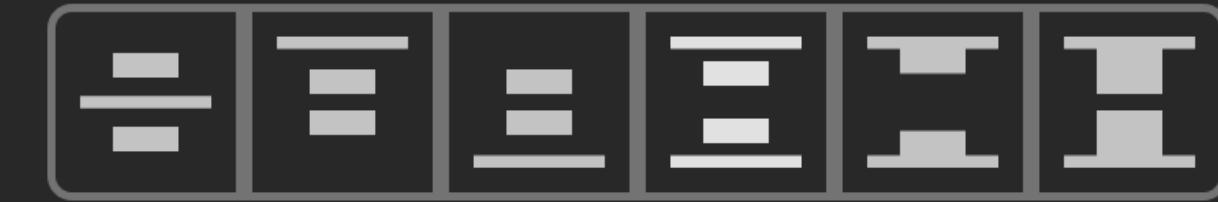
`flex-direction: row`



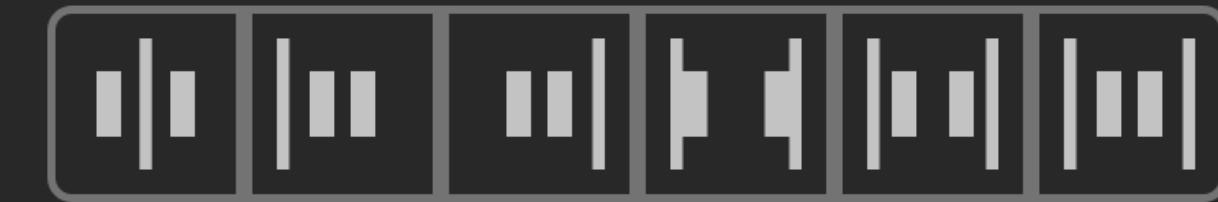
`flex-wrap: nowrap`



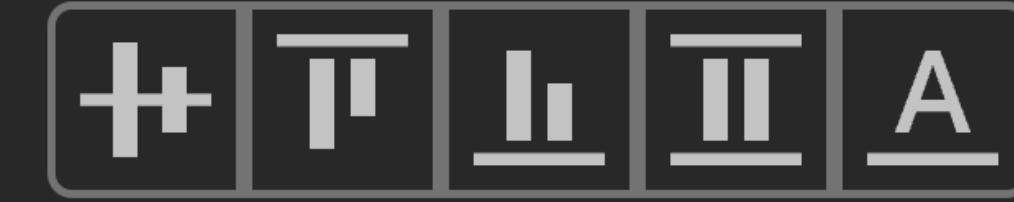
`align-content: normal`



`justify-content: normal`



`align-items: normal`

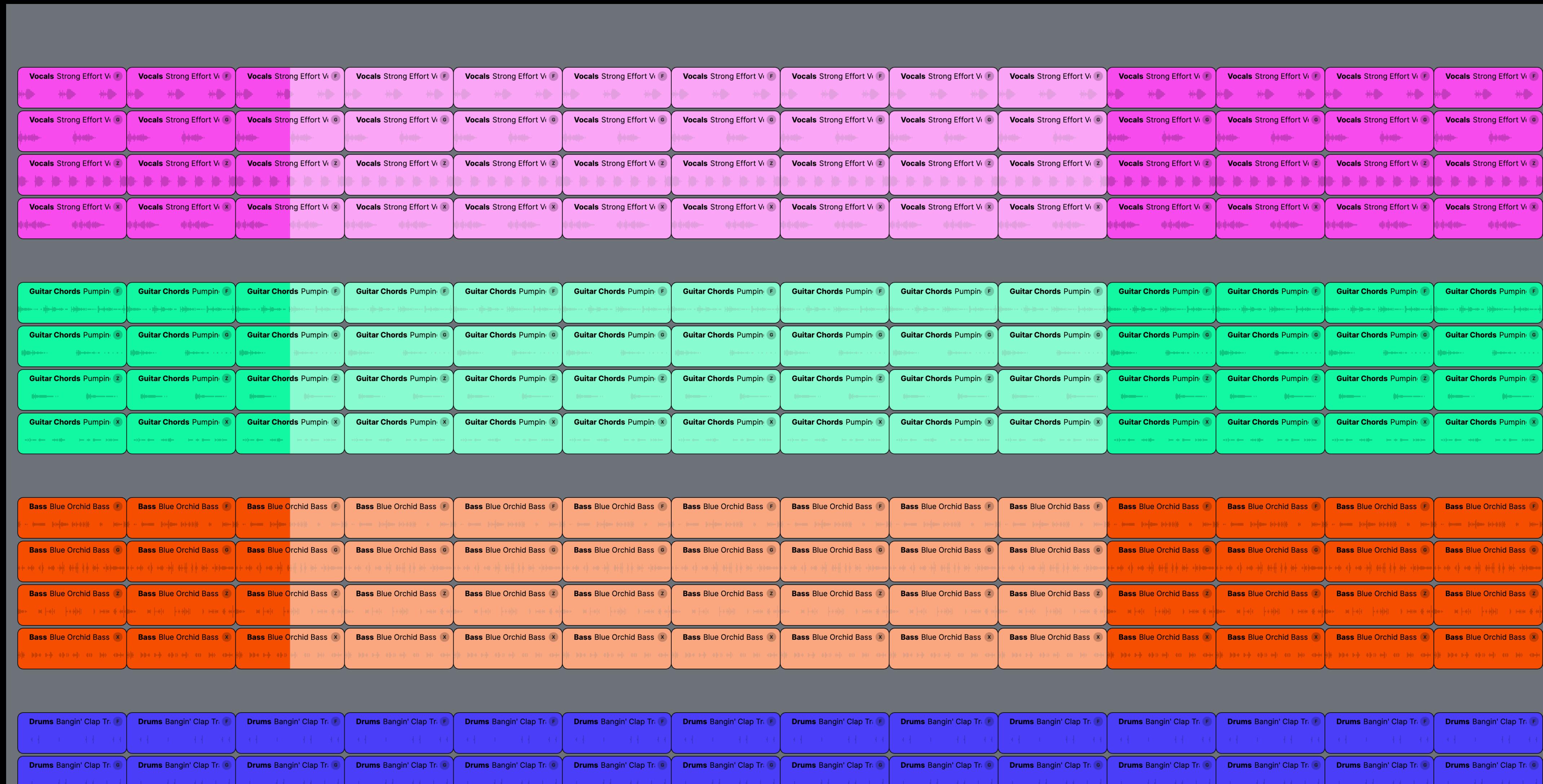


"It is very unlikely in any given situation  
that one thing exists in a vacuum.

The most common case is I have multiple,  
so I may as well transform them together"

**Mike Acton**

CppCon 2014: "Data-Oriented Design and C++" 37m 30s  
<https://www.youtube.com/watch?v=rXOltVEVjHc>



# **Slow: individual rendering**

```
const renderEvent = (ctx: CanvasRenderingContext2D, event: TimelineEvent) => {
  >  { ...
  | }
};

export const renderEvents = (ctx: CanvasRenderingContext2D, events: TimelineEvent[]) => {
  for (const event of events) {
    renderEvent(ctx, event);
  }
};
```

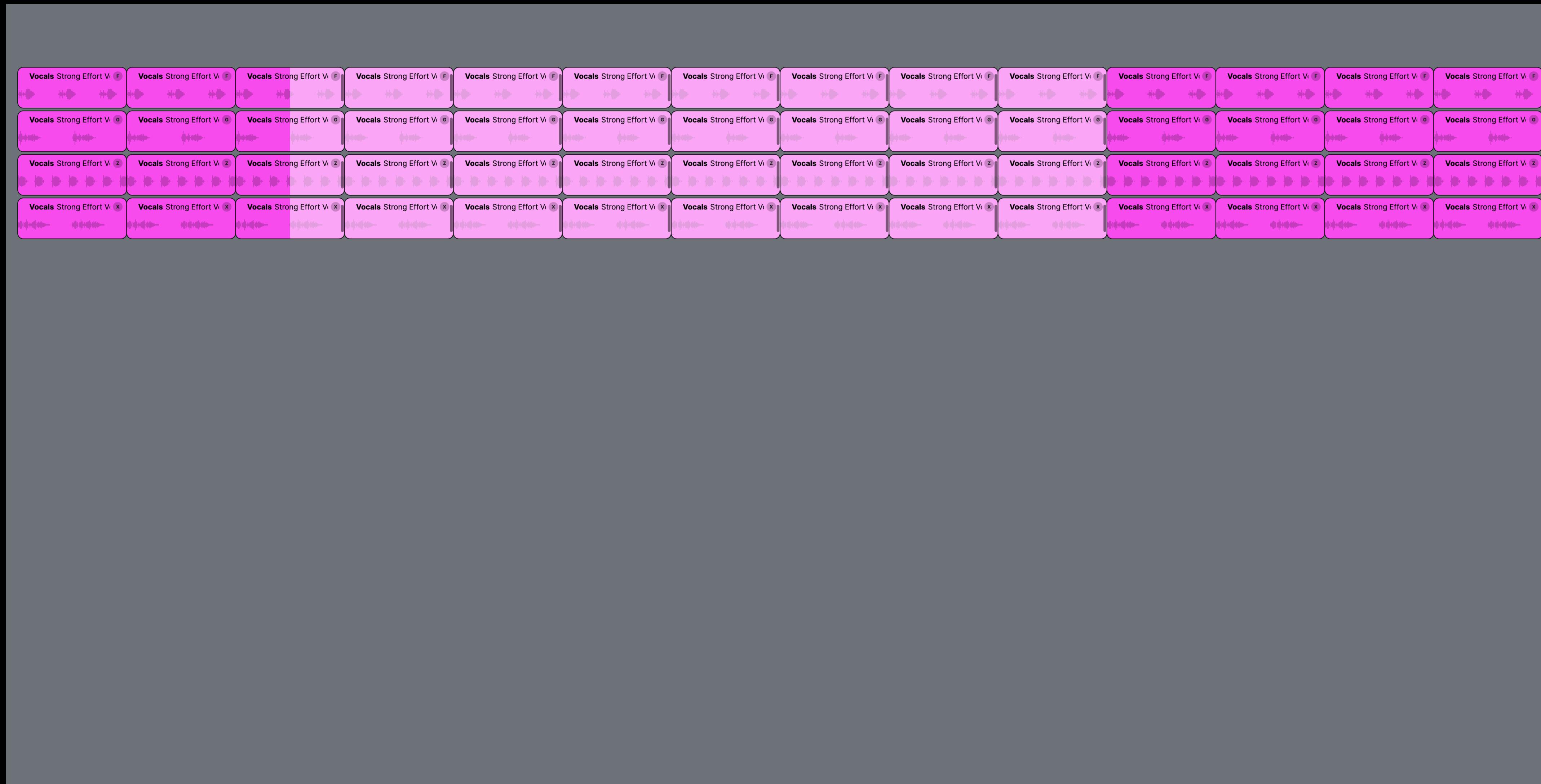


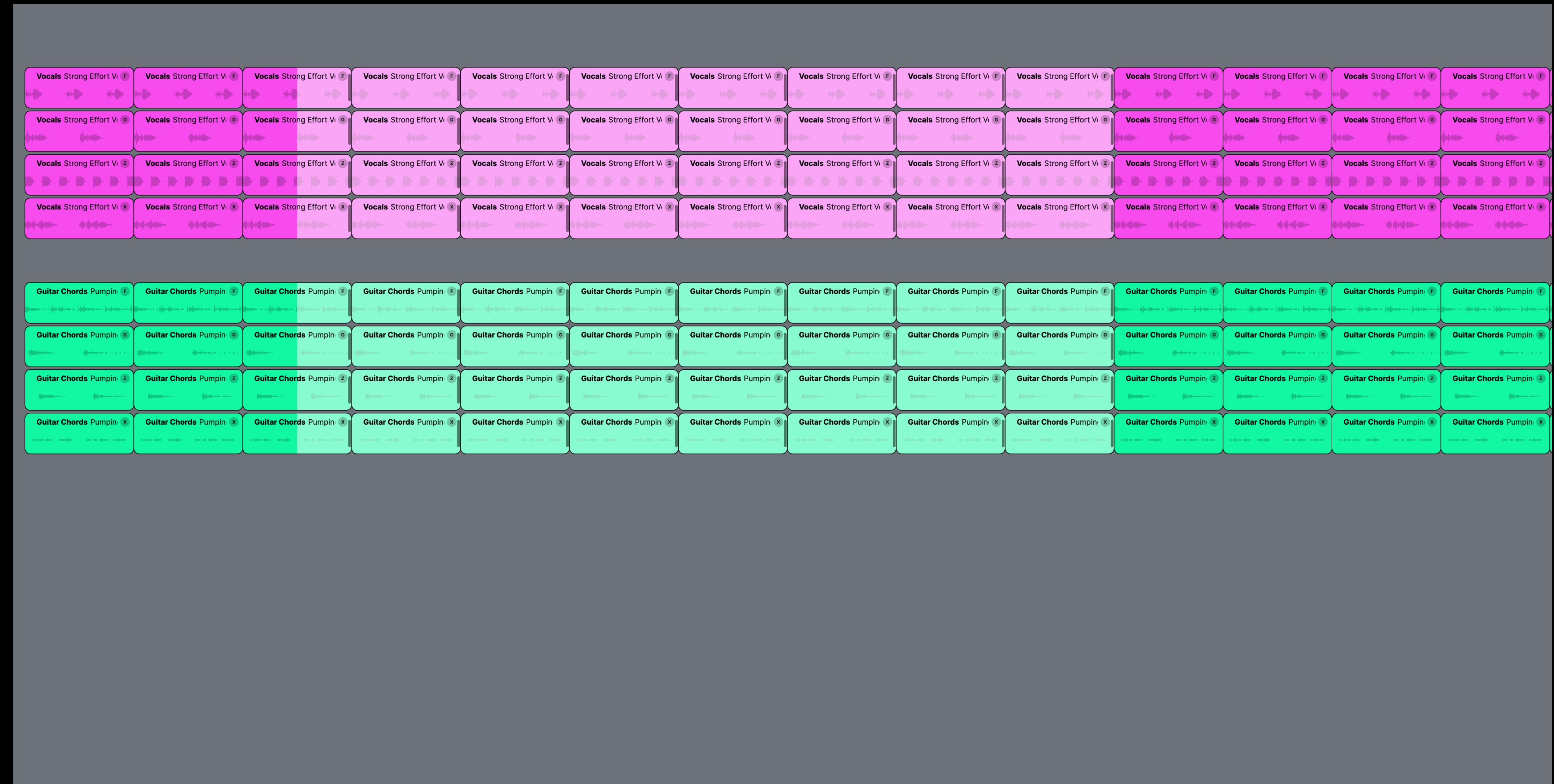




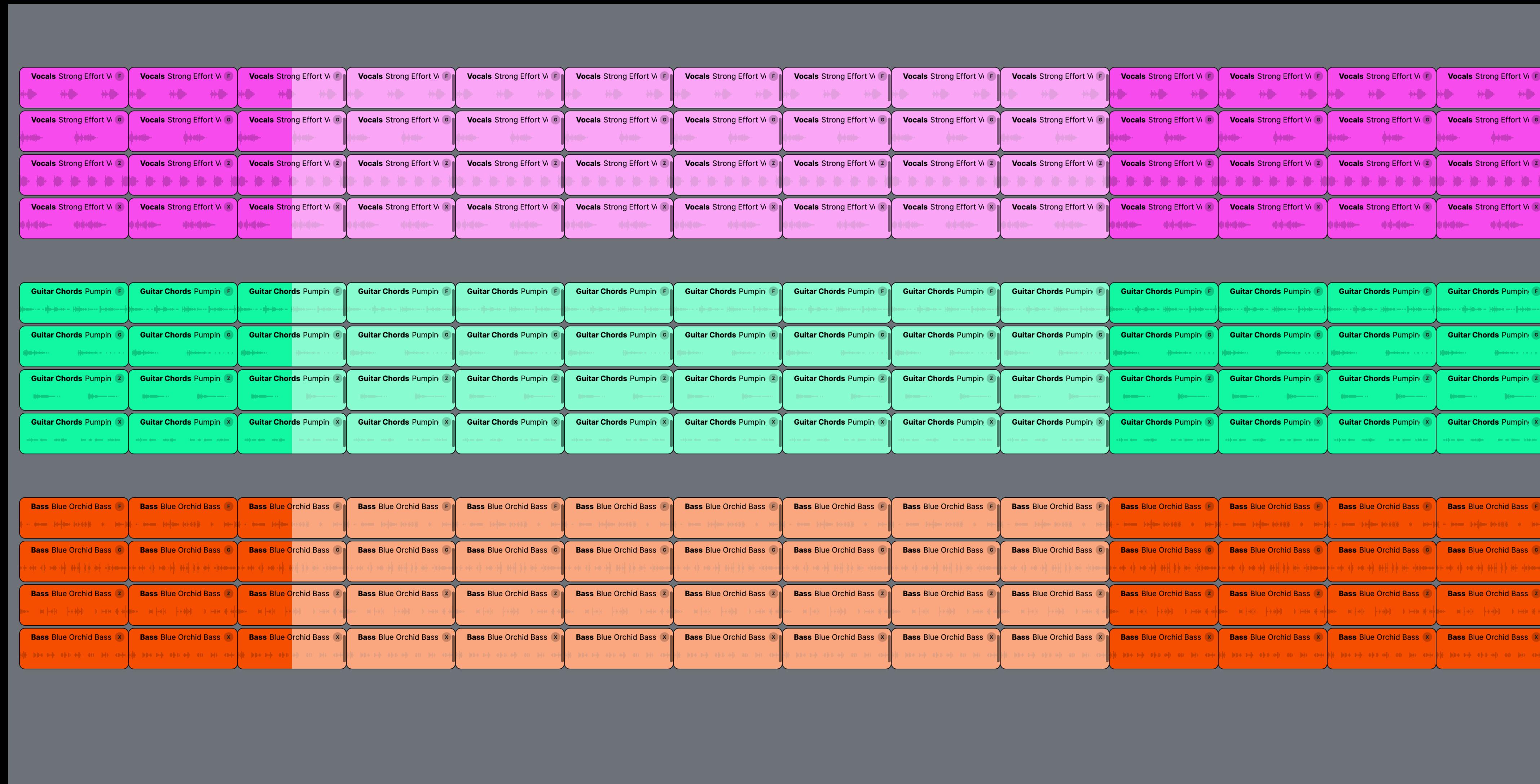








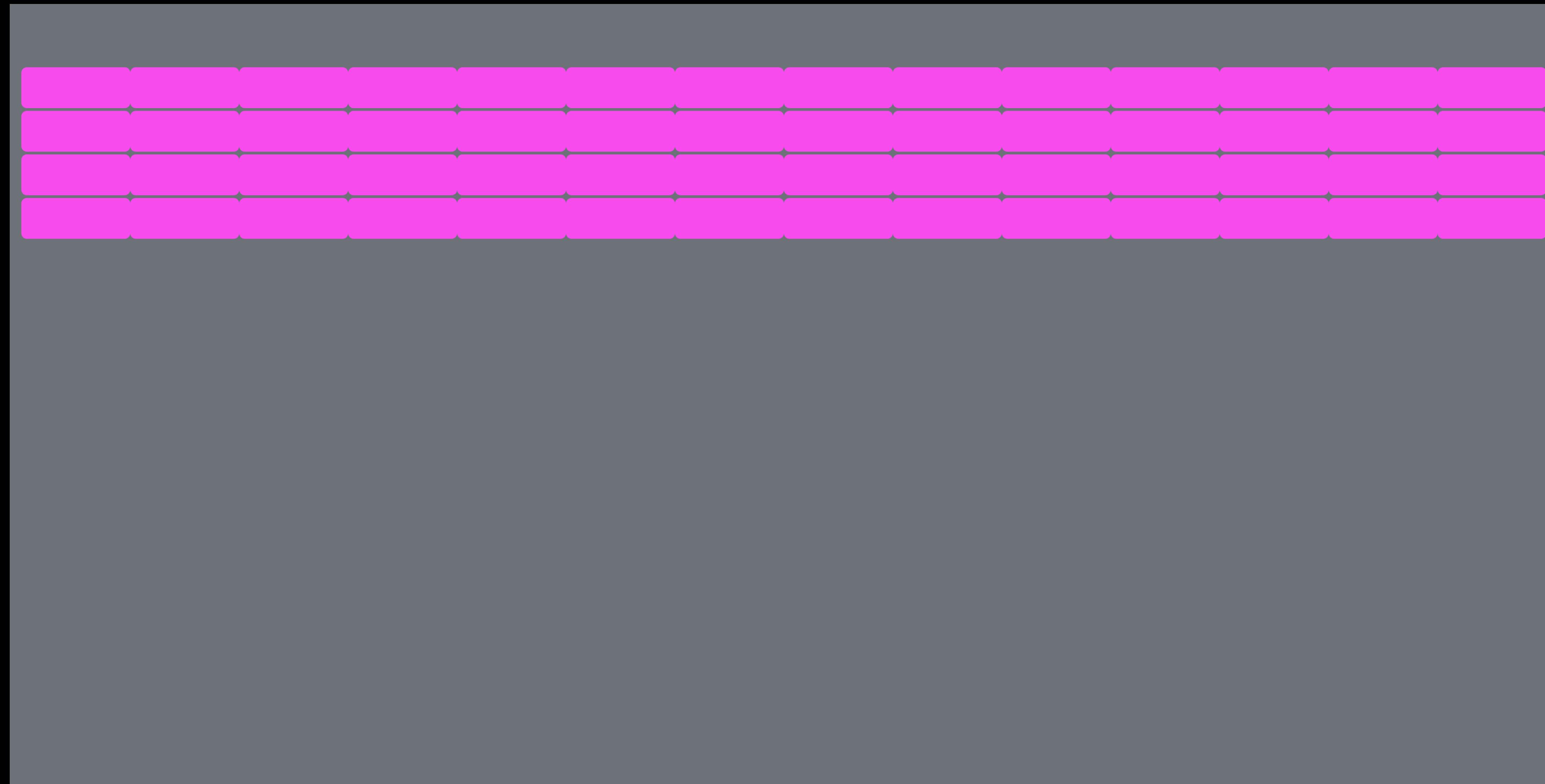
# OUTPUT



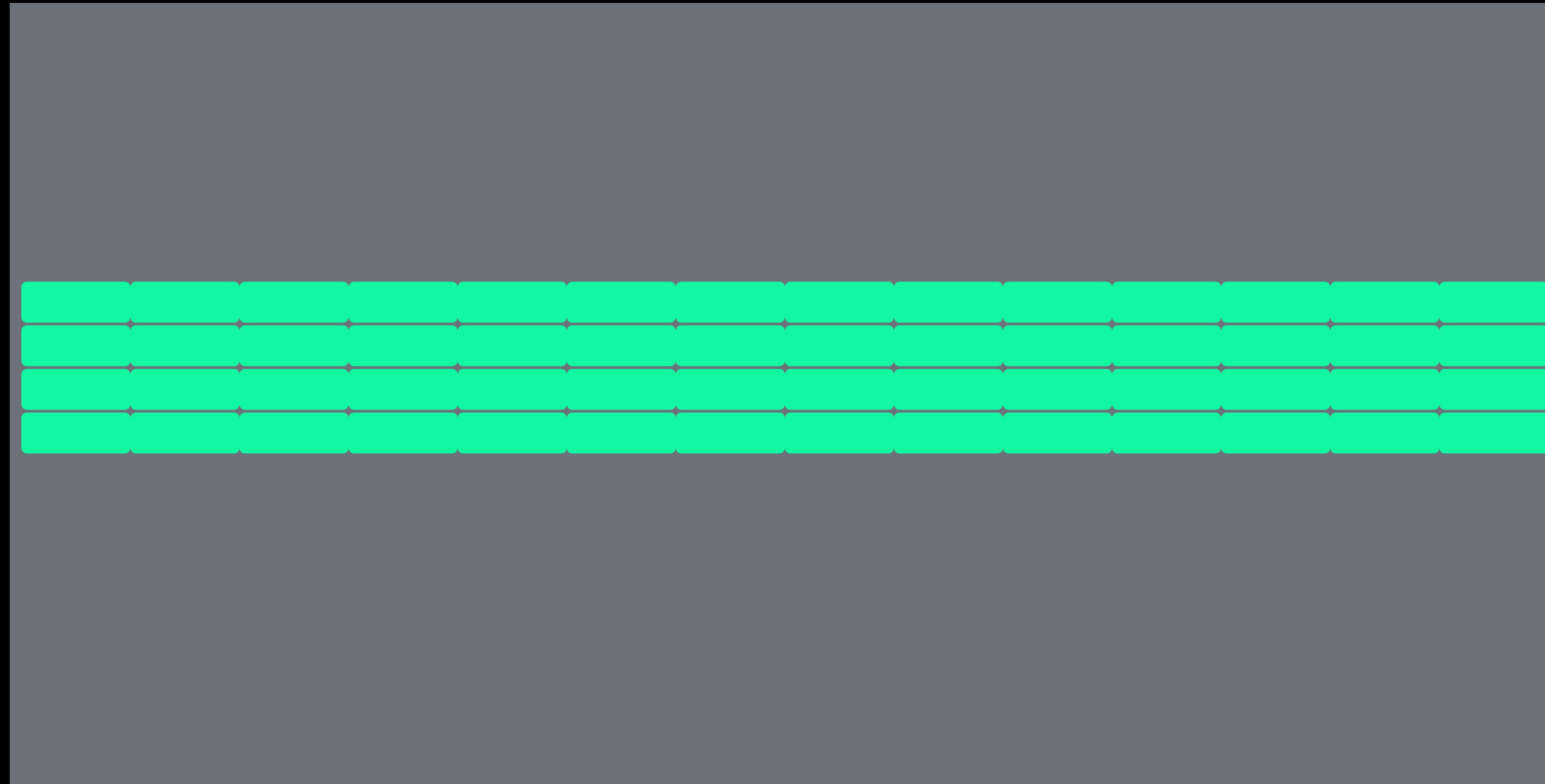


# Fast: batch rendering

## OUTPUT



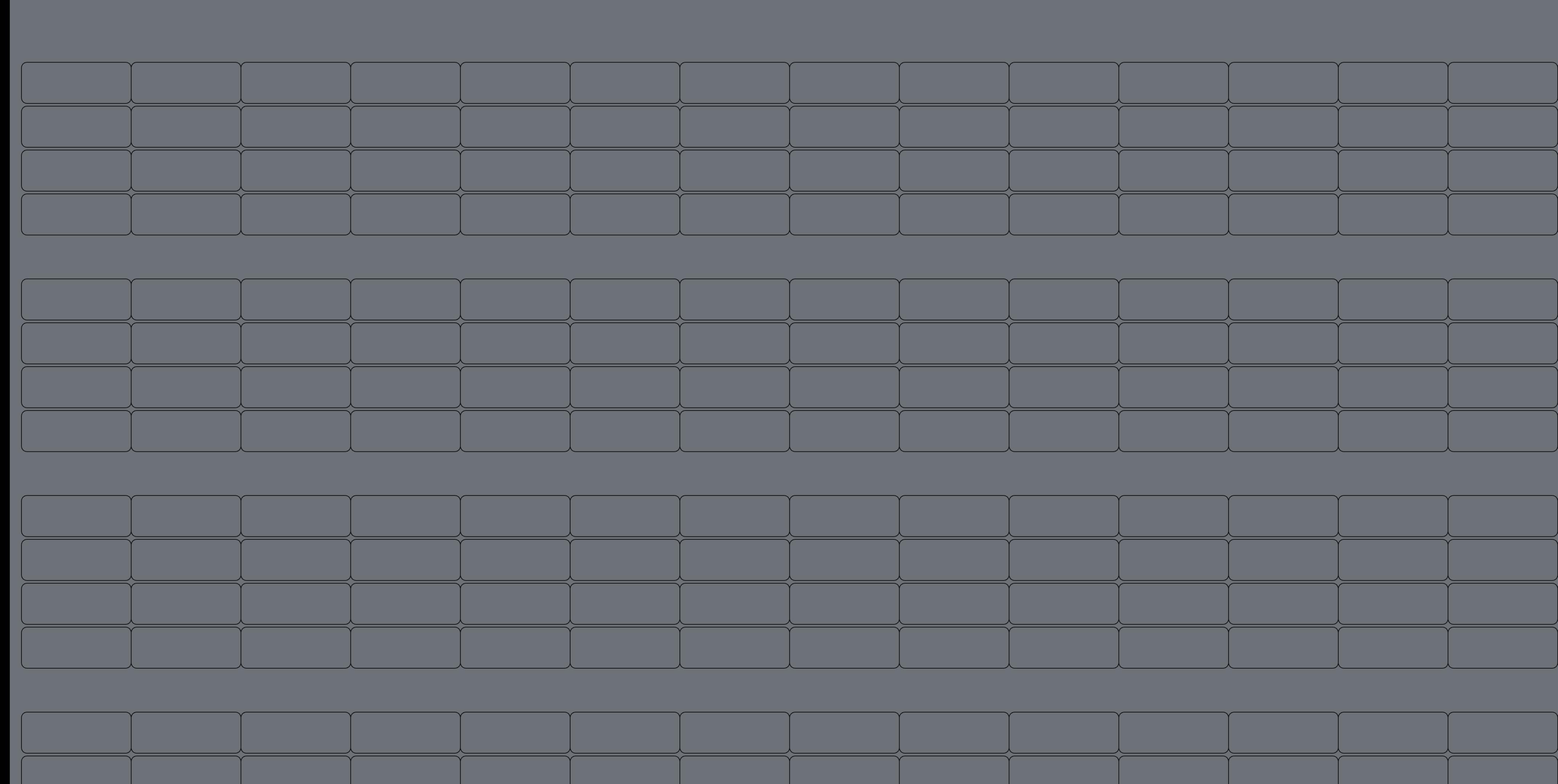
## OUTPUT

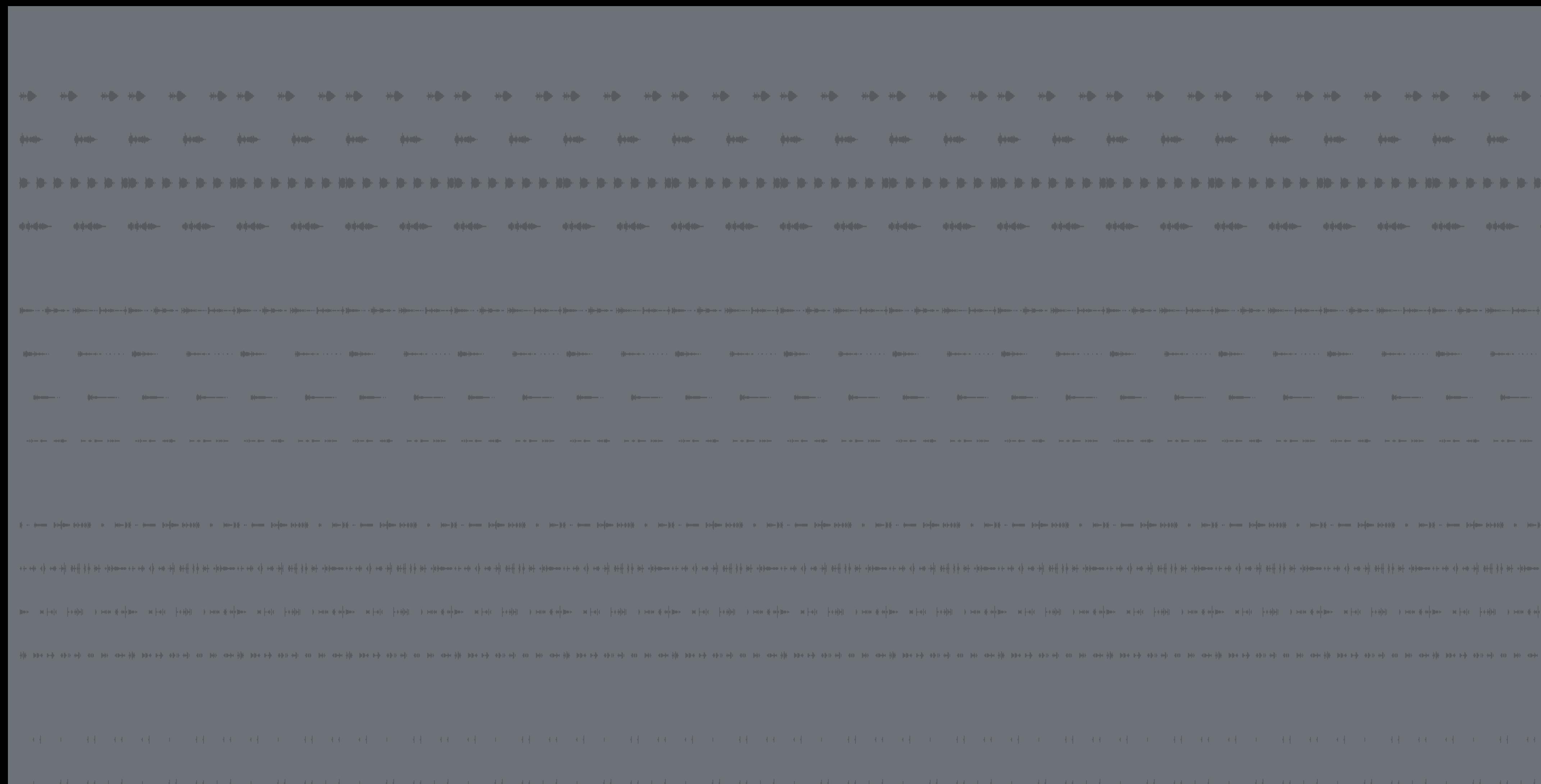




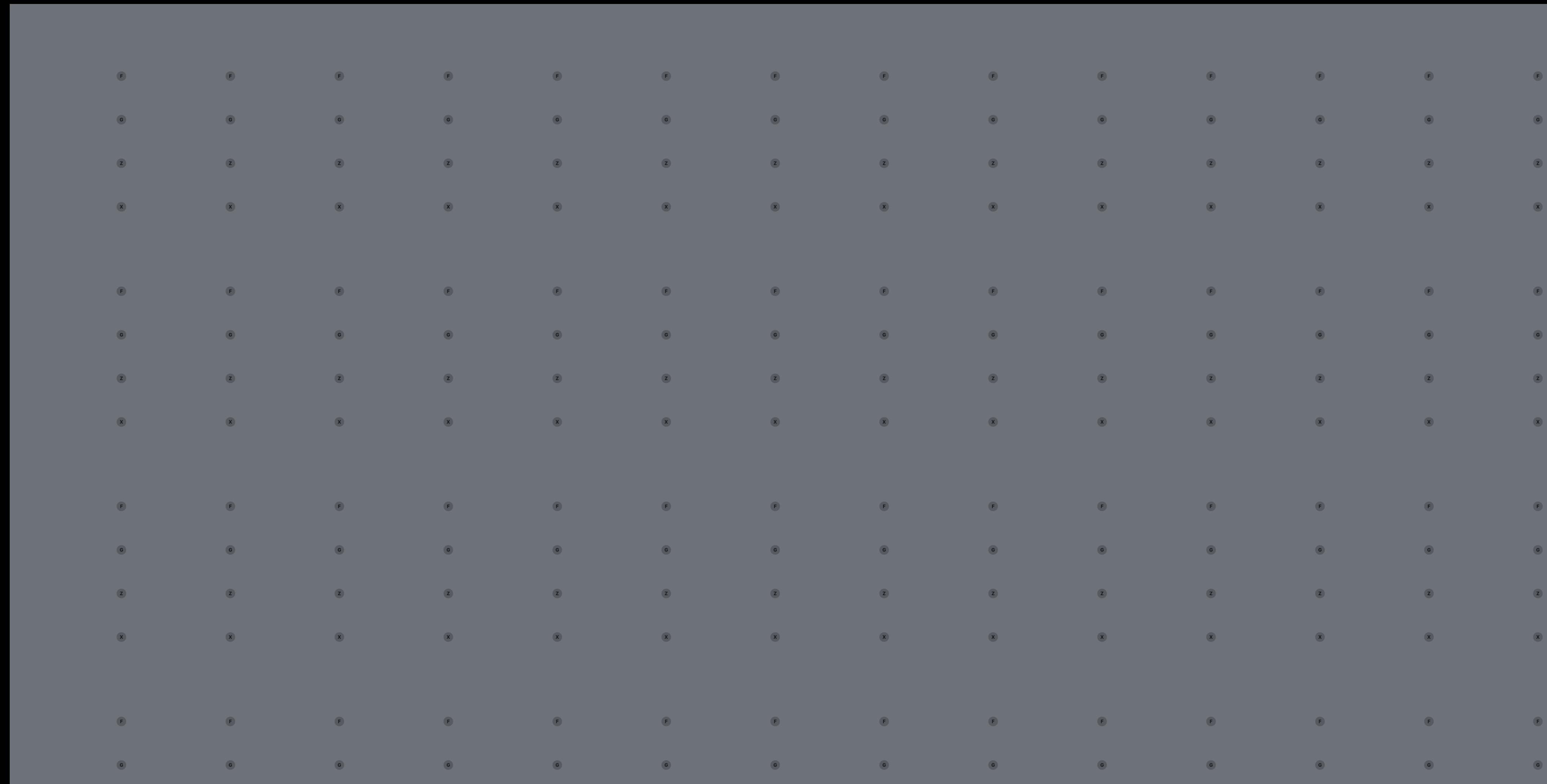
**OUTPUT**



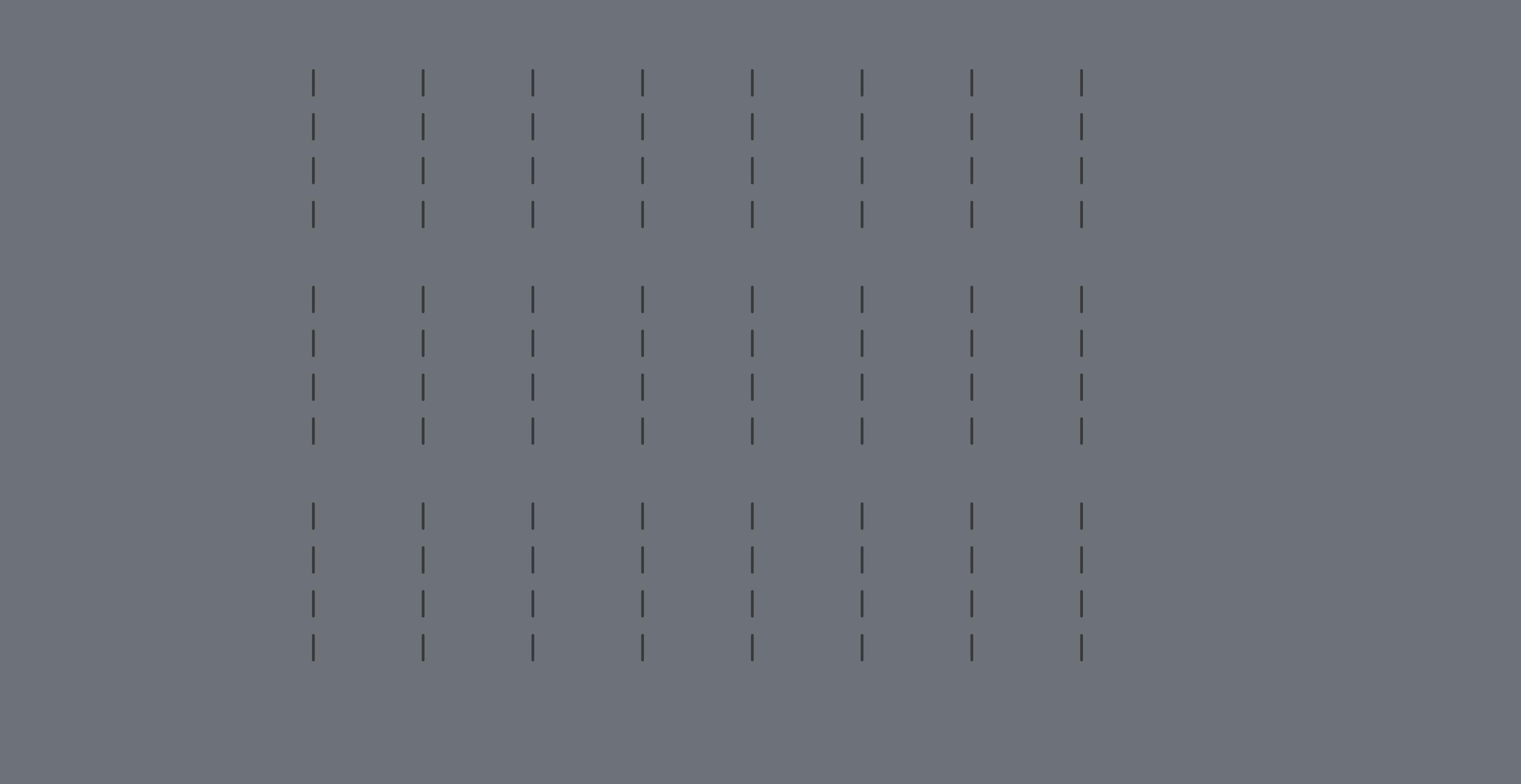




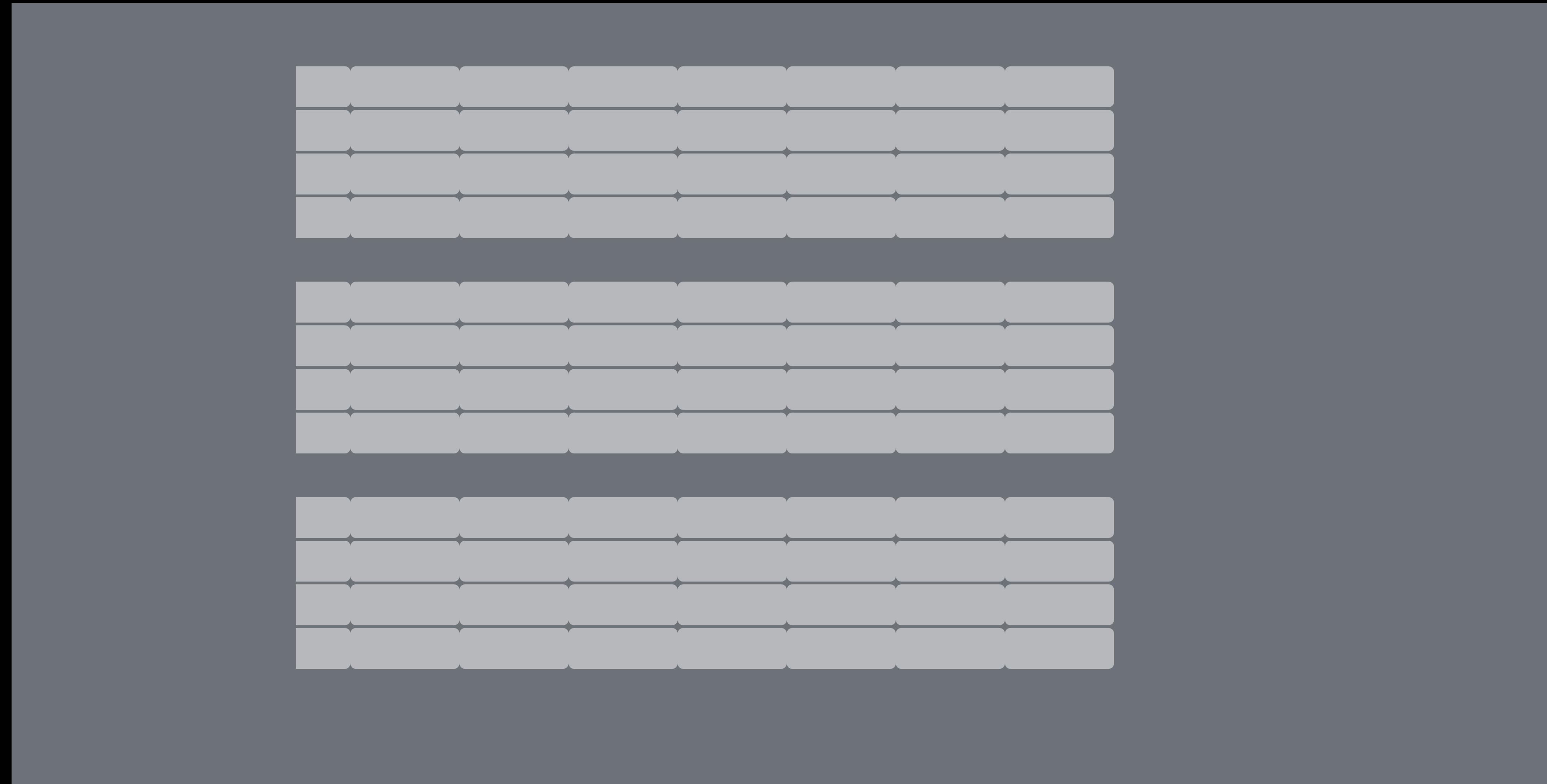
# OUTPUT

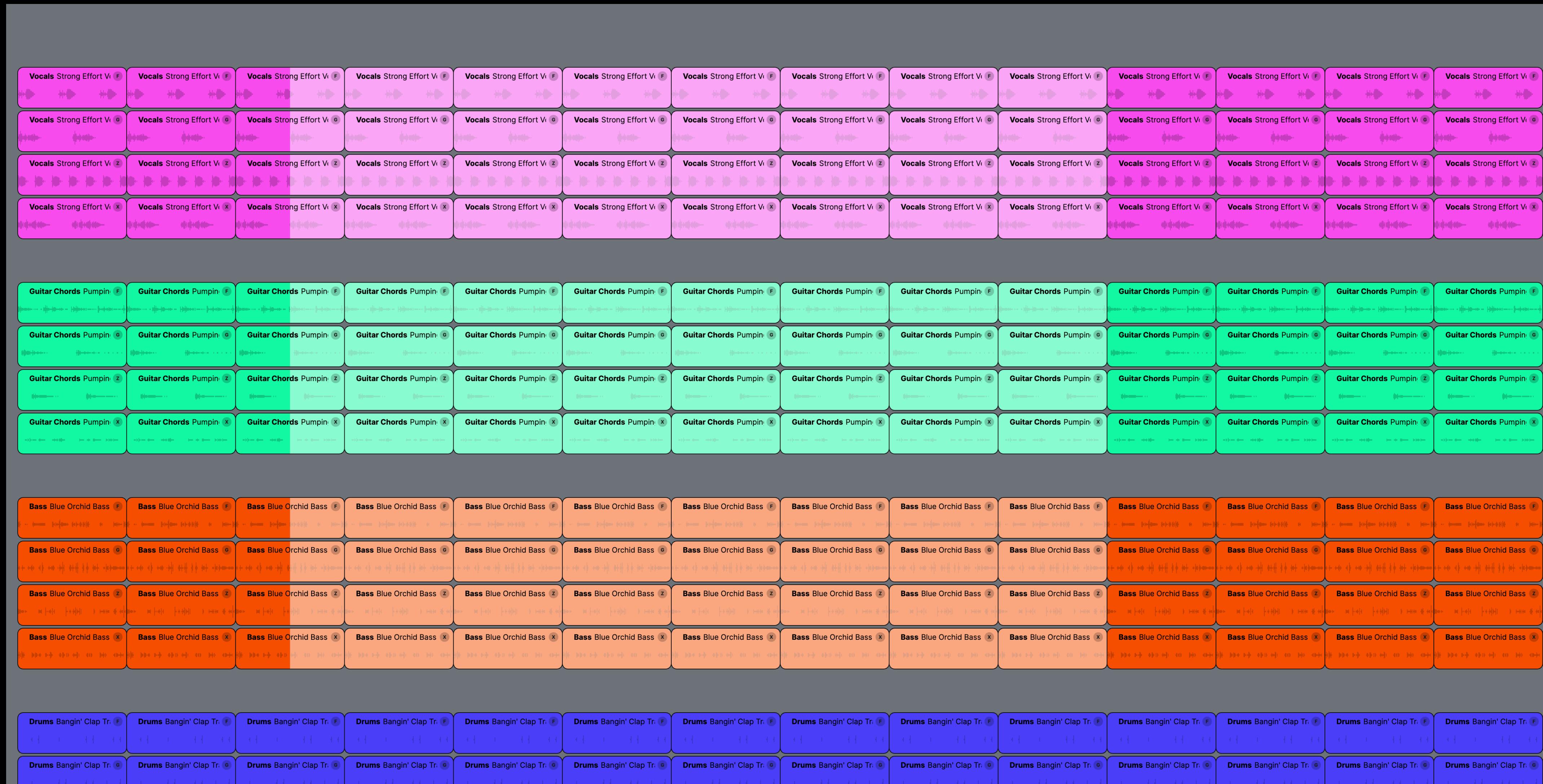


**OUTPUT**



## OUTPUT





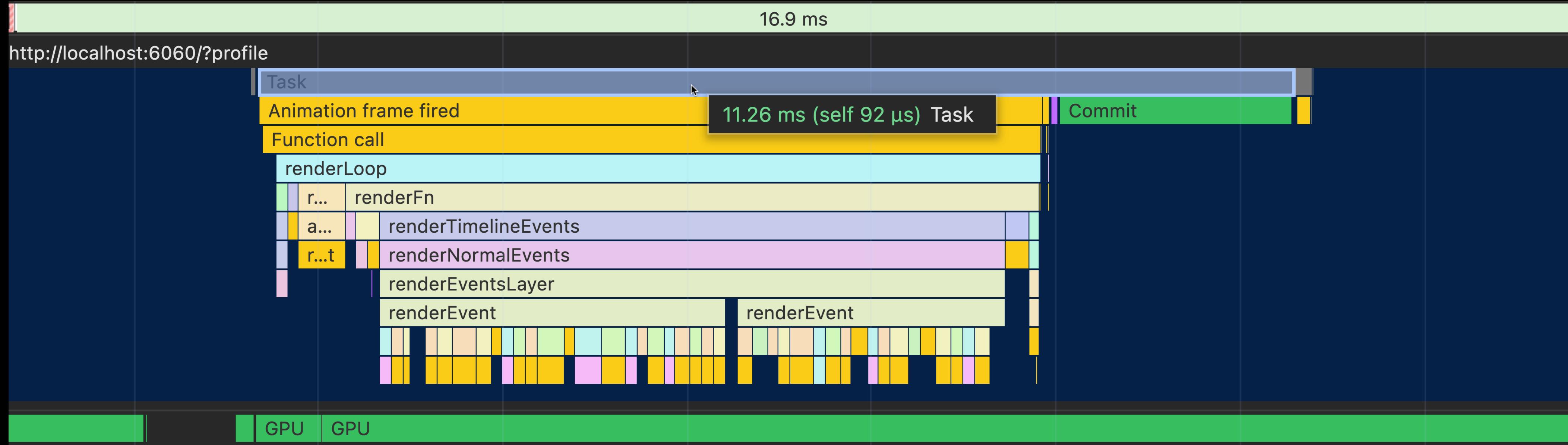
```
const renderEvents = (ctx: CanvasRenderingContext2D, events: TimelineEvent[]) => {
    // ----- 1. create paths and view models -----
    const allEventsPath = new Path2D();
    // [...]

    // ----- 2. shared calculations -----
    const pixelsPerBeat = calculatePixelsPerBeat(zoomLevel);
    // [...]

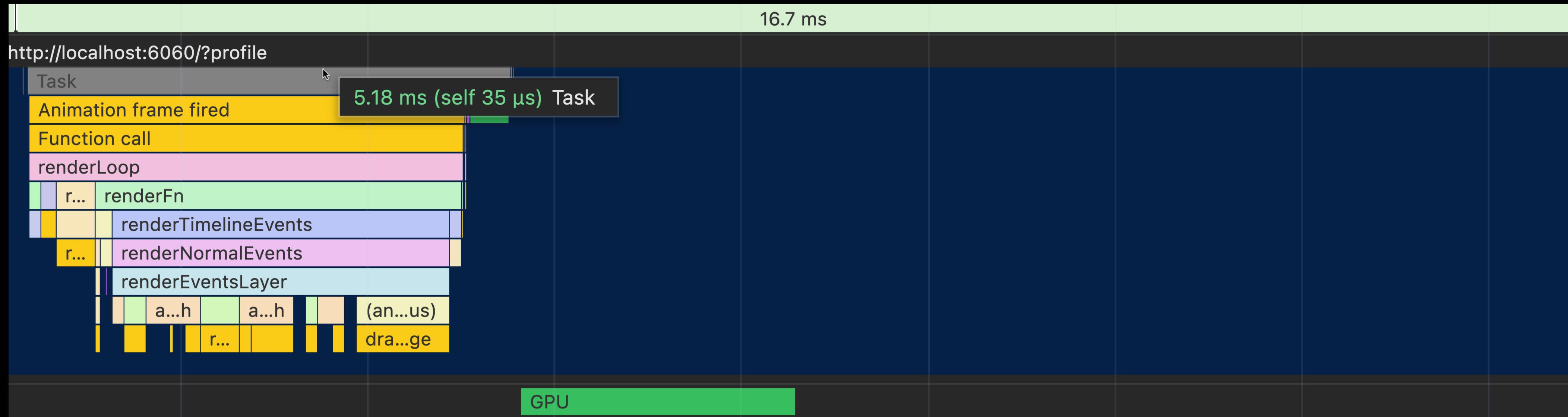
    // ----- 3. add to paths and view models -----
    for (const event of events) {
        const bounds = createEventBounds(event, pixelsPerBeat);
        allEventsPath.roundRect(bounds.x, bounds.y, bounds.w, bounds.h, kEventRadii_px);
        // [...]
    }

    // ----- 4. rendering -----
    ctx.fillStyle = "pink";
    ctx.fill(allEventsPath);
    // [...]
};
```

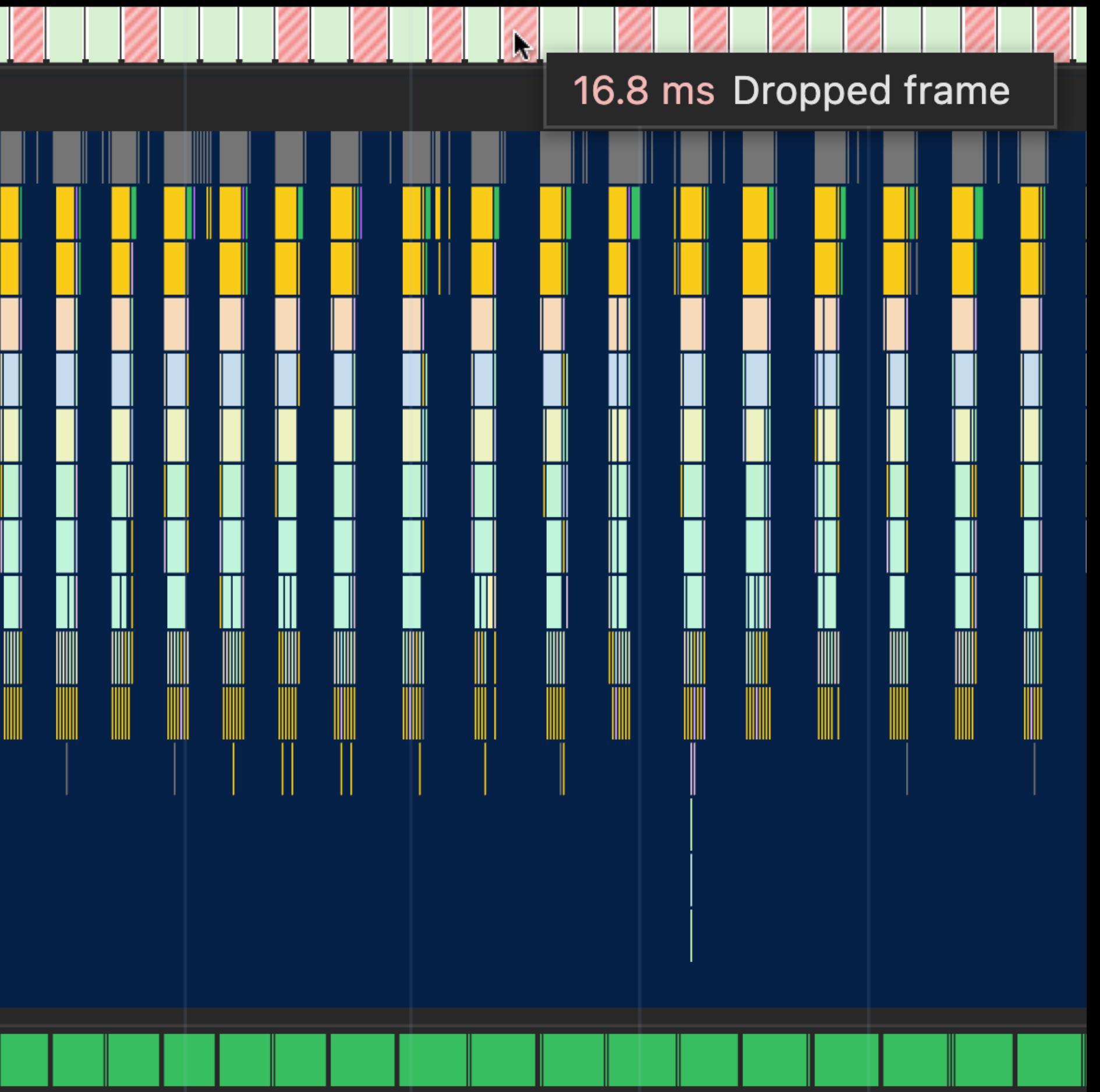
**Slow**  
**11.26ms**



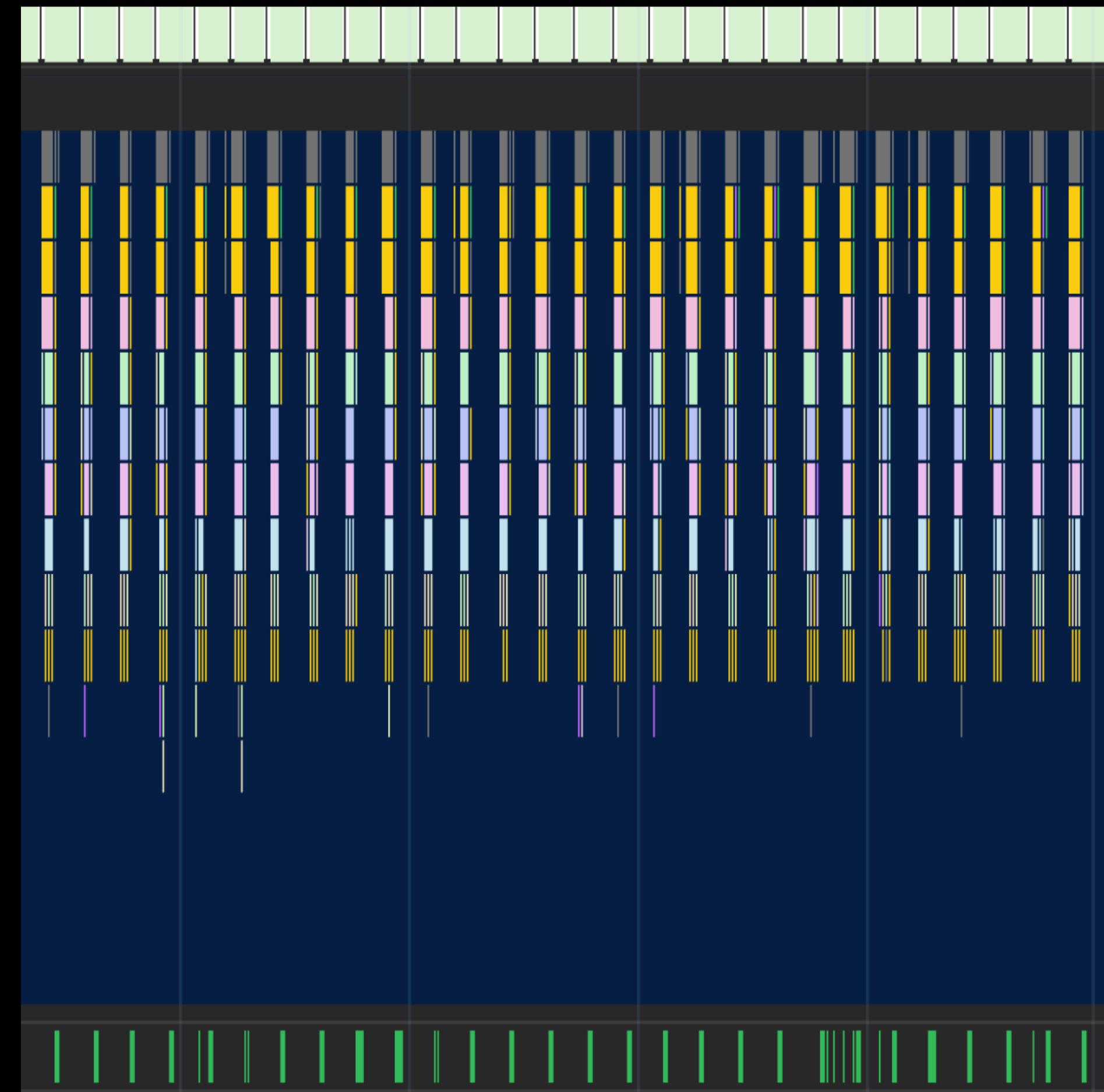
**Fast**  
**5.18ms**  
**(2.2x)**



# Slow



# Fast



# Why?

# Changing global properties

```
// event 1  
ctx.fillStyle = "pink";
```

```
// event 2  
ctx.fillStyle = "blue";
```

```
// event 3  
ctx.fillStyle = "pink";
```

# JavaScript

ctx.fillStyle = "pink";



# C++ (64 LOC)

```
void BaseRenderingContext2D::setFillStyle(v8::Isolate* isolate,
                                         v8::Local<v8::Value> value,
                                         ExceptionState& exception_state) {
    ValidateStateStack();

    CanvasRenderingContext2DState& state = GetState();
    // This block is similar to that in setStrokeStyle(), see comments there for
    // details on this.
    if (value->IsString()) {
        v8::Local<v8::String> v8_string = value.As<v8::String>();
        UpdateIdentifiabilityStudyBeforeSettingStrokeOrFill(
            v8_string, CanvasOps::kSetFillStyle);
        if (state.IsUnparsedFillColor(v8_string)) {
            return;
        }
        Color parsed_color = Color::kTransparent;
        if (!ExtractColorFromV8StringAndUpdateCache(
                isolate, v8_string, exception_state, parsed_color)) {
            return;
        }
        if (state.FillStyle().IsEquivalentColor(parsed_color)) {
            state.SetUnparsedFillColor(isolate, v8_string);
            return;
        }
        state.SetFillColor(parsed_color);
        state.ClearUnparsedFillColor();
        state.ClearResolvedFilter();
        return;
    }
    V8CanvasStyle v8_style;
    if (!ExtractV8CanvasStyle(isolate, value, v8_style, exception_state))
        return;

    UpdateIdentifiabilityStudyBeforeSettingStrokeOrFill(v8_style,
                                                       CanvasOps::kSetFillStyle);

    switch (v8_style.type) {
        case V8CanvasStyleType::kCSSColorValue:
            state.SetFillColor(v8_style.css_color_value);
            break;
        case V8CanvasStyleType::kGradient:
            state.SetFillGradient(v8_style.gradient);
            break;
        case V8CanvasStyleType::kPattern:
            if (!origin_tainted_by_content_ && !v8_style.pattern->OriginClean())
                SetOriginTaintedByContent();
            state.SetFillPattern(v8_style.pattern);
            break;
        case V8CanvasStyleType::kString: {
            Color parsed_color = Color::kTransparent;
            if (ParseColorOrCurrentColor(v8_style.string, parsed_color) ==
                ColorParseResult::kParseFailed) {
                return;
            }
            if (!state.FillStyle().IsEquivalentColor(parsed_color)) {
                state.SetFillColor(parsed_color);
            }
            break;
        }
    }
}
```

# Drawing multiple paths

```
// event 1  
ctx.fill(eventPath1);
```

```
// event 2  
ctx.fill(eventPath2);
```

```
// event 3  
ctx.fill(eventPath3);
```

# JavaScript

# C++ (91 LOC)

ctx.fill(eventPath1);



```
void BaseRenderingContext2D::DrawPathInternal(
    const CanvasPath& path,
    CanvasRenderingContext2DState::PaintType paint_type,
    SkPathFillType fill_type,
    UsePaintCache use_paint_cache) {
    if (!path.IsEmpty()) {
        if (!path.IsEmpty()) {
            return;
        }

        gfx::RectF bounds(path.BoundingRect());
        if (std::isnan(bounds.x()) || std::isnan(bounds.y()) ||
            std::isnan(bounds.width()) || std::isnan(bounds.height()))
            return;

        if (paint_type == CanvasRenderingContext2DState::kStrokePaintType)
            InflateStrokeRect(bounds);

        if (path.IsLine()) {
            if (paint_type == CanvasRenderingContext2DState::kFillPaintType)
                [unlikely] {
                    // Filling a line is a no-op.
                    // Also, SKCanvas::drawLine() ignores paint type and always strokes.
                    return;
                }
            auto line = path.line();
            Draw<OverdrawOp::kNone>(
                [line](cc::PaintCanvas* c,
                       const cc::PaintFlags* flags) // draw lambda
                {
                    c->drawLine(SkFloatToScalar(line.start.x()),
                                 SkFloatToScalar(line.start.y()),
                                 SkFloatToScalar(line.end.x()),
                                 SkFloatToScalar(line.end.y()), *flags);
                },
                [](const SkIRect& rect) // overdraw test lambda
                { return false; },
                bounds, paint_type,
                GetState().HasPattern(paint_type)
                    ? CanvasRenderingContext2DState::kNonOpaqueImage
                    : CanvasRenderingContext2DState::kNoImage,
                CanvasPerformanceMonitor::DrawType::kPath);
            return;
        }

        if (path.IsArc()) {
            const auto& arc = path.arc();
            const SkScalar x = WebCoreFloatToSkScalar(arc.x);
            const SkScalar y = WebCoreFloatToSkScalar(arc.y);
            const SkScalar radius = WebCoreFloatToSkScalar(arc.radius);
            const SkScalar diameter = radius + radius;
            const SkRect oval =
                SkRect::MakeXYWH(x - radius, y - radius, diameter, diameter);
            const SkScalar start_degrees =
                WebCoreFloatToSkScalar(arc.start_angle_radians * 180 / kPiFloat);
            const SkScalar sweep_degrees =
                WebCoreFloatToSkScalar(arc.sweep_angle_radians * 180 / kPiFloat);
            const bool closed = arc.closed;
            Draw<OverdrawOp::kNone>(
                [oval, start_degrees, sweep_degrees, closed](cc::PaintCanvas* c,
                                                               const cc::PaintFlags* flags) // draw lambda
                {
                    cc::PaintFlags arc_paint_flags(*flags);
                    arc_paint_flags.setArcClosed(closed);
                    c->drawArc(oval, start_degrees, sweep_degrees, arc_paint_flags);
                },
                [](const SkIRect& rect) // overdraw test lambda
                { return false; },
                bounds, paint_type,
                GetState().HasPattern(paint_type)
                    ? CanvasRenderingContext2DState::kNonOpaqueImage
                    : CanvasRenderingContext2DState::kNoImage,
                CanvasPerformanceMonitor::DrawType::kPath);
            return;
        }

        SkPath sk_path = path.getPath().GetSkPath();
        sk_path.setFillType(fill_type);

        Draw<OverdrawOp::kNone>(
            [sk_path, use_paint_cache](cc::PaintCanvas* c,
                                       const cc::PaintFlags* flags) // draw lambda
            {
                c->drawPath(sk_path, *flags, use_paint_cache);
            },
            [](const SkIRect& rect) // overdraw test lambda
            { return false; },
            bounds, paint_type,
            GetState().HasPattern(paint_type)
                ? CanvasRenderingContext2DState::kNonOpaqueImage
                : CanvasRenderingContext2DState::kNoImage,
            CanvasPerformanceMonitor::DrawType::kPath);
        }
    }
}
```

# JavaScript

```
// slow
for (const event of oneThousandEvents) {
    const path = new Path2D();
    path.roundRect(event.x, event.y, event.w, event.h, 10);
    ctx.fillStyle = "pink";
    ctx.fill(path);
}
```

```
// fast
const path = new Path2D();
for (const event of oneThousandEvents) {
    path.roundRect(event.x, event.y, event.w, event.h, 10);
}
ctx.fillStyle = "pink";
ctx.fill(path);
```

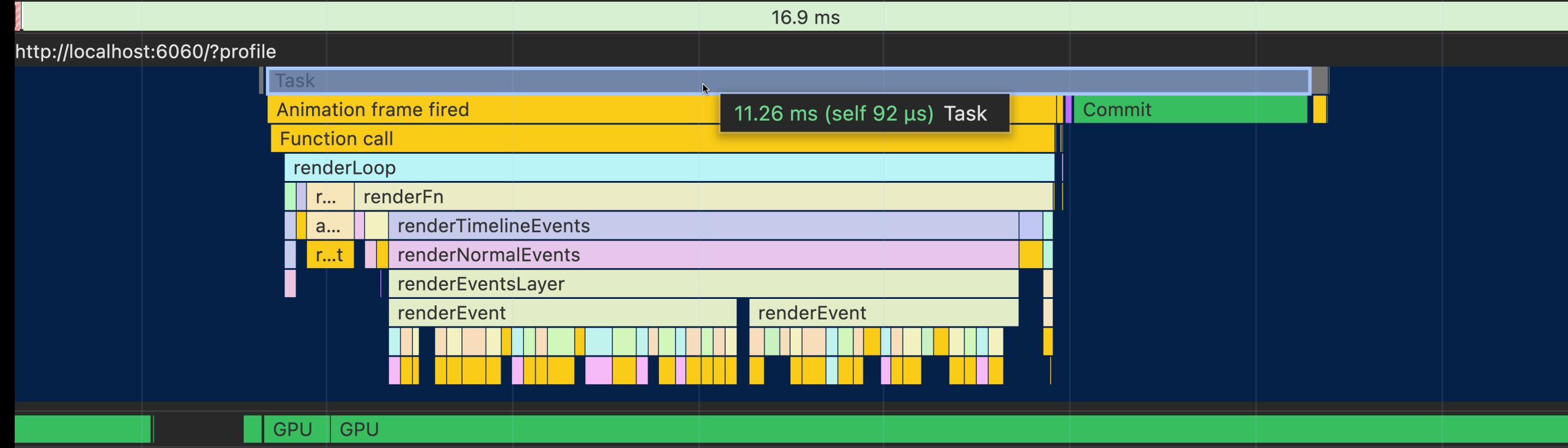
# C++

$$\begin{aligned}(1 + 133 + 64 + 91) \times 1000 \\= 289,000 \text{ LOC}\end{aligned}$$

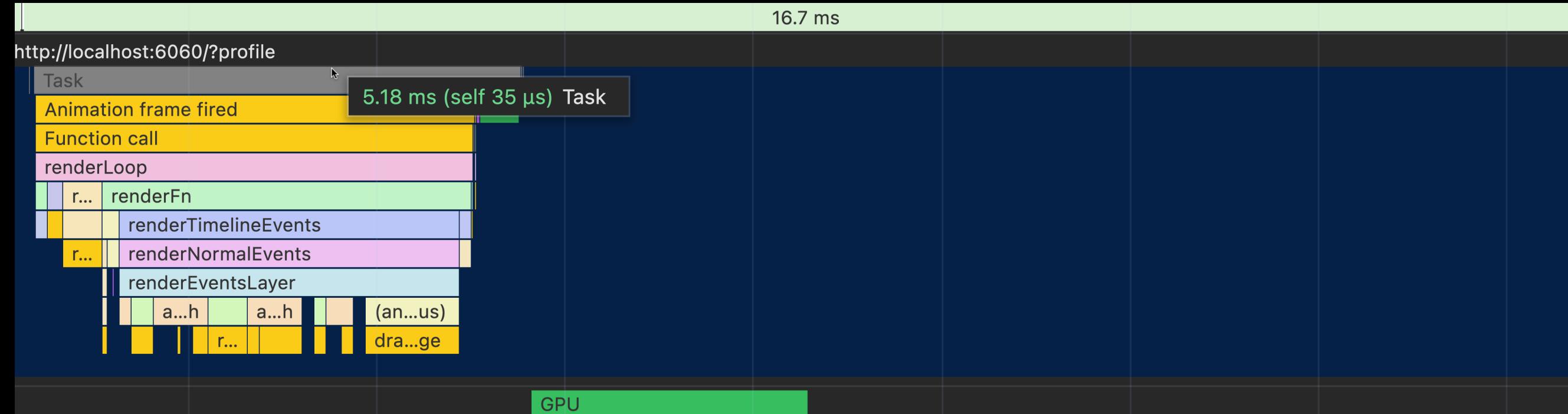
$$\begin{aligned}1 + (133 \times 1000) + 64 + 91 \\= 133,156 \text{ LOC}\end{aligned}$$

$$289,000 / 133,156 = 2.17x$$

Slow  
11.26ms



Fast  
5.18ms  
(2.2x)



# Inlining

```
const renderEvent = (ctx: CanvasRenderingContext2D, event: TimelineEvent) => {
  >  { ...
  | }
};

export const renderEvents = (ctx: CanvasRenderingContext2D, events: TimelineEvent[]) => {
  for (const event of events) {
    renderEvent(ctx, event);
  }
};
```

```
const renderEvent = (ctx: CanvasRenderingContext2D, event: TimelineEvent) => {
    // calculate bounds
    const bounds: Rect = {
        x: calculateX(eventStart, pixelsPerBeat, scroll_px.x),
        y: calculateY(event, voices, scroll_px.y),
        w: calculateWidth(eventDuration, pixelsPerBeat),
        h: kVoiceHeight_px,
    };

    // render background
    {
        ctx.beginPath();
        ctx.roundRect(bounds.x, bounds.y, bounds.w, bounds.h, kEventRadii_px);
        ctx.fillStyle = "pink";
        ctx.fill();
    }

    // [...]
};
```

**OUTPUT**

```
// ----- event 1 -----
// calculate bounds 1
const bounds1: Rect = {
  x: calculateX(eventStart, pixelsPerBeat, scroll_px.x),
  y: calculateY(event, voices, scroll_px.y),
  w: calculateWidth(eventDuration, pixelsPerBeat),
  h: kVoiceHeight_px,
};

// render background 1
{
  ctx.beginPath();
  ctx.roundRect(bounds1.x, bounds1.y, bounds1.w, bounds1.h, kEventRadii_px);
  ctx.fillStyle = "pink";
  ctx.fill();
}

// ----- event 2 -----
// calculate bounds 2
const bounds2: Rect = {
  x: calculateX(eventStart, pixelsPerBeat, scroll_px.x),
  y: calculateY(event, voices, scroll_px.y),
  w: calculateWidth(eventDuration, pixelsPerBeat),
  h: kVoiceHeight_px,
};

// render background 2
{
  ctx.beginPath();
  ctx.roundRect(bounds2.x, bounds2.y, bounds2.w, bounds2.h, kEventRadii_px);
  ctx.fillStyle = "pink";
  ctx.fill();
}
```

## OUTPUT

```
// ----- event 1 -----
// calculate bounds 1
const bounds1: Rect = {
  x: calculateX(eventStart, pixelsPerBeat, scroll_px.x),
  y: calculateY(event, voices, scroll_px.y),
  w: calculateWidth(eventDuration, pixelsPerBeat),
  h: kVoiceHeight_px,
};
```

```
// render background 1
{
  ctx.beginPath();
  ctx.roundRect(bounds1.x, bounds1.y, bounds1.w, bounds1.h, kEventRadii_px);
  ctx.fillStyle = "pink";
  ctx.fill();
}
```

```
// ----- event 2 -----
// calculate bounds 2
const bounds2: Rect = {
  x: calculateX(eventStart, pixelsPerBeat, scroll_px.x),
  y: calculateY(event, voices, scroll_px.y),
  w: calculateWidth(eventDuration, pixelsPerBeat),
  h: kVoiceHeight_px,
};
```

```
// render background 2
{
  ctx.beginPath();
  ctx.roundRect(bounds2.x, bounds2.y, bounds2.w, bounds2.h, kEventRadii_px);
  ctx.fillStyle = "pink";
  ctx.fill();
}
```

```
// ----- event 1 -----
{
    ctx.beginPath();
    ctx.roundRect(bounds1.x, bounds1.y, bounds1.w, bounds1.h, kEventRadii_px);
    ctx.fillStyle = "pink";
    ctx.fill();
}

// ----- event 2 -----
{
    ctx.beginPath();
    ctx.roundRect(bounds2.x, bounds2.y, bounds2.w, bounds2.h, kEventRadii_px);
    ctx.fillStyle = "pink";
    ctx.fill();
}
```

```
// ----- event 1 -----
{
    ctx.beginPath();
    ctx.roundRect(bounds1.x, bounds1.y, bounds1.w, bounds1.h, kEventRadii_px);
    ctx.fillStyle = "pink";
    ctx.fill();
}

// ----- event 2 -----
{
    ctx.beginPath();
    ctx.roundRect(bounds2.x, bounds2.y, bounds2.w, bounds2.h, kEventRadii_px);
    ctx.fillStyle = "pink";
    ctx.fill();
}
```

----- combined -----

```
ctx.beginPath();
ctx.roundRect(bounds1.x, bounds1.y, bounds1.w, bounds1.h, kEventRadii_px);
ctx.roundRect(bounds2.x, bounds2.y, bounds2.w, bounds2.h, kEventRadii_px);
ctx.fillStyle = "pink";
ctx.fill();
```

```
// ----- combined -----
{
    ctx.beginPath();
    ctx.roundRect(bounds1.x, bounds1.y, bounds1.w, bounds1.h, kEventRadii_px);
    ctx.roundRect(bounds2.x, bounds2.y, bounds2.w, bounds2.h, kEventRadii_px);
    ctx.fillStyle = "pink";
    ctx.fill();
}
```

```
ctx.beginPath();
for (const event of events) {
    const bounds: Rect = ...
    ctx.roundRect(bounds.x, bounds.y, bounds.w, bounds.h, kEventRadii_px);
}
ctx.fillStyle = "pink";
ctx.fill();
```

**OUTPUT**

```
const renderEvents = (ctx: CanvasRenderingContext2D, events: TimelineEvent[]) => {
    // ----- 1. create paths and view models -----
    const allEventsPath = new Path2D();
    // [...]

    // ----- 2. shared calculations -----
    const pixelsPerBeat = calculatePixelsPerBeat(zoomLevel);
    // [...]

    // ----- 3. add to paths and view models -----
    for (const event of events) {
        const bounds = createEventBounds(event, pixelsPerBeat);
        allEventsPath.roundRect(bounds.x, bounds.y, bounds.w, bounds.h, kEventRadii_px);
        // [...]
    }

    // ----- 4. rendering -----
    ctx.fillStyle = "pink";
    ctx.fill(allEventsPath);
    // [...]
};
```

```
// ----- 1. create paths and view models -----
const allBoundsPath = new Path2D();
const waveformsPath = new Path2D();
const selectionsPath = new Path2D();
const resizeHandlesPath = new Path2D();
const boundsPaths: Record<TrackID, Path2D> = {};
const sampleTextVMs: SampleTextVM[] = [];
const qwertyLabelVMs: Readonly<QwertyLabelVM>[] = [];
```

```
// ----- 2. shared calculations -----
const pixelsPerBeat = calculatePixelsPerBeat(zoomLevel);

// calculate and cache voice y positions
const voiceYs: Record<TrackID, Record<VoiceIndex, number>> = {};
{ ... }
```

```
// ----- 3. add to paths and view models -----
for (const event of events) {
    const eventY = voiceYs[event.trackId][event.voiceIndex];
    const bounds = createEventRect(event.interval_ppqn, eventY, pixelsPerBeat, scrollX_px);

    // culling
    { ...
    }

    // all events bounds path
    {
        allBoundsPath.roundRect(bounds.x, bounds.y, bounds.w, bounds.h, kEventRadii_px);
    }

    // track specific bounds path
    { ...
    }

    // text vms
    { ...
    }

    // waveforms path
    { ...
    }
}
```

```
// ----- 4. rendering -----
// backgrounds
{ ...
}

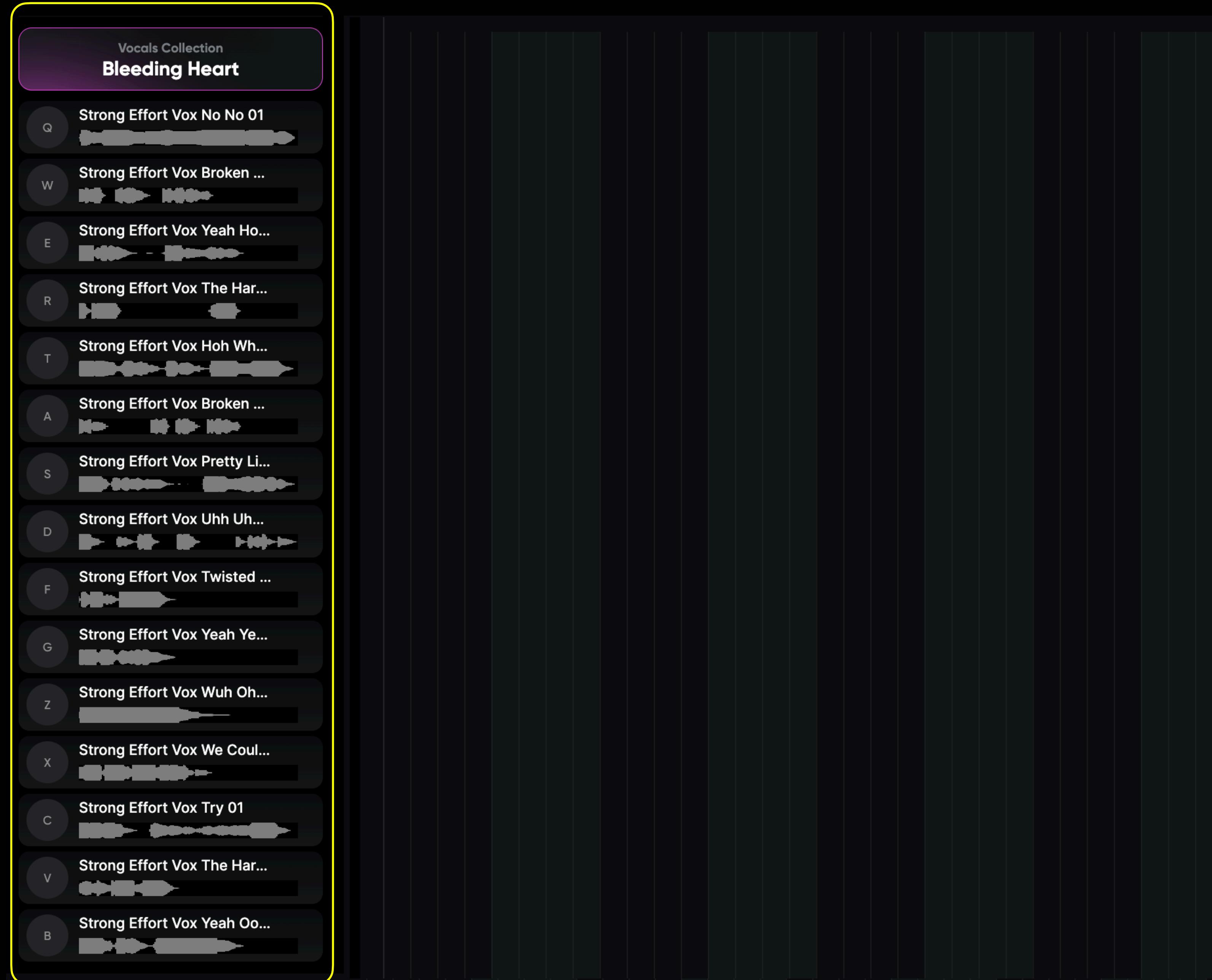
// waveforms
{
    ctx.strokeStyle = waveformColor;
    ctx.lineWidth = kEventWaveformLineWidth_px;
    ctx.stroke(waveformsPath);
}

// selections
{ ...
}

// resize handles
{ ...
```

1. Canvas
2. DOM
3. Layering

**OUTPUT**



**Bleeding Heart**

Vocals Collection

**Bleeding Heart**

- Strong Effort Vox No No 01
- Strong Effort Vox Broken ...
- Strong Effort Vox Yeah Ho...
- Strong Effort Vox The Har...
- Strong Effort Vox Hoh Wh...
- Strong Effort Vox Broken ...
- Strong Effort Vox Pretty Li...
- Strong Effort Vox Uhh Uh...
- Strong Effort Vox Twisted ...
- Strong Effort Vox Yeah Ye...
- Strong Effort Vox Wuh Oh...
- Strong Effort Vox We Coul...
- Strong Effort Vox Try 01
- Strong Effort Vox The Har...
- Strong Effort Vox Yeah Oo...

**Keep It Casual**

Guitar Chords Pumping It Kick Simple Snappy Kick Pattern 01

Guitar Chords Pumping It Hats Simple Straight Moving Hi Hat Pattern 01

Drag or play another sound

Drag or play another sound

The image shows a digital audio workstation (DAW) interface with a dark theme. On the left, there is a vertical list of guitar chord patterns, each with a small preview icon and a letter keybinding (Q through B). The first item is highlighted with a yellow border. The main workspace contains two tracks: a vocal track labeled "Bleeding Heart" and a guitar chords track labeled "Keep It Casual". The vocal track has two clips: "Vocals Strong Effort Vox No No 01" (labeled Q) and "Vocals Strong Effort Vox Hoh Whoa 01" (labeled T). The guitar chords track has two clips: "Guitar Chords Pumping It Kick Simple Snappy Kick Pattern 01" (labeled Q) and "Guitar Chords Pumping It Hats Simple Straight Moving Hi Hat Pattern 01" (labeled T). Both tracks have volume faders, solo buttons, and pan controls. A "Drag or play another sound" placeholder is visible at the bottom of the guitar chords track.

Guitar Chords Collection

**Keep It Casual**

Pumping It Kick Simple Snappy Kick Pattern 01 (Q)

Pumping It Clap Verby Snappy Clap Pattern 01 (W)

Pumping It Snaps Varied Finger Snaps Pattern 01 (E)

Pumping It Clap Simple Trigger Clap Pattern 01 (R)

Pumping It Hats Simple Staccato Hat Pattern 01 (T)

Pumping It Bass Low Deep Bass Pattern 01 (A)

Pumping It Keys Reversed Key Click Pattern 01 (S)

Pumping It Synth Ambient Synth Pad Pattern 01 (D)

Pumping It Guitar Sexy Cool Electric Guitars Pattern 01 (F)

Pumping It Guitar Sexy Warm Acoustic Guitars Pattern 01 (G)

Pumping It Guitar Ambien Ambient Sounds Pattern 01 (Z)

Pumping It Guitar Staccat Staccato Guitars Pattern 01 (X)

Pumping It Guitar Sexy Low Bass Guitars Pattern 01 (C)

Pumping It Guitar Distorte Distorted Guitars Pattern 01 (V)

Pumping It Synth Ambient Synth Pad Pattern 01 (B)

Bleeding Heart

Vocals Strong Effort Vox No No 01 (Q)

Vocals Strong Effort Vox Hoh Whoa 01 (T)

Drag or play another sound

Keep It Casual Fx SOLO 100 L 0 R

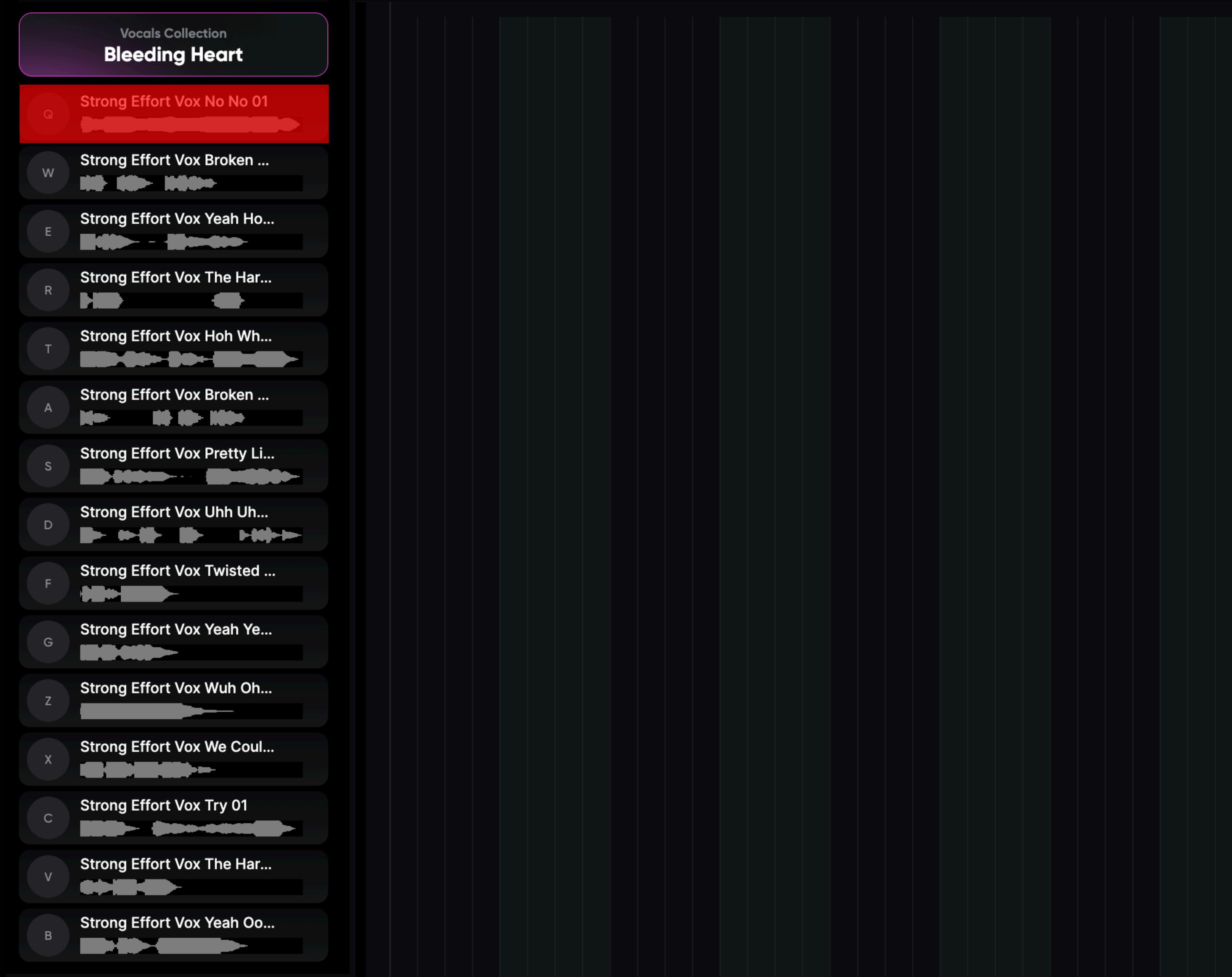
Guitar Chords Pumping It Kick Simple Snappy Kick Pattern 01 (Q)

Guitar Chords Pumping It Hats Simple Straight Moving Hi Hat Pattern 01 (T)

Drag or play another sound

Vocals Collection  
**Bleeding Heart**

- Q Strong Effort Vox No No 01
- W Strong Effort Vox Broken ...
- E Strong Effort Vox Yeah Ho...
- R Strong Effort Vox The Har...
- T Strong Effort Vox Hoh Wh...
- A Strong Effort Vox Broken ...
- S Strong Effort Vox Pretty Li...
- D Strong Effort Vox Uhh Uh...
- F Strong Effort Vox Twisted ...
- G Strong Effort Vox Yeah Ye...
- Z Strong Effort Vox Wuh Oh...
- X Strong Effort Vox We Coul...
- C Strong Effort Vox Try 01
- V Strong Effort Vox The Har...
- B Strong Effort Vox Yeah Oo...



**Vocals Collection**  
**Bleeding Heart**

- Strong Effort Vox No No 01
- Strong Effort Vox Broken ...
- Strong Effort Vox Yeah Ho...
- Strong Effort Vox The Har...
- Strong Effort Vox Hoh Wh...
- Strong Effort Vox Broken ...
- Strong Effort Vox Pretty Li...
- Strong Effort Vox Uhh Uh...
- Strong Effort Vox Twisted ...
- Strong Effort Vox Yeah Ye...
- Strong Effort Vox Wuh Oh...
- Strong Effort Vox We Coul...
- Strong Effort Vox Try 01
- Strong Effort Vox The Har...
- Strong Effort Vox Yeah Oo...

A yellow arrow points upwards from the bottom of the list towards the header.

```
// create parent
const listEl = document.createElement("ol");

// create children, add to parent
for (const sample of kit.samples) {
    const sampleEl = createSidebarSampleEl(sample);
    listEl.appendChild(sampleEl);
}

// add all to document
document.body.appendChild(listEl);
```

```
requestAnimationFrame(() => {  
    // create parent  
    const listEl = document.createElement("ol");  
  
    // create children, add to parent  
    for (const sample of kit.samples) {  
        const sampleEl = createSidebarSampleEl(sample);  
        listEl.appendChild(sampleEl);  
    }  
  
    // add all to document  
    document.body.appendChild(listEl);  
});
```

 [paulirish / what-forces-layout.md](#)

Last active 2 days ago • Report abuse

<> [Code](#)    ⚙️ [Revisions](#) (27)    ☆ Stars 5,000+    ⚡ Forks 833    [Embed](#) ▾    <script src="https://>

What forces layout/reflow. The comprehensive list.

[what-forces-layout.md](#)    [Raw](#)

# What forces layout / reflow

All of the below properties or methods, when requested/called in JavaScript, will trigger the browser to synchronously calculate the style and layout\*. This is also called [reflow](#) or [layout thrashing](#), and is common performance bottleneck.

Generally, all APIs that synchronously provide layout metrics will trigger forced reflow / layout. Read on for additional cases and details.

## Element APIs

### Getting box metrics

- `elem.offsetLeft`, `elem.offsetTop`, `elem.offsetWidth`, `elem.offsetHeight`, `elem.offsetParent`
- `elem.clientLeft`, `elem.clientTop`, `elem.clientWidth`, `elem.clientHeight`
- `elem.getClientRects()`, `elem.getBoundingClientRect()`

<https://gist.github.com/paulirish/5d52fb081b3570c81e3a>

1. Canvas
2. DOM
3. Layering

**OUTPUT**

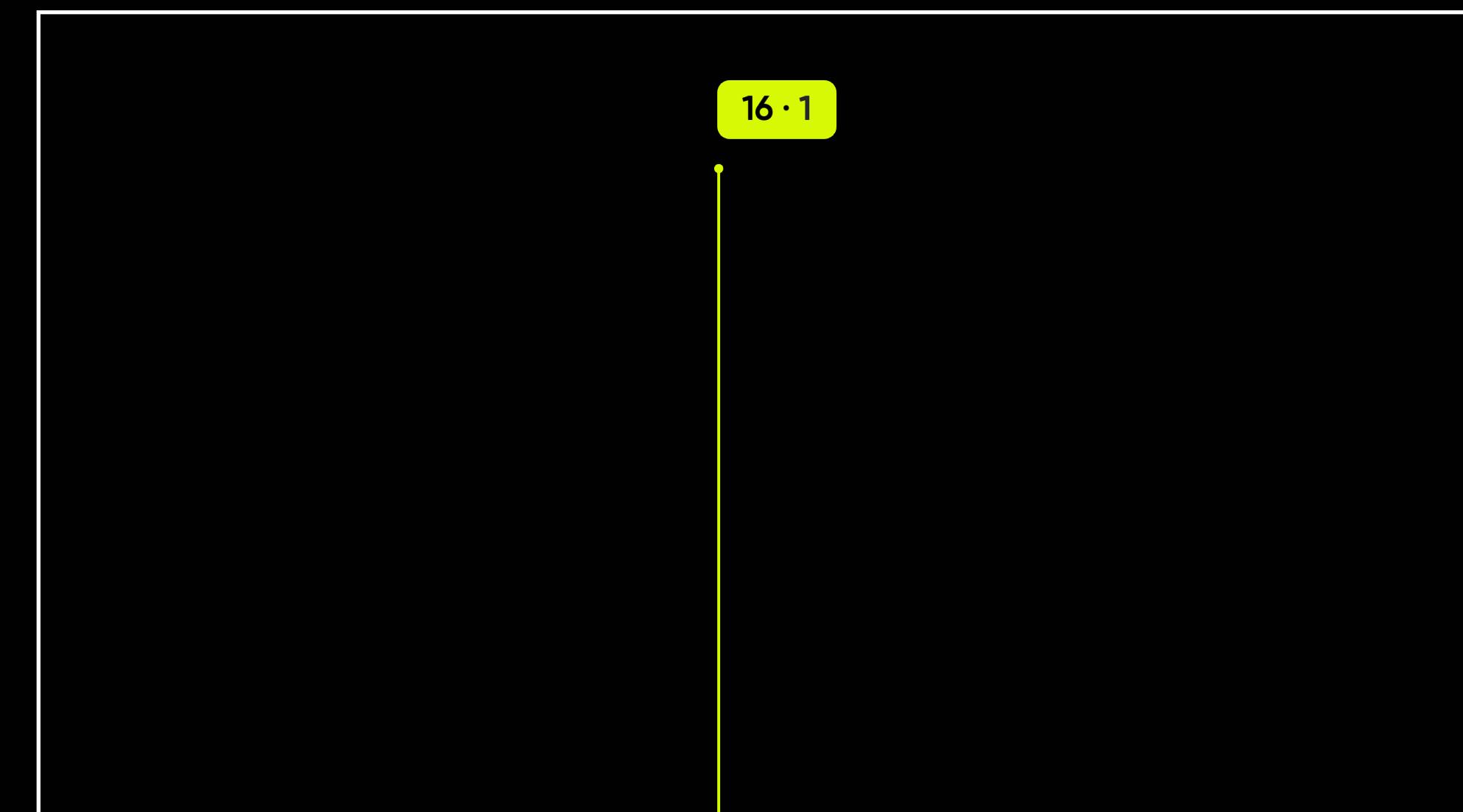
11 12 13 14 15 16 · 1 17 18 19 20 21 22



# Canvas 1



# Canvas 2



```
<canvas style="z-index: 1;"></canvas> <!-- Events -->  
<canvas style="z-index: 2;"></canvas> <!-- Playhead -->
```





**OUTPUT**



# In conclusion...

# Batch

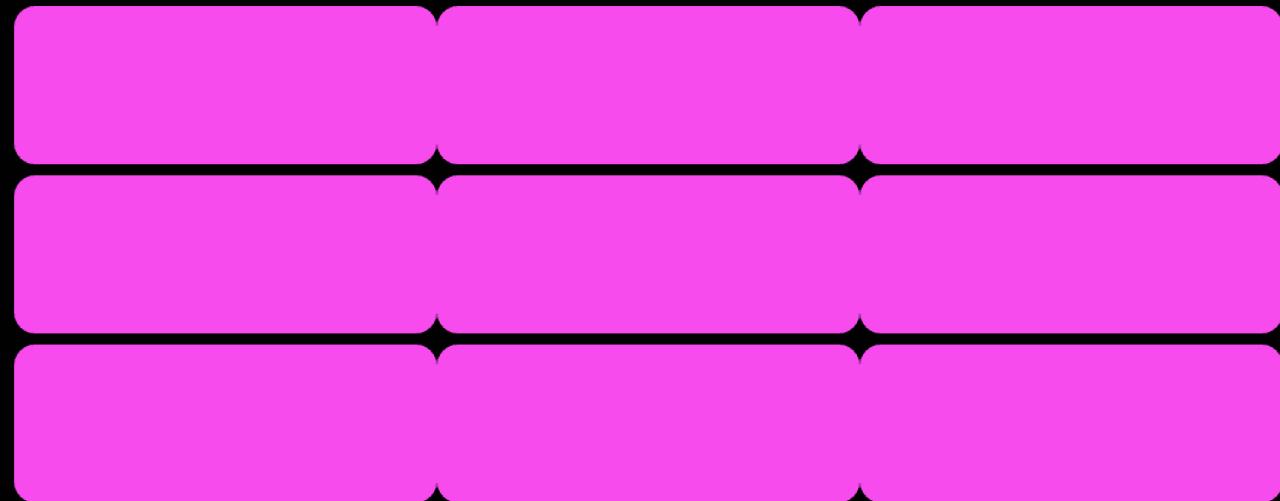
It is very unlikely in any given situation  
that one thing exists in a vacuum.

**Mike Acton**

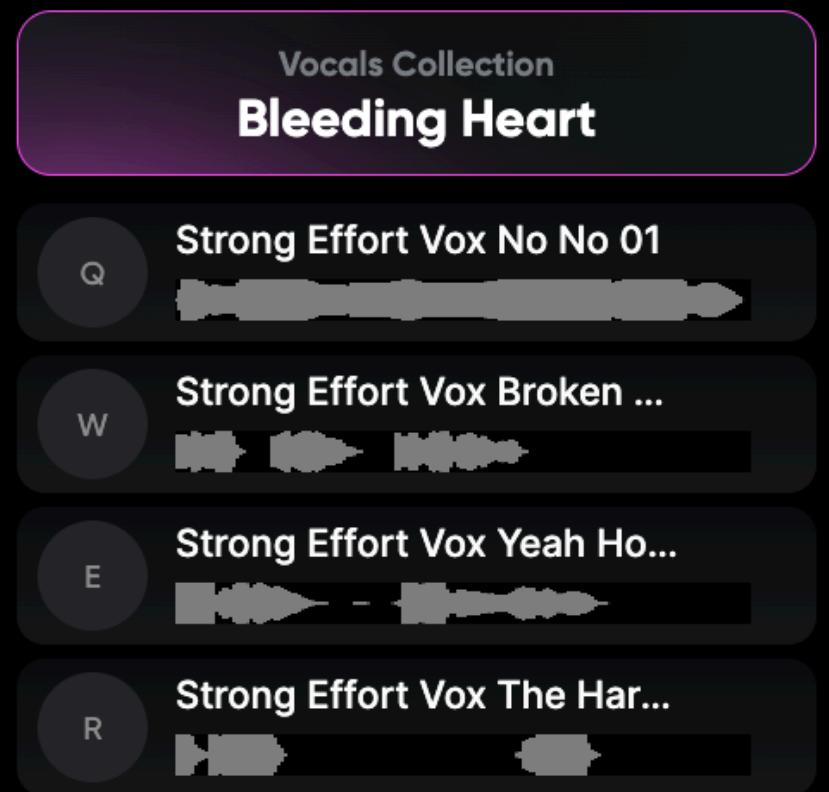
CppCon 2014: "Data-Oriented Design and C++" 37m 30s  
<https://www.youtube.com/watch?v=rXOltVEVjHc>

# Consider the context

Batch when visual properties are the same...



Batch when changes happen at the same time



Separate entirely where appropriate





Slides and book mailing list

[arthurcarabott.com/adc-2024](http://arthurcarabott.com/adc-2024)

Contact

[arthur@arthurcarabott.com](mailto:arthur@arthurcarabott.com)

