chrome | CAMPUS SEOUL

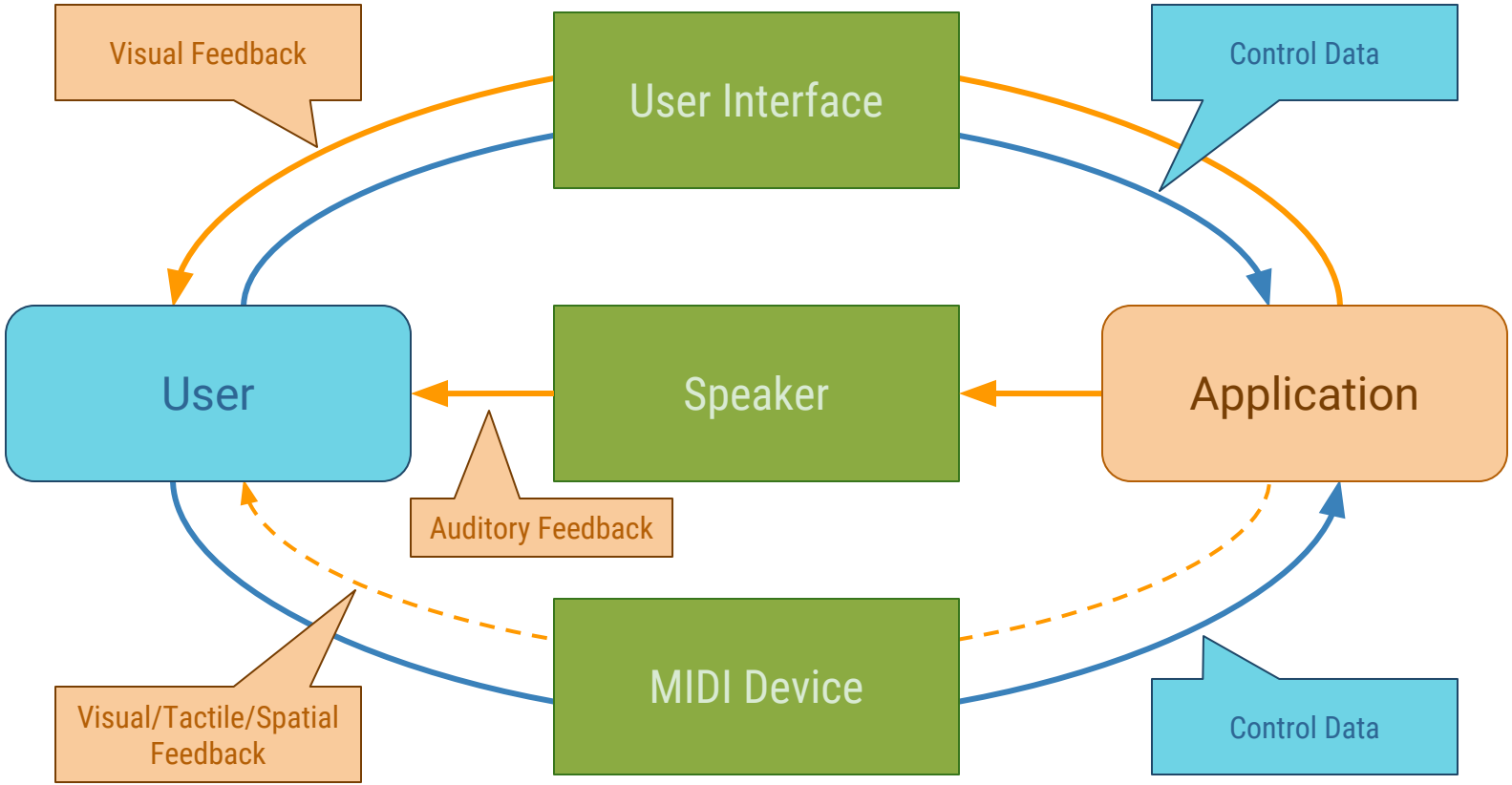
Musicking on the Web

Workshop Day 2 / Lecture 3

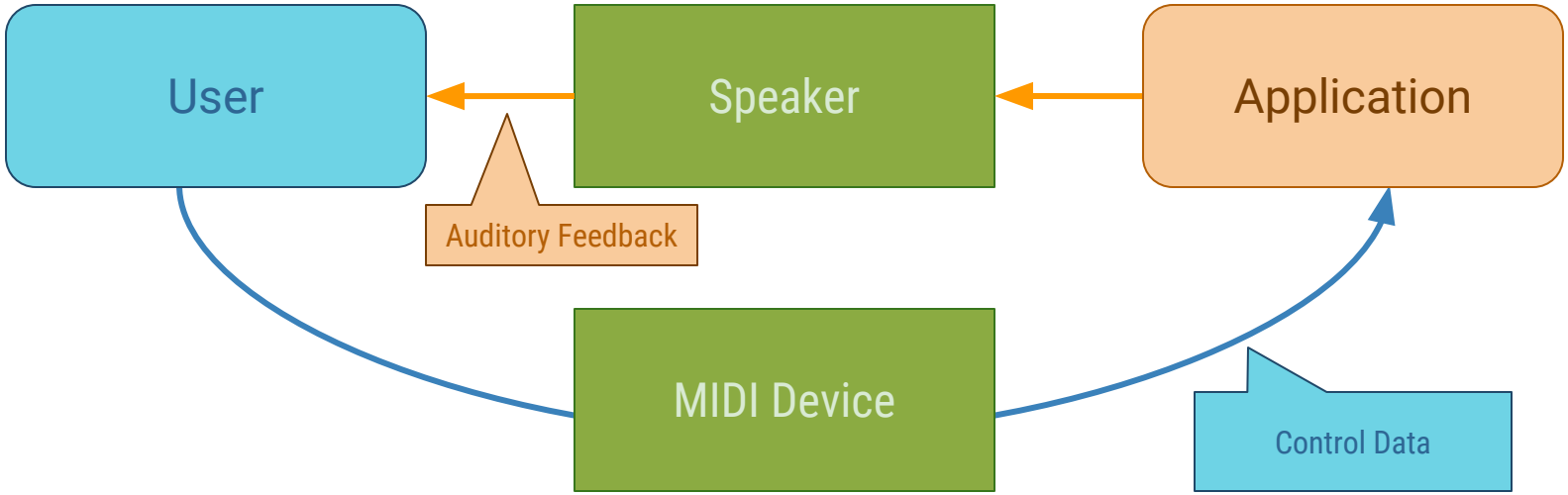
Hongchan Choi

Software Engineer, Google Chrome
Web Audio API + Music Apps

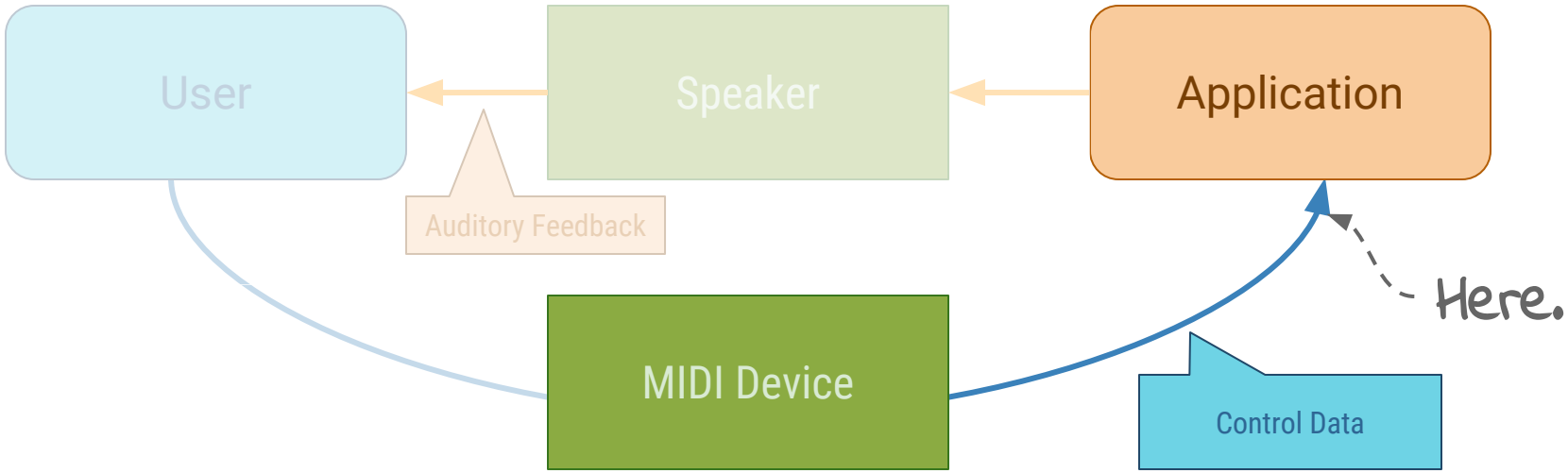
hoch.io


Visual Feedback
User Interface
Control Data
User
Speaker
Application
Auditory Feedback
Visual/Tactile/Spatial Feedback
MIDI Device
Control Data

Day 2:
Lecture 3



Day 2:
Lecture 3



MIDI*

* Musical Instrument Digital Interface

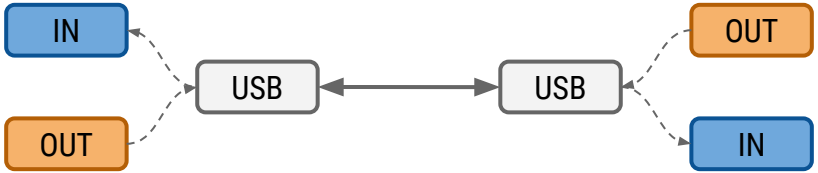
How MIDI works

http://www.midi.org/

“a system that allows
electronic musical instruments and computers
to send instructions to each other.”

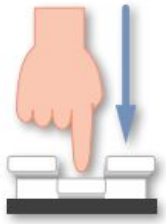
“Introduction to MIDI”, MIDI Manufacturers Association

"Introduction to MIDI", MIDI Manufacturers Association


IN
OUT
USB
USB
OUT
IN


LIVID
BASE

MIDI Messages



NOTE ON: CH #1	PITCH: C5	VELOCITY: 64

NOTE OFF: CH #1	PITCH: C5	VELOCITY: 0

"Introduction to MIDI", MIDI Manufacturers Association

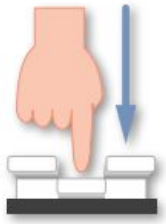
MIDI Messages

	Status	Databyte 1	Databyte 2
Note On	1001NNNN	0KKKKKKK	0VVVVVVV
Note Off	1000NNNN	0KKKKKKK	0VVVVVVV
Control Change	1011NNNN	0CCCCCCC	0VVVVVVV
Pitch Bend	1110NNNN	0LLLLLLL	0MMMMMMM

More: polyphonic key pressure, program change, channel pressure...

"Summary of MIDI Messages", MIDI Manufacturers Association

MIDI Messages



10010001	00111111	01000000
NOTE ON / CH #1	PITCH = 60 (C5)	VELOCITY = 64

10000001	00111111	00000000
NOTE OFF / CH #1	PITCH = 60 (C5)	VELOCITY = 0

"Introduction to MIDI", MIDI Manufacturers Association

Programming Web MIDI API

W3C Editor’s Draft

https://webaudio.github.io/web-midi-api/

Accessing MIDI system

- Gateway to the MIDI subsystem.
- **Promise** pattern: `onsuccess` and `onfailure`

```
var globalMIDIAccess = null;

function onsuccess(midiAccess) { globalMIDIAccess = midiAccess; }
function onfailure(errorMessage) { console.log('error : ' + errorMessage); }

navigator.requestMIDIAccess().then(onsuccess, onfailure);
```

Handling Incoming MIDI Message

- Attach `.onmidimessage` event handler to input.
- `MIDIAccess.inputs()` returns ES6 Map.

```
var firstInput = globalMIDIAccess.inputs().values().next();      // Get 1st MIDI input.

firstInput.onmidimessage = function (event) {
     console.log(event.data[0] + event.data[1] + event.data[2]); // 3-bytes message.
};
```

Parsing MIDI message

- 3-bytes in a row: `data[0]`, `data[1]` and `data[2]`.
- Use bitwise operation to filter data out - or use **Spiral** for parsing.

```
firstInput.onmidimessage = function (event) {
      switch (event.data[0] & 0xf0) {                        // Mask off the channel number.
            case 0x90:                                       // Note on!
                  noteOn(event.data[1], event.data[2]);      // Hit a note with pitch and velocity.
                  break;
            case 0x80:                                       // Note off.
                  noteOff(event.data[1]);                    // Stop a note at pitch.
                  break;
      }
};
```

Walkthrough: Vanilla Web MIDI API

```
<script>
	var synth = {
		noteOn: function (pitch, velocity) {
			/* play sound with pitch and velocity... */
		},
		onmidimessage: function (message) {
			if (message.type === ‘noteon’)
				this.noteOn(message.data1, message.data2);
		}
	};

	navigator.requestMIDIAccess().then(function (midiAccess) {
		var input = midiAccess.inputs().values().next();
		input.onmidimessage = synth.onmidimessage.bind(synth);
	}, function (errorMessage) {
		console.log(‘nope... ‘ + message);
	});
</script>
```

Spiral to the rescue!

https://github.com/hoch/spiral

Spiral::MIDIManager

- MIDI system abstraction: router, sources and targets.
- Thenable `start()` method to populate the underlying MIDI system.

```
var MIDIMan = Spiral.createMIDIManager();
MIDIMan.start().then(
      function (MIDI) {
            var labels = MIDI.report();
      },                                                    Resolution
      function (errorMessage) {
            /* MIDIManager couldn't start for some reason... */
      }                                                     Rejection
);
```

MIDIManager.report()

- Returns the name of populated sources and targets in JSON format.

```
var names = Spiral.MIDI.report();
```

```
{
      sources: [‘myKeys’, ‘myPads’, ‘myKnobs’ ...],
      targets: [‘mySynth’, ‘myDrums’ ...]
}
```

Definition of MIDITarget

- Register an object with `onmidimessage` handler with a label.

```
var synth = {};
synth.onmidimessage = function (message) {
	if (message.type === ‘noteon’)
		console.log(‘pitch = ’ + message.data1 + ‘ velocity = ’ + message.data2);
};

// Define a target inside of the promise resolver.
MIDIMan.start().then(function (MIDI) {
	MIDI.defineTarget(‘mySynth’, synth);
}, function () {});
```

Parsing MIDI Message

- Call `Spiral.parseMIDIMessage()` to parse the data manually.
- Connect with `connect().to()` and get the data parsed automatically.

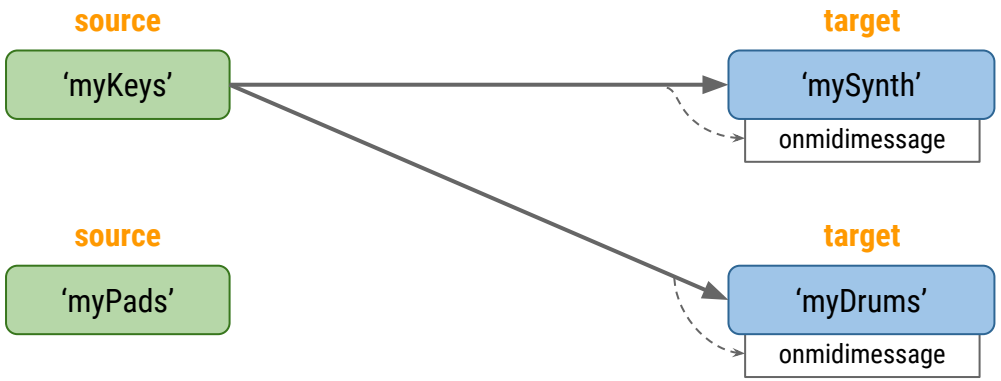
```
var parsedData = Spiral.parseMIDIMessage(midiMessage);
```

```
{
      type: ‘noteon’,          // Data type: ‘noteon’, ‘noteoff’, ‘controlchange’, ‘pitchwheel’...
      channel: 1,              // MIDI Channel: 1~16
      data1: 64,               // Data byte 1: pitch for ‘noteon’ type.
      data2: 96                // Data byte 2: velocity for ‘noteon’ type.
}
```

Routing

- Supports one-to-many routing between sources and targets.

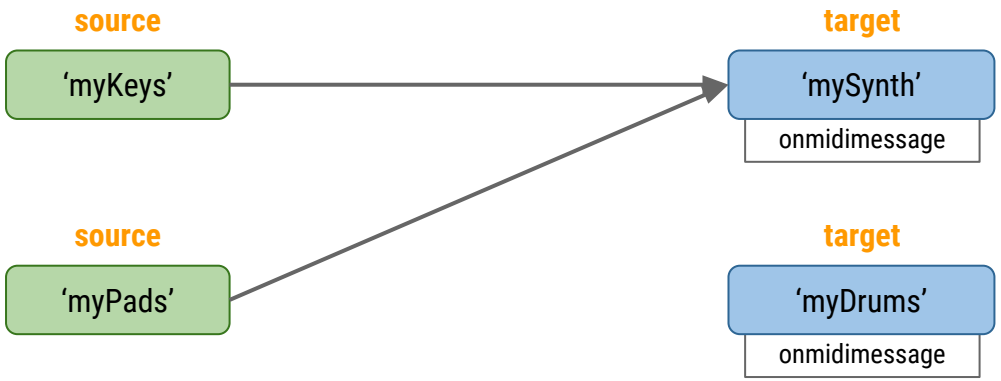
```
MIDI.connect('myKeys').to('mySynth', 'myDrums');      // Do this in the promise resolver.
```



Routing

- Or use `connectAll()` to route all the available inputs to a target.

```
MIDI.connectAll().to('mySynth');                    // Do this in the promise resolver.
```



Walkthrough: Spiral MIDI

```
<script src="spiral.min.js"></script>
<script>
	var synth = {
		noteOn: function (pitch, velocity) {
			/* play sound with pitch and velocity... */
		},
		onmidimessage: function (message) {
			if (message.type === 'noteon')
				this.noteOn(message.data1, message.data2);
		}
	};

	Spiral.createMIDIMananger().start().then(function (MIDI) {
		MIDI.defineTarget('mySynth', synth);
		MIDI.connectAll().to('mySynth);
	}, function (error) { console.log(error); });
</script>
```

Food for thoughts...

- `MIDIMessageEvent.timeStamp`
- Security concerns...

enjoy your coffee!

Let’s Chat!

Workshop Day 2 / Lecture 3