

SeqEx

MIDI Sequencer in Elixir

MIDI

Musical Instrument Digital Interface

MIDI Messages

Important Concepts

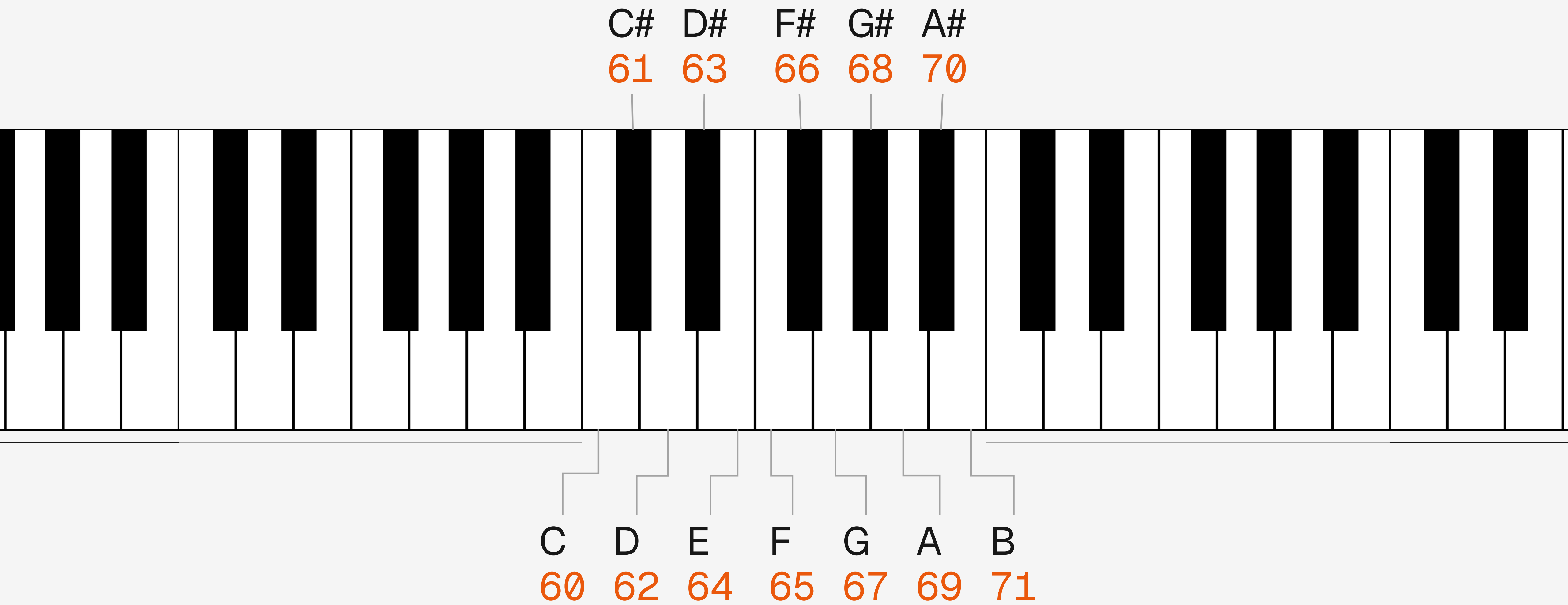
Notes

Velocity

Channels

MIDI Messages

Important Concepts – Notes



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

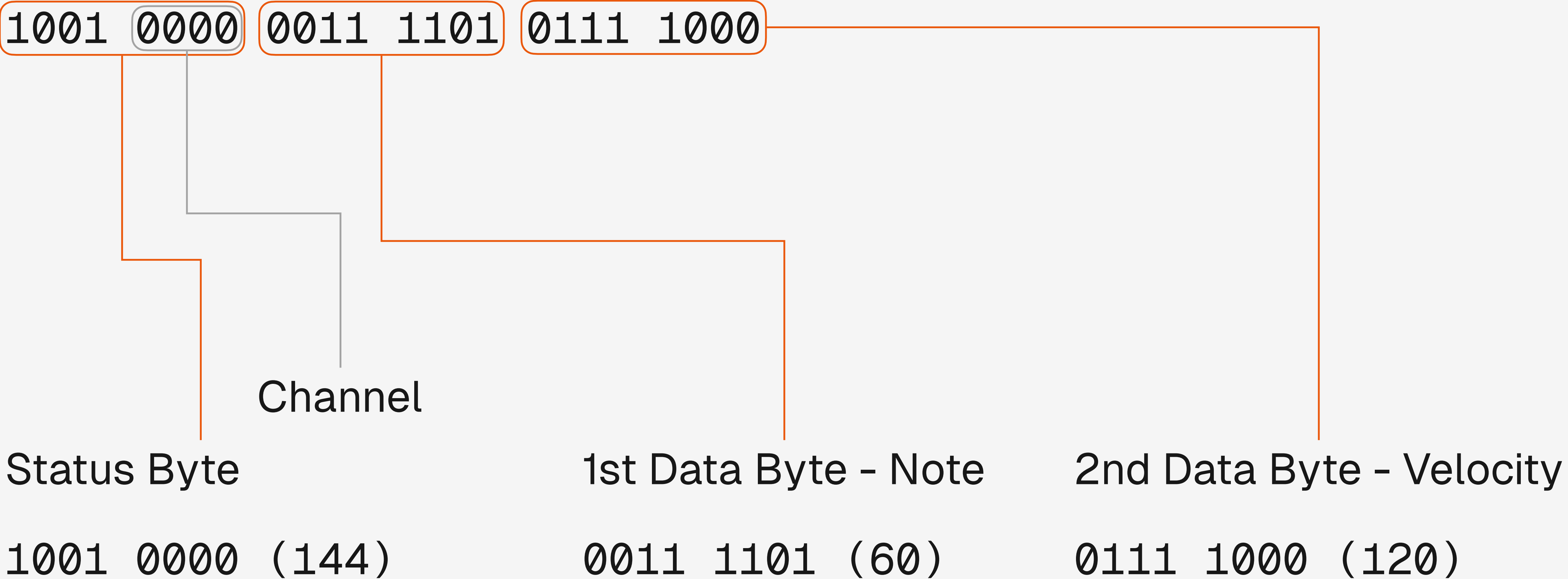
Format

Status Byte – 1XXX XXXX

Data Bytes (Optional) – 0XXX XXXX

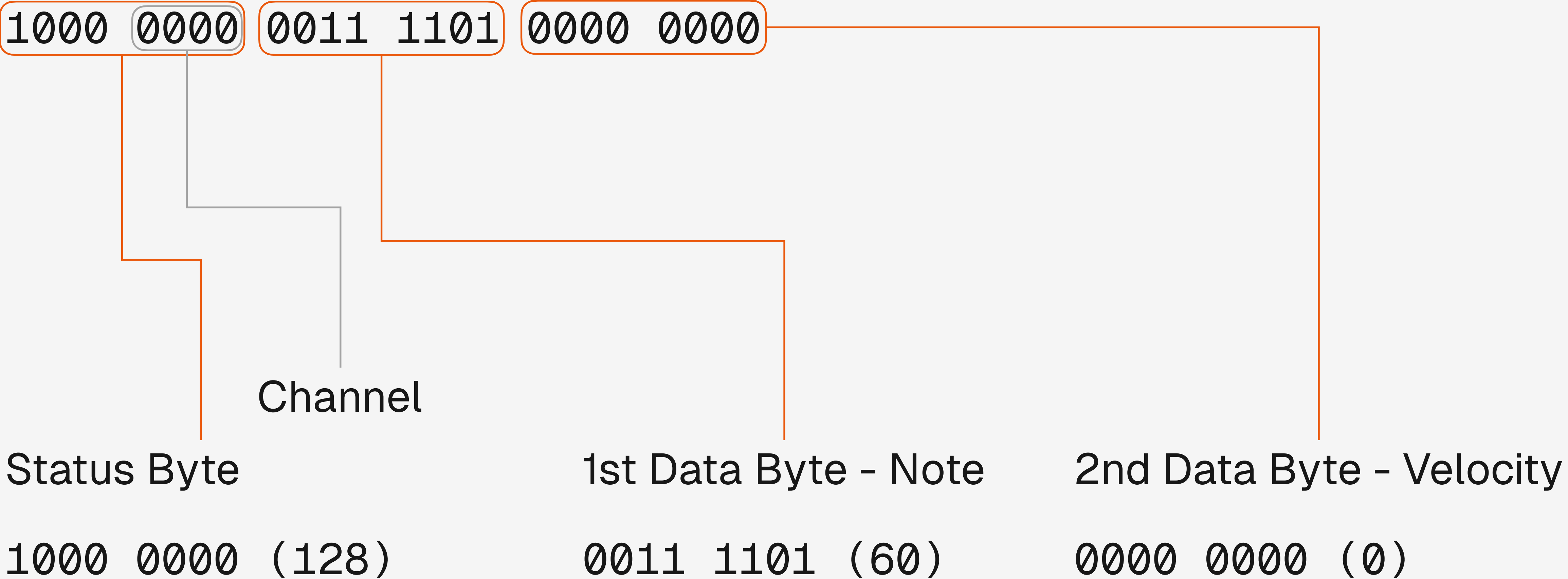
MIDI Messages

Note On



MIDI Messages

Note Off



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

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

```
Midiex.send_msg(connection, message)
```

Midiex

Note Off

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

```
Midiex.send_msg(connection, message)
```

Sequencer

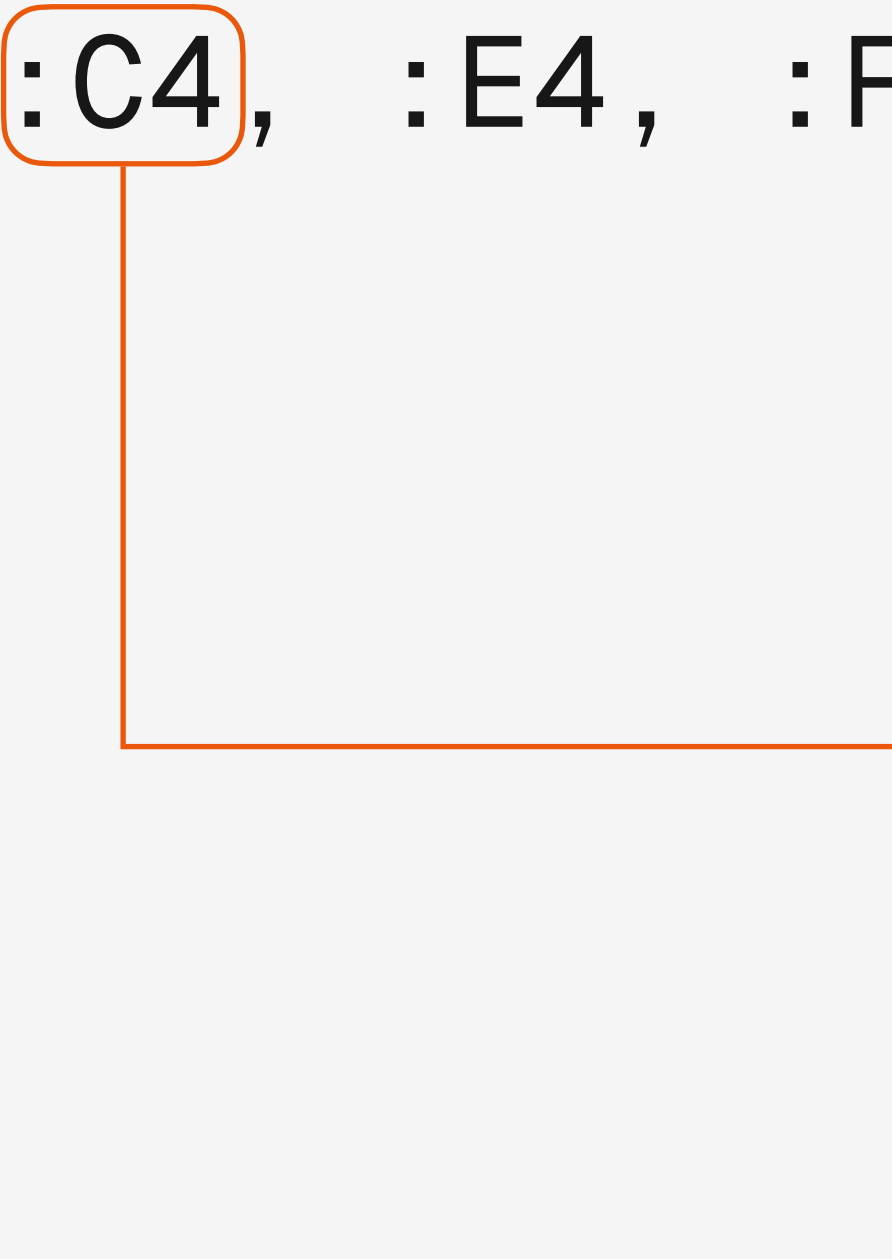
GenServer as a MIDI Sequencer

Sequencer

Play

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

Step: 1

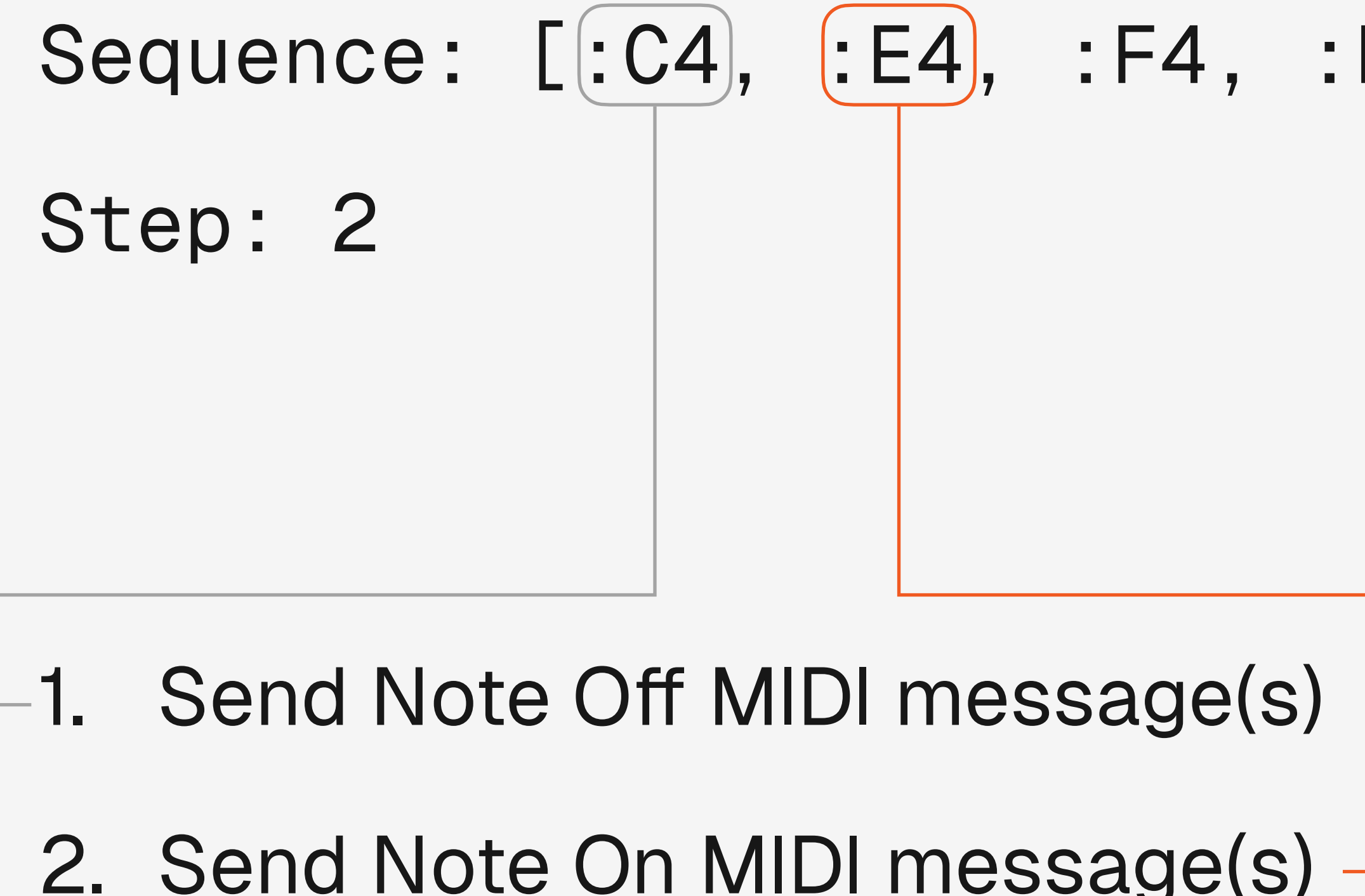
- 
1. Send Note On MIDI message(s)
 2. Wait for specific interval, continue to next step

Sequencer

Play

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

Step: 2


- 
- The diagram illustrates the MIDI message sequence for two notes. A grey line connects the first note ':C4' in the sequence to step 1 (Send Note Off) and step 2 (Send Note On). An orange line connects the second note ':E4' to step 1 and step 2. Step 3 (Wait for specific interval) is a common step for both notes.
1. Send Note Off MIDI message(s)
 2. Send Note On MIDI message(s)
 3. Wait for specific interval, continue to next step

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. Wait for specific interval, continue to next step

Sequencer

Note Duration

BPM → Beats Per Minute

120 BPM → 120 Quarter Notes Per Minute

1 Minute → 60 000ms

Sequencer

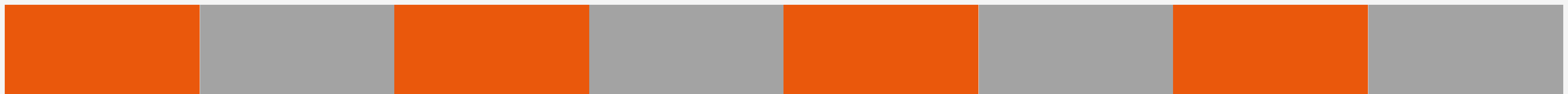
Note Duration

$60000\text{ms} / 120 = 500\text{ms}$ per 1/4 note



500ms

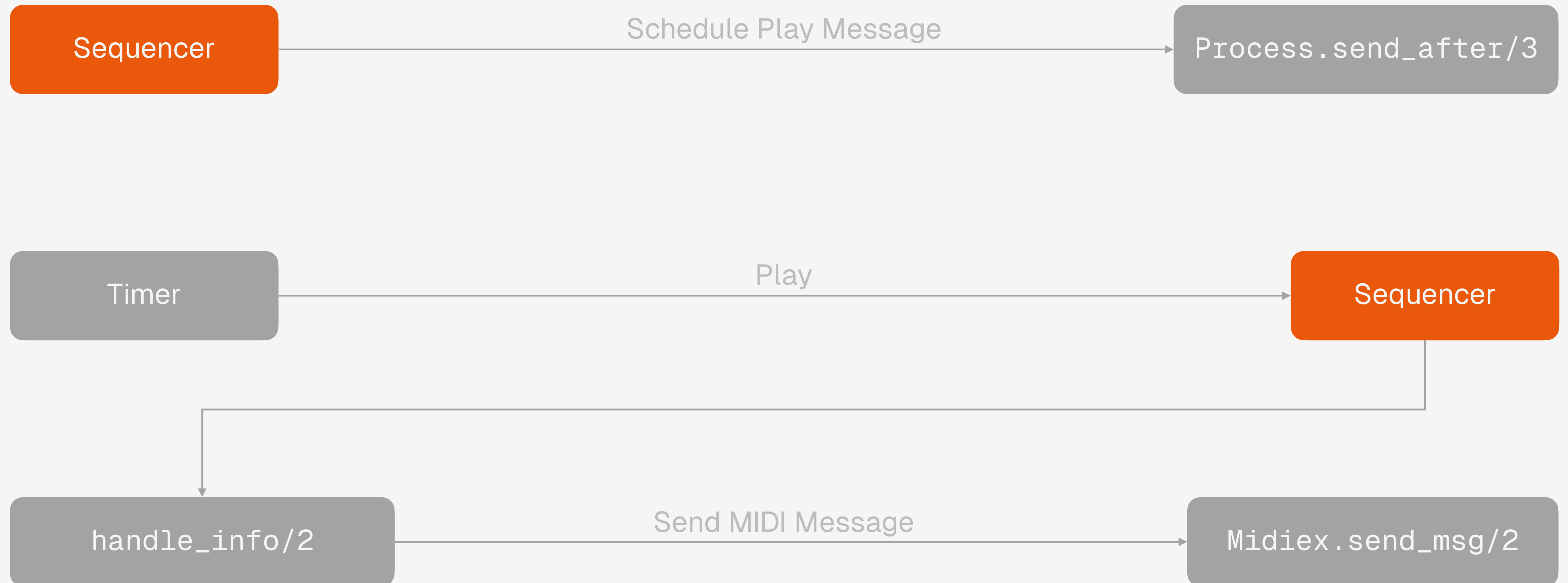
$60000\text{ms} / 120 / 2 = 250\text{ms}$ per 1/8 note



250ms

Sequencer

Scheduling Messages



Syncing Sequencers

MIDI System Real-Time Messages to the rescue

MIDI System Real-Time Messages

MIDI Start

11111010

Tells other MIDI devices to start playing

Resets device's song position

MIDI System Real-Time Messages

MIDI Stop

11111100

Tells other MIDI devices to stop playing

MIDI System Real-Time Messages

MIDI Continue

11111011

Tells MIDI device to continue playing from the location where it stopped

MIDI System Real-Time Messages

MIDI Clock

11111000

Used to keep multiple MIDI devices synchronised

Sent at consistent regular intervals, depending on the Master Tempo

MIDI Clock

Interval

24 Pulses Per Quarter (PPQ)



1/4 Note

1/4 Note

1/4 Note



1/8 Note

1/8 Note

1/8 Note

1/8 Note

1/8 Note

1/8 Note

MIDI Clock

Interval

24 Pulses Per Quarter (PPQ)

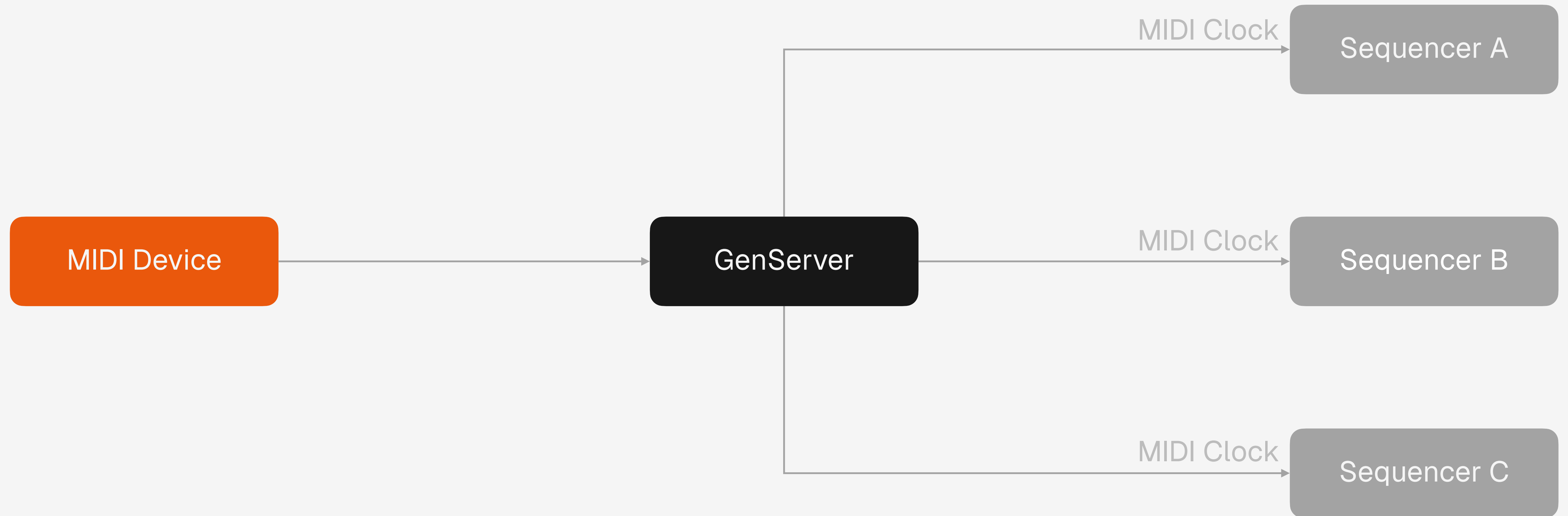


120 BPM → 120 quarter notes in 60 seconds

$60000\text{ms} / (120 * 24 \text{ PPQ}) \rightarrow 20.8333\text{ms}$

Sync

Message Forwarding



Midiex

Subscribe

```
# 1. Subscribe to MIDI messages from an input port.
```

```
input_port = Midiex.ports(:input) ▷ List.first()
```

```
Midiex.subscribe(input_port)
```

```
# 2. Handle the messages.
```

```
def handle_info(%Midiex.Message{}, state), do: . . .
```

Sequencers Galore

This might get noisy!

Midix

Subscribe

```
# 1. Start sequencer on mount/3.
```

```
def mount(_params, _session, _socket) do
```

```
  Midix.ports(:output)
```

```
  ▷ List.first()
```

```
  ▷ Sequencer.start_link()
```

```
  ...
```

```
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 sequencers!

Collaborative Live Sequencer

Mount - Sequencer Running

1. Fetch sequencer's PID.

```
pid = Map.get(params, "pid") ▷ IEx.Helpers.pid()
```

2. Get sequencer's current state.

socket

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

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

Collaborative Live Sequencer

Mount - Sequencer Running

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

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

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)}
end
```

Collaborative Live Sequencer

PubSub



Thank you

Go make some noise

 github.com/dinocosta/seqex

 github.com/haubie/midiex

 x.com/dinocosta_

 dino.codes