SeqEx

MIDI Sequencer in Elixir

MIDI

Musical Instrument Digital Interface

MIDI Messages
Important Concepts

Notes

Velocity

Channels

MIDI Messages

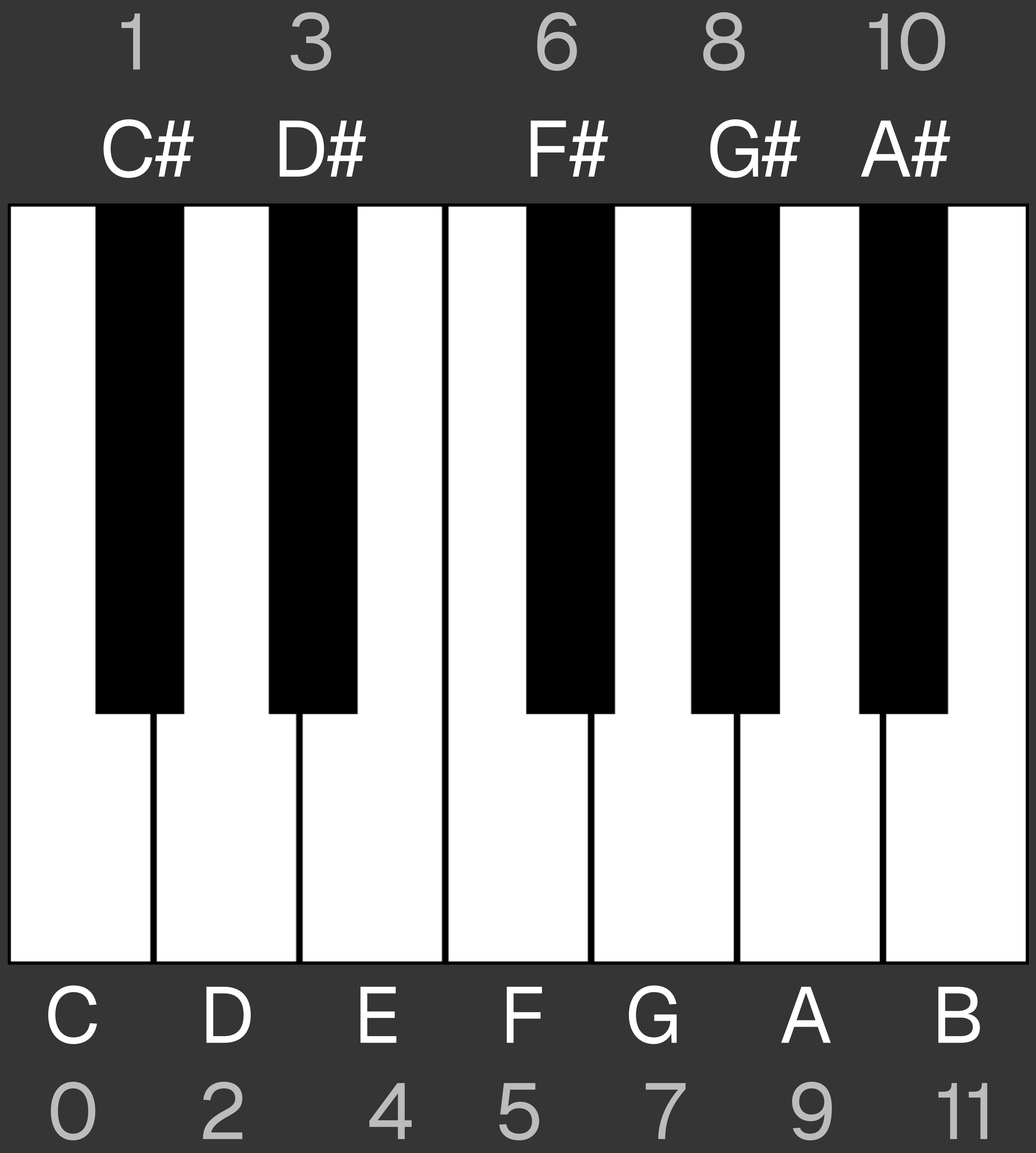
Important Concepts – Notes

12 Notes

Integers from 0 to 11

For each octave +12

C - 0, 12, 24, 36, ...



MIDI Messages

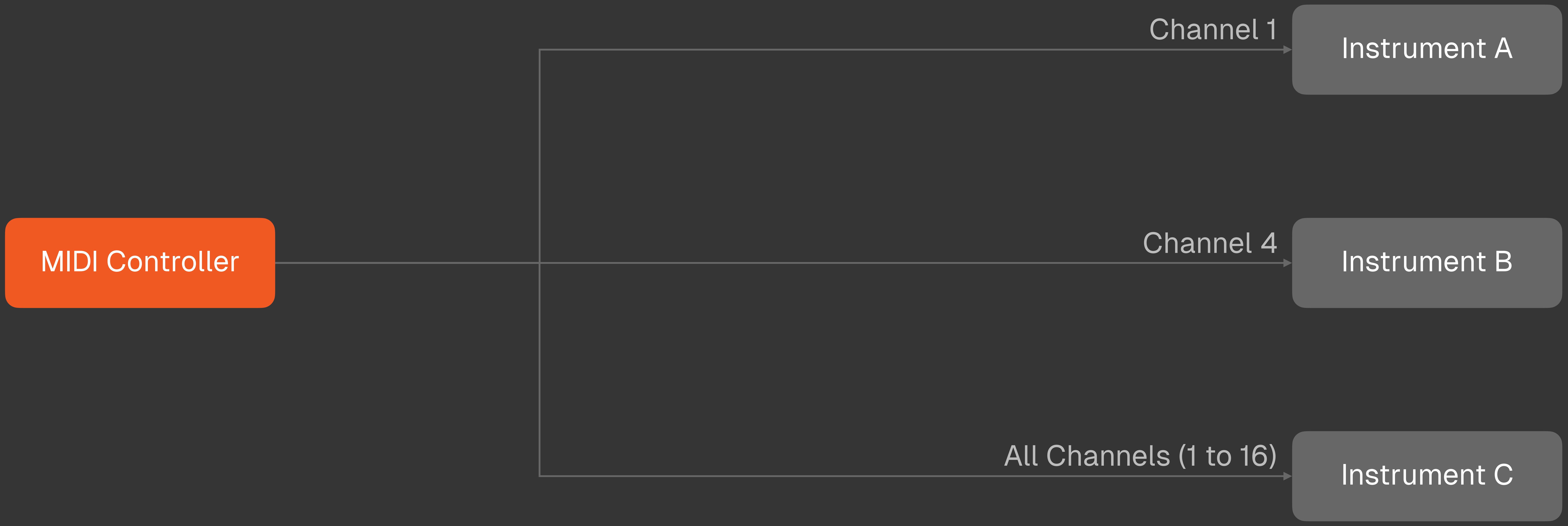
Important Concepts – Velocity

How fast/hard a key is pressed

Any value between 0 and 127

MIDI Messages

Important Concepts – Channels



MIDI Messages

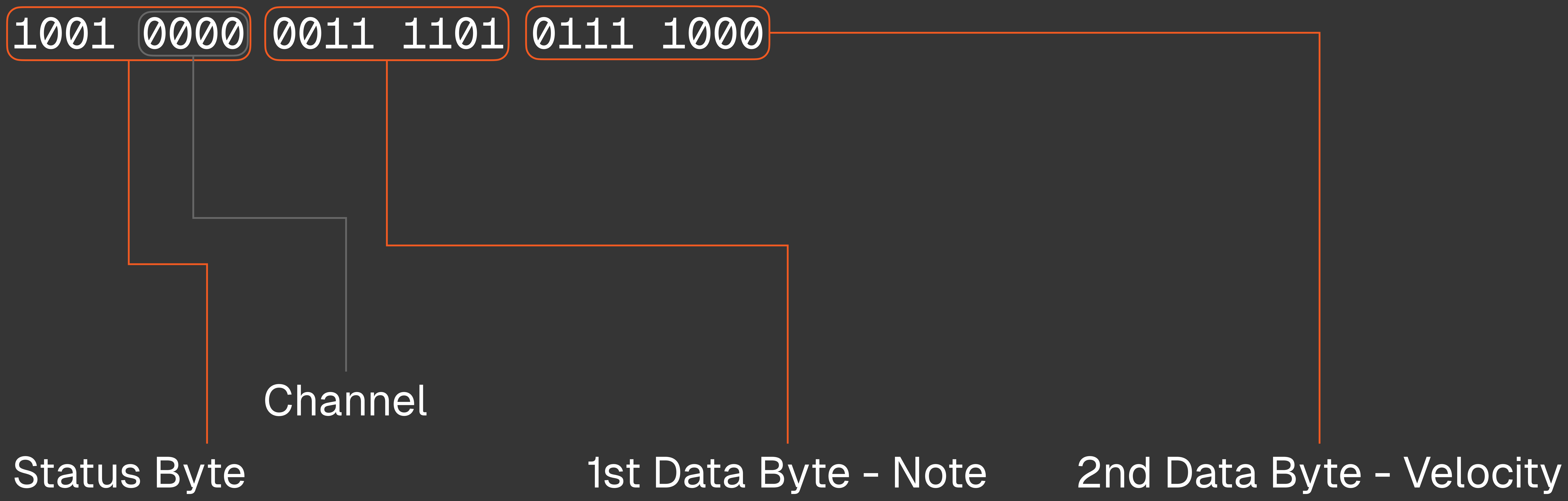
Format

Status Byte – `1XXX XXXX`

Data Bytes (Optional) – `0XXX XXXX`

MIDI Messages

Note On



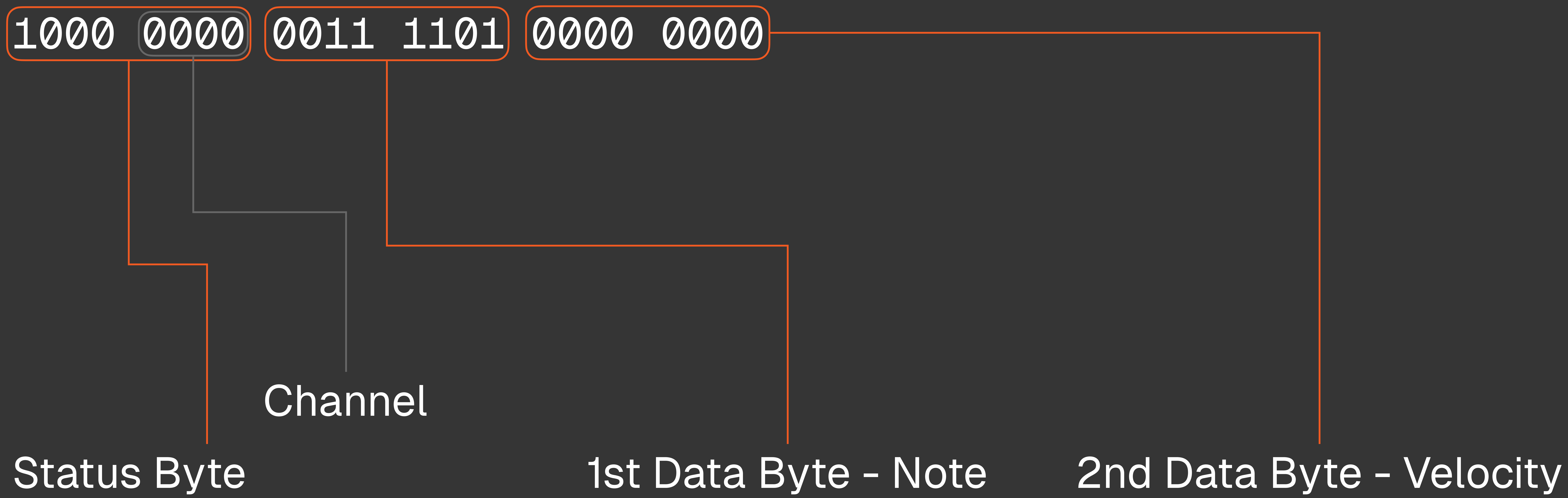
1001 0000 (144)

0011 1101 (60)

0111 1000 (120)

MIDI Messages

Note Off



```
1000 0000 (128)          0011 1101 (60)          0000 0000 (0)
```

Midiex

A cross-platform, realtime MIDI processing library in Elixir which wraps the midir Rust library

hex.pm/packages/midiex

Midiex

Connections

```
[output_port | _] = Midiex.ports(:output)

connection = Midiex.open(output_port)
```

Midiex

Note On

```elixir
message = <<144, 60, 120>>

Midiex.send_msg(connection, message)
```

Midiex

Note Off

```
message = <<128, 60, 0>>

Midiex.send_msg(connection, message)
```

Midiex

Listener

```elixir
port = Midiex.ports(:input) |> List.first()

Midiex.Listener.start_link(port: port)

Midiex.Listener.add_handler(

  listener,

  fn message -> IO.inspect(message)

end)
```

MIDI Visualisation

Listening to MIDI Messages in a LiveView

Sequencer

GenServer as a MIDI Sequencer

Sequencer

Note Duration

BPM – Beats Per Minute

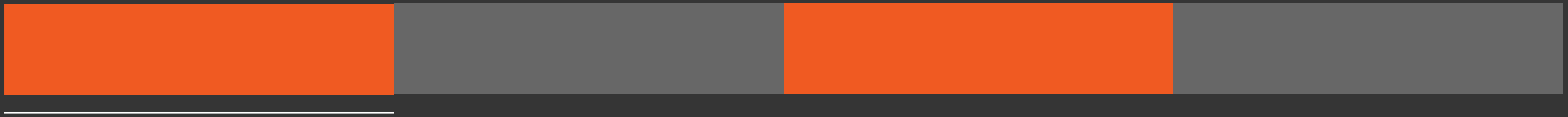
120 BPM – 120 Quarter Notes Per Minute

1 Minute – 60 000ms

Sequencer

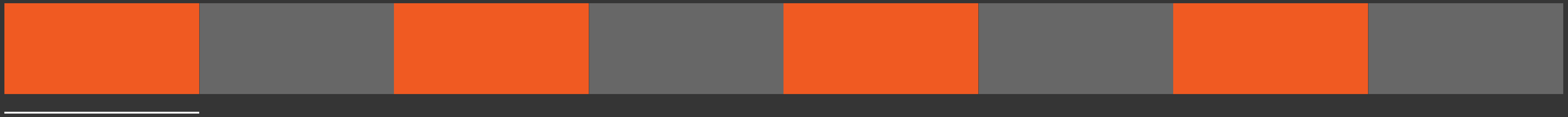
Note Duration

60000ms / 120 = 500ms per 1/4 note



500ms

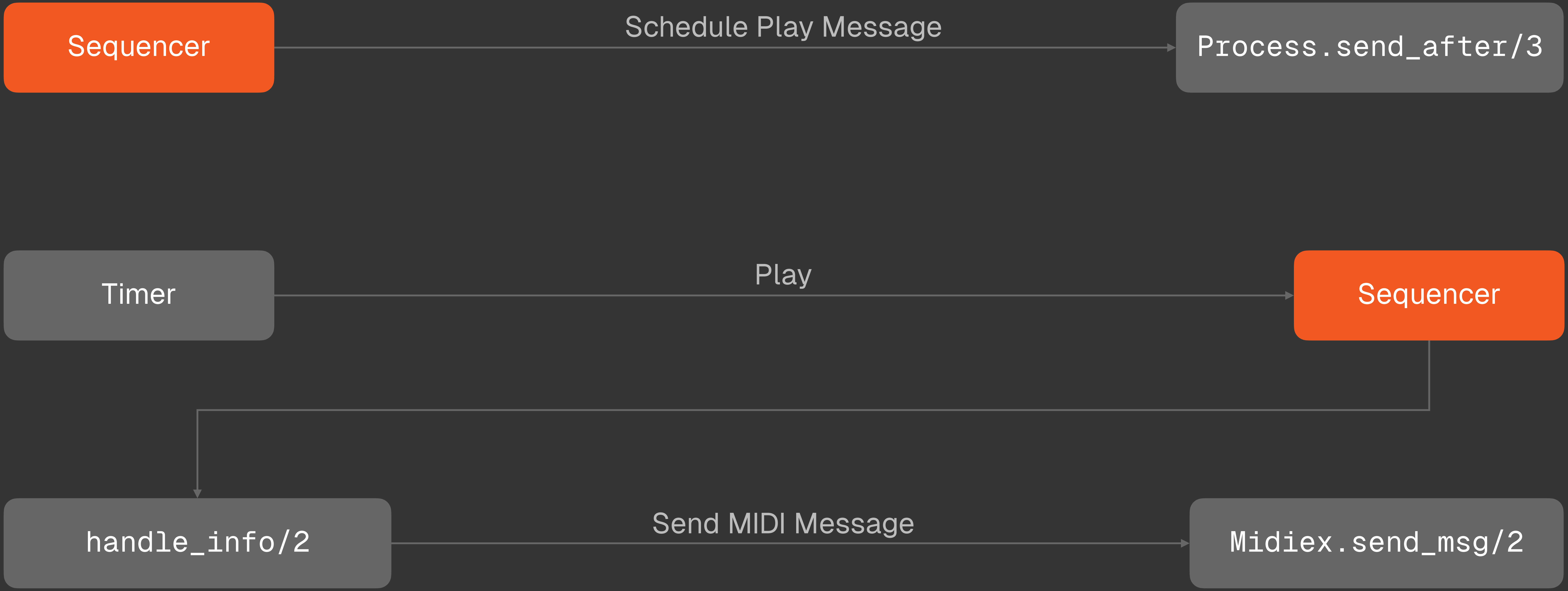
60000ms / 120 / 2 = 250ms per 1/8 note



250ms

Sequencer

Scheduling Messages



Sequencer

Code

```
# 1. Stop previous notes and play current ones.
def play(state) do
    previous_note = note(state.sequence, state.step)
    new_note = notes(state.sequence, next_step(state))
    Midiex.send_msg(state.connection, <<144, current_note, _>>)
    Midiex.send_msg(state.connection, <<128, previous_note, _>>)
    Process.send_after(self(), :play, interval)
    {:noreply, Map.put(state, :step, next_step(state)}
end
```

Sequencer

Code

```
# 2. Handle message with handle_info/2.

def handle_info(:play, state) do

  . . .

  play(state)

end
```

Sequencer

Play

`Sequence: [:C4, :E4, :F4, :B4, :C5, :B4, :F4, :E4]`

`Step: 1`

1. Send Note On MIDI message(s)
2. Schedule next step (`Process.send_after/3`)

Sequencer

Play

```
Sequence: [:C4, :E4, :F4, :B4, :C5, :B4, :F4, :E4]

Step: 2
```

1. Send Note Off MIDI message(s)
2. Send Note On MIDI message(s)
3. Schedule next step (`Process.send_after/3`)

Sequencer

Play

```
Sequence: [:C4, :E4, :F4, :B4, :C5, :B4, :F4, :E4]

Step: 3
```

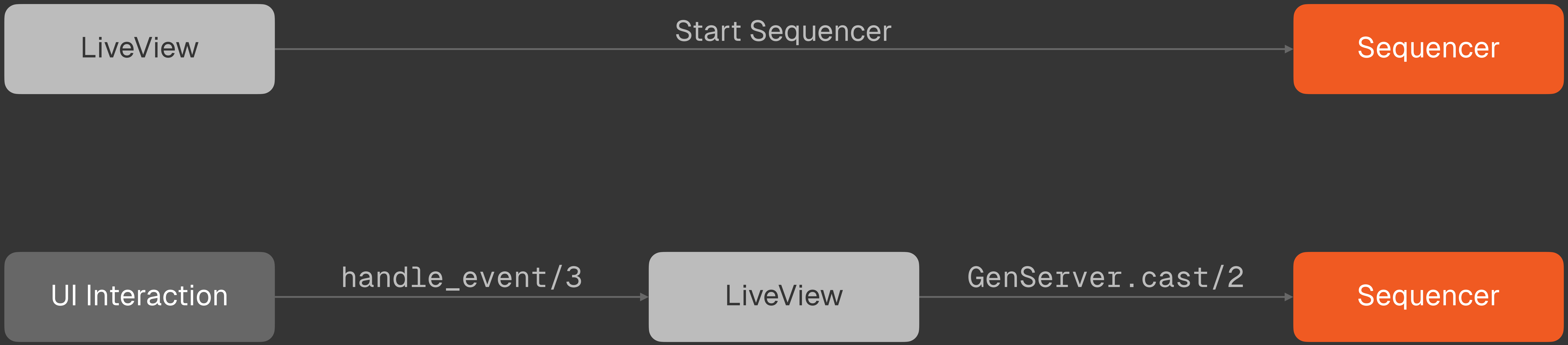
1. Send Note Off MIDI message(s)
2. Send Note On MIDI message(s)
3. Schedule next step (`Process.send_after/3`)

Live Sequencer

Controlling MIDI sequencer with LiveView

Live Sequencer

Mount



Live Sequencer

Code

```
# 1. Start sequencer on mount/3.
def mount(_params, _session, socket) do
  Midiex.ports(:output)
  ▷ List.first()
  ▷ Sequencer.start_link(name: Seqex.Sequencer)
  …
end
```

Live Sequencer

Code

```
# 2. Leverage phx-click and phx-value-x to update state.

<button

  phx-click=“update-note”

  phx-value-index={step}

  phx-value-note={note}

/>
```

Live Sequencer

Code

```
# 3. Handle UI event with handle_event/3.

def handle_event(“update-note”, params, %{assigns: assigns} = socket) do

   assigns.sequence

   ▷ update_sequence(params[“index”], params[“note”])

   ▷ then(fn sequence →

      Sequencer.update_sequence(assigns.sequencer, sequence)

   end)

end
```

Collaborative Live Sequencer

Your turn to control the sequencer!

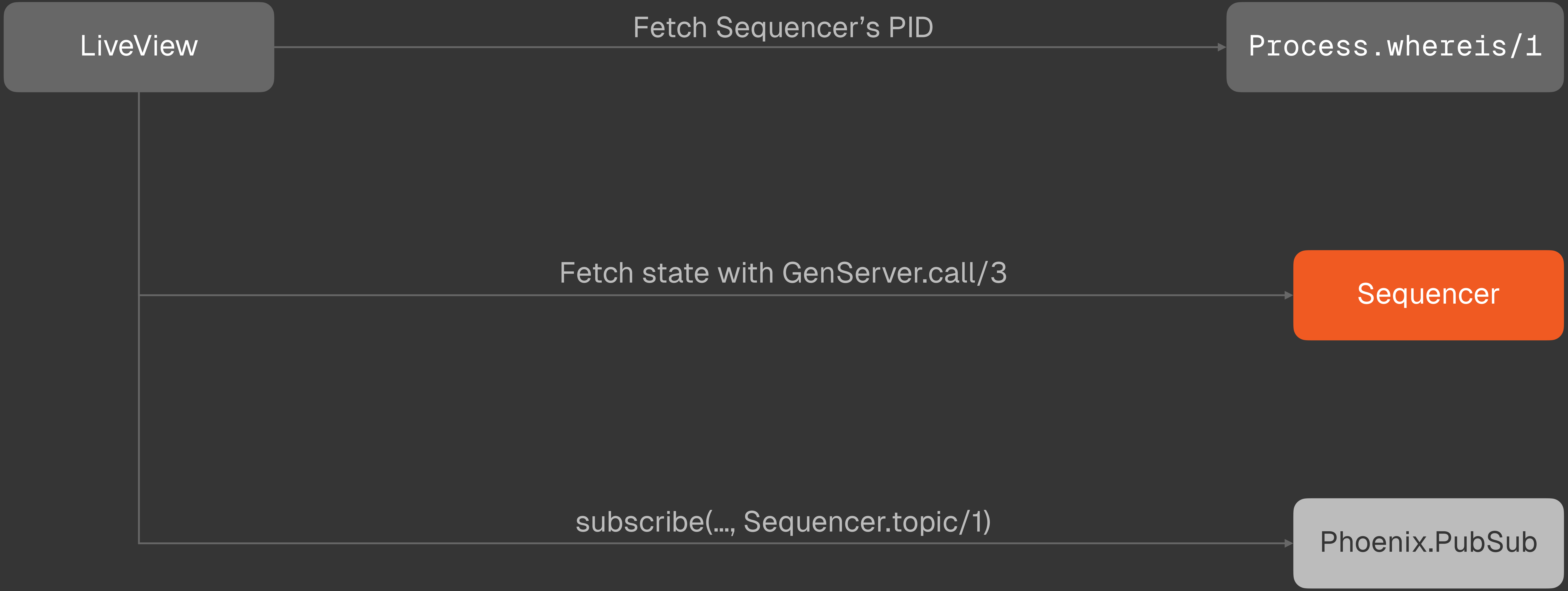
Collaborative Live Sequencer

Demo

`seqex.ngrok.app/sequencer`

Collaborative Live Sequencer

Mount – Sequencer Running



Collaborative Live Sequencer

Mount – Sequencer Running

```
# 1. Find sequencer’s PID.

pid = Process.whereis(Seqex.Sequencer)


# 2. Get sequencer’s current sate.

socket

▷ assign(:sequence, Sequencer.sequence(pid))

▷ assign(:bpm, Sequencer.bpm(pid))
```

Collaborative Live Sequencer

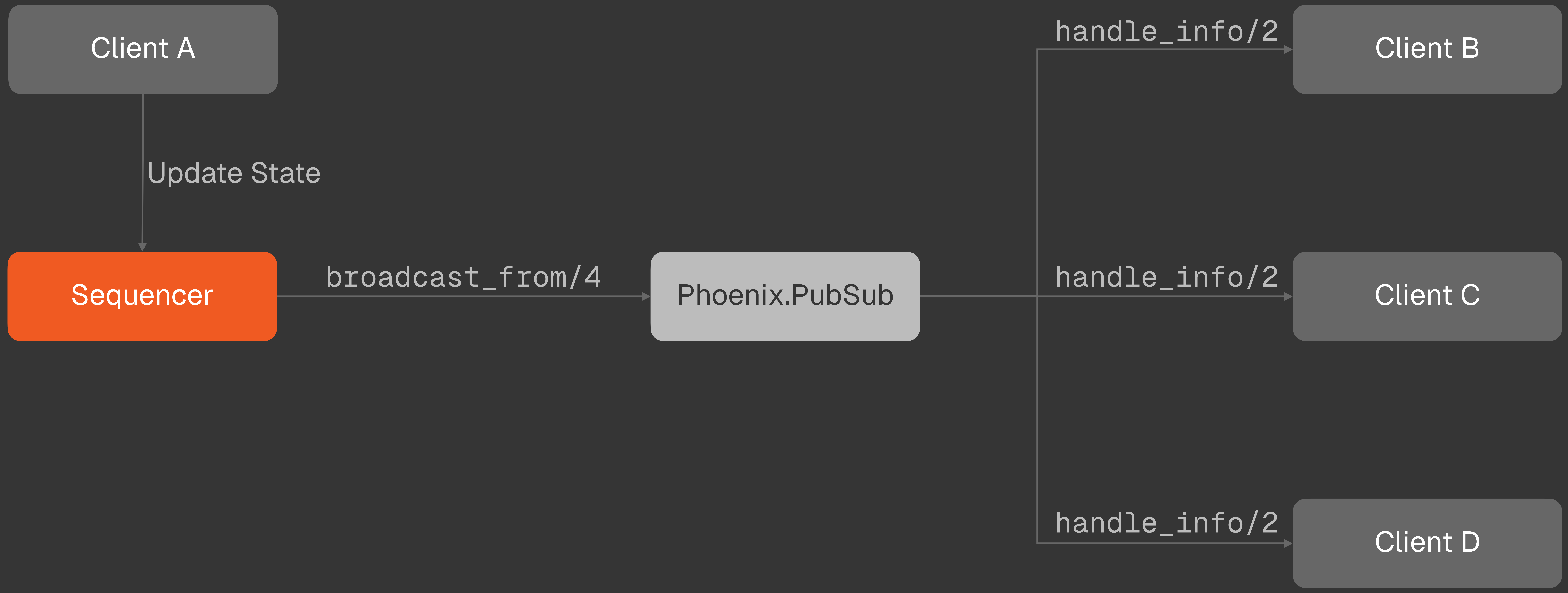
Mount – Sequencer Running

```
# 3. Subscribe to sequencer’s messages.

PubSub.subscribe(Seqex.PubSub, Sequencer.topic(pid))
```

Collaborative Live Sequencer

PubSub



Collaborative Live Sequencer

PubSub

```
# 1. Sequencer broadcasts message to clients.
PubSub.broadcast(
  Seqex.PubSub,
  Sequencer.topic(self()),
  {:step, step}
)
```

Collaborative Live Sequencer

PubSub

```
# 2. Client updates state based on message’s
information.
def handle_info({:step, step}, socket) do
  {:noreply, assign(socket, :step, step + 1)}
end
```

Thank you

Go make some noise

github.com/dinocosta/seqex
github.com/haubie/midiex

x.com/dinocosta_
dino.codes