AUDIO MOSTLY 2023 WORKSHOP ON SUPERCOLLIDER

MIDI SYNTHESIZERS FOR MUSIC AND SONIFICATION

NIKLAS RÖNNBERG

LINKÖPING UNIVERSITY
niklas.ronnberg@liu.se

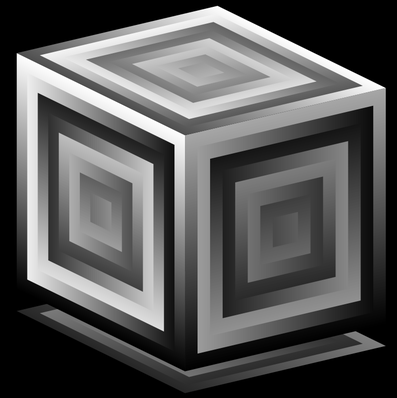
THIS WORKSHOP

- Introduction to SuperCollider
  - Presentation
  - Coding session
  - Discussion
- Subtractive sound synthesis
  - Presentation
  - Coding session
  - Discussion
- Additive sound synthesis
  - Presentation
  - Coding session
  - Discussion
- Sample sound synthesis
  - Presentation
  - Coding session
  - Discussion

- Equipment
  - Laptop
  - Headphones
  - MIDI keyboard

SUPERCOLLIDER - 1996...

- SuperCollider is pretty old...
- ...with an ugly syntax...
- ...some weird behaviour...
- ...and amazing possibilities...
- ...and it is incredibly fun!



SUPERCOLLIDER

SUPERCOLLIDER

SUPERCOLLIDER

```
(
 code
    code
    …
    …
    …
    …
)
```

Client side

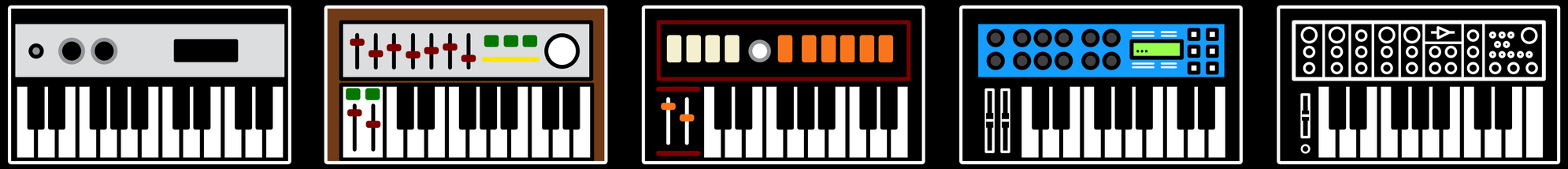
Server side

SUPERCOLLIDER

```
(
 code
    code
    …
    …
    …
    …
)
```

Client side

Server side



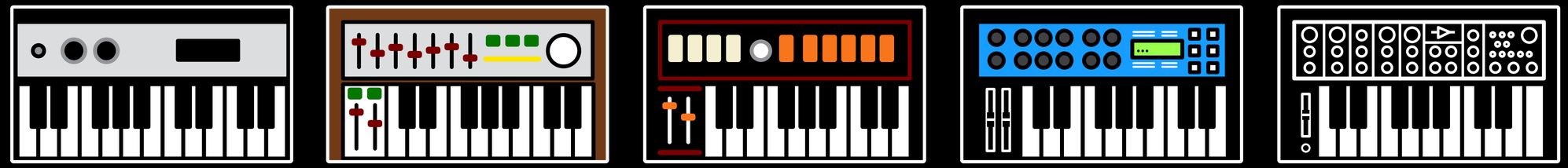
Synth definitions

SUPERCOLLIDER

```
(
 code
    code
    …
    …
    …
    …
)
```

Client side

Server side



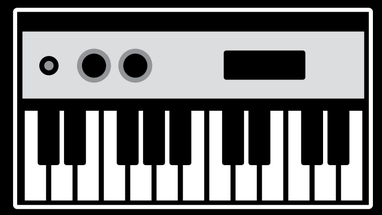
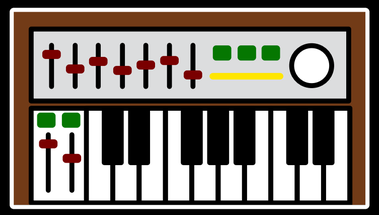
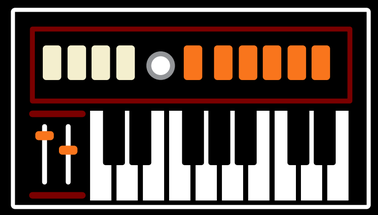
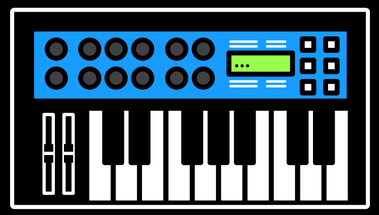
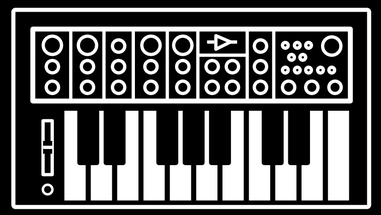
Synth definitions

SUPERCOLLIDER

```
(
fork{
    loop{
        h = ([36,39,43].choose.midicps) * (2**((0 .. 4).choose));
        play{
            Splay.ar({
                SinOsc.ar(exprand(h-(h/64),h+(h/64)),0,0.1)}!8) * LFGauss.kr(10,1/4,0,0,2);
        };
        0.5.wait;
    };
};
)
```

Client side

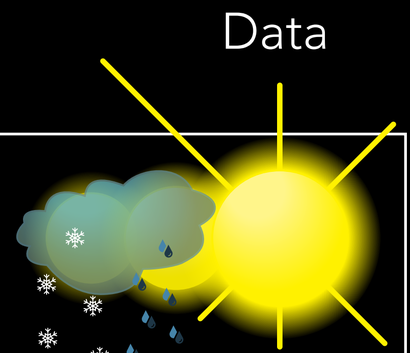
Server side

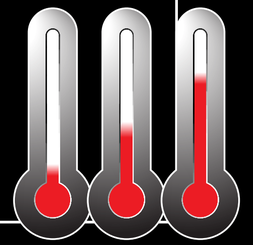
Synth definitions

SUPERCOLLIDER

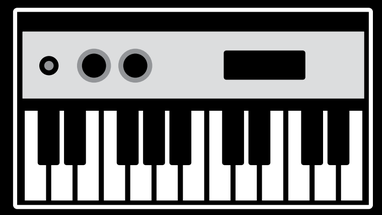
```
(
fork{
    loop{
        h = ([36,39,43].choose.midicps) * (2**((0 .. 4).choose));
        play{
            Splay.ar({
                SinOsc.ar(exprand(h-(h/64),h+(h/64)),0,0.1)}!8) * LFGauss.kr(10,1/4,0,0,2);
        };
        0.5.wait;
    };
};
)
```

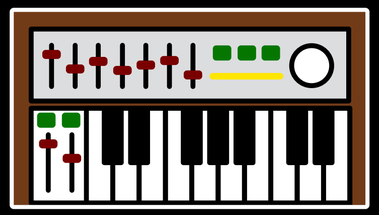


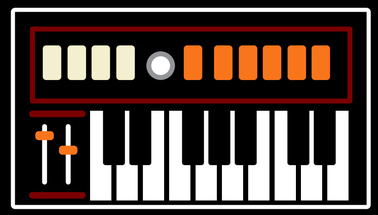
Client side

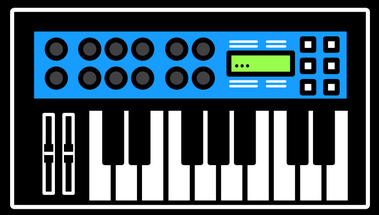


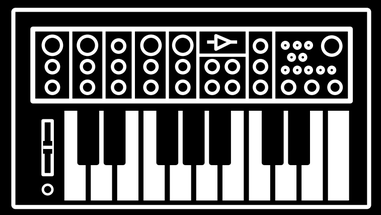
Server side











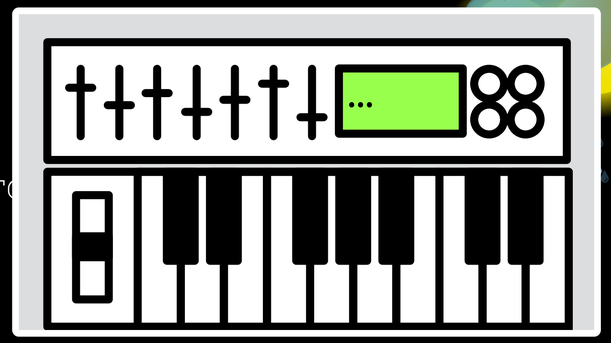
Synth definitions

SUPERCOLLIDER

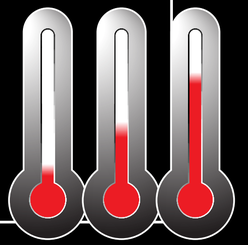
```
(
fork{
    loop{
        h = ([36,39,43].choose.midicps) * (2**((0 .. 4).choose));
        play{
            Splay.ar({
                SinOsc.ar(exprand(h-(h/64),h+(h/64)),0,0.1)}!8) * LFC
        };
        0.5.wait;
    };
};
)
```

External controls

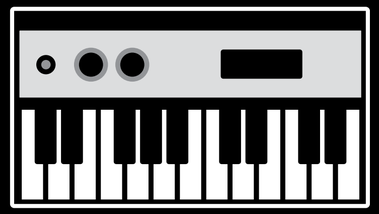
Data

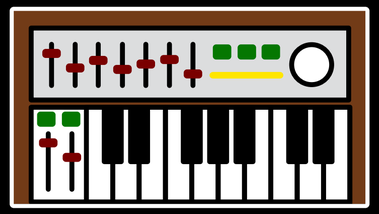


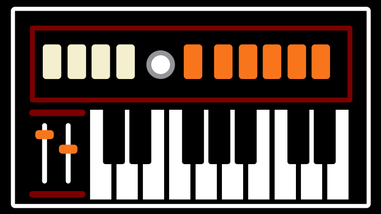
Client side

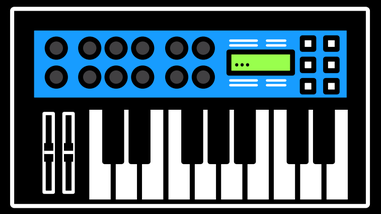


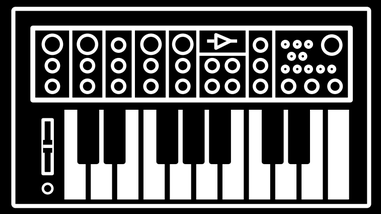
Server side



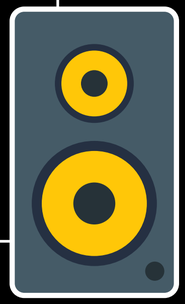








Synth definitions



SUPERCOLLIDER

Code editor

Help browser

Post window

Start the server…

Start executing code with:

<Ctrl> and <Enter>

or

<Cmd> and <Enter>

Start the server…

(
  Then execute code …
)

fork = a fork is a parallel process

```
(
  fork{

  };
)
```

loop = a loop that runs code

```
(
  fork{
        loop{

        };
  };
)
```

wait = halts execution of code (seconds)

```
(
  fork{
        loop{
              1.wait;
        };
  };
)
```

choose = selects one at random

```
(
  fork{
        loop{
              h = ([33, 38, 40].choose);
              1.wait;
        };
  };
)
```

postln = post a line to the post window

```
(
  fork{
        loop{
              h = ([33, 38, 40].choose);
              h.postln;
              1.wait;
        };
  };
)
```

```
(
  fork{
        loop{
              h = ([33, 38, 40].choose);
              h.postln;
              1.wait;
        };
  };
)
```

Stop executing code with:

<Ctrl> and <.>

or

<Cmd> and <.>

Start executing a marked line with:

<Shift> and <Enter>

```
(
  fork{
        loop{
              h = ([33, 38, 40].choose);
              h.postln;
              1.wait;
        };
  };
)
```

```
(
  fork{
        loop{
              h = ([33, 38, 40].choose.midicps);
              h.postln;
              1.wait;
        };
  };
)
```

midi = musical instrument digital interface

midinote number = 0 to 127

0 = 8.18Hz and 127 = 12543.85Hz

cps = cycles per seconds

cycles per seconds = Hz

```
(
 fork{
      loop{
           h = ([33, 38, 40].choose.midicps);
           play{
                Splay.ar({SinOsc.ar(h)});
           };
           1.wait;
      };
 };
)
```

Splay = spreads sounds in stereo

SinOsc = sine wave oscillator

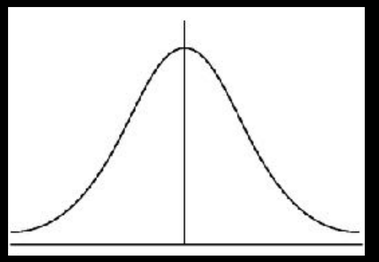
ar = audio rate

** = to the power of

```
(
  fork{
        loop{
              h = ([33, 38, 40].choose.midicps) * (2**((0 .. 4).choose));
              play{
                    Splay.ar({SinOsc.ar(h)});
              };
              1.wait;
        };
  };
)
```

LFGauss = a gaussian curve

kr = control rate



```
(
  fork{
        loop{
              h = ([33, 38, 40].choose.midicps) * (2**((0 .. 4).choose));
              play{
                    Splay.ar({SinOsc.ar(h)}) * LFGauss.kr(10, 1/4, 0, 0, 2);
              };
              1.wait;
        };
  };
)
```

```
(
  fork{
        loop{
              h = ([33, 38, 40].choose.midicps) * (2**((0 .. 4).choose));
              play{
                    Splay.ar({SinOsc.ar(exprand(h - (h/64), h + (h/64)), 0, 0.1}) * LFGauss.kr(10, 1/4, 0, 0, 2);
              };
              1.wait;
        };
  };
)
```

exprand = exponential random number

s.scope;

Server.local.plotTree;

!8 = make eight in parallel

```
(
  fork{
        loop{
              h = ([33, 38, 40].choose.midicps) * (2**((0 .. 4).choose));
              play{
                    Splay.ar({SinOsc.ar(exprand(h - (h/64), h + (h/64)), 0, 0.1}!8) * LFGauss.kr(10, 1/4, 0, 0, 2);
              };
              0.5.wait;
        };
  };
)
```

LFTri = triangle wave oscillator

```
(
  fork{
        loop{
              h = ([33, 38, 40].choose.midicps) * (2**((0 .. 4).choose));
              play{
                    Splay.ar({LFTri.ar(exprand(h - (h/64), h + (h/64)), 0, 0.1}!8) * LFGauss.kr(10, 1/4, 0, 0, 2);
              };
              0.5.wait;
        };
  };
)
```

LFSaw = sawtooth wave oscillator

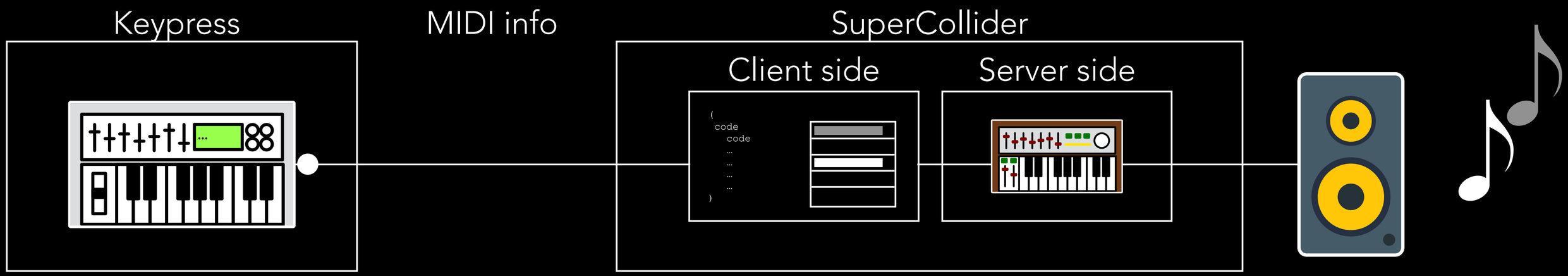
```
(
  fork{
        loop{
              h = ([33, 38, 40].choose.midicps) * (2**((0 .. 4).choose));
              play{
                    Splay.ar({LFSaw.ar(exprand(h - (h/64), h + (h/64)), 0, 0.1}!8) * LFGauss.kr(10, 1/4, 0, 0, 2);
              };
              0.5.wait;
        };
  };
)
```

LET'S GET STARTED

- SuperCollider
  https://supercollider.github.io/
- Go to Downloads
- Choose the current version (3.13.0) suitable for your system
- Download SuperCollider
- Install SuperCollider
- Then start SuperCollider

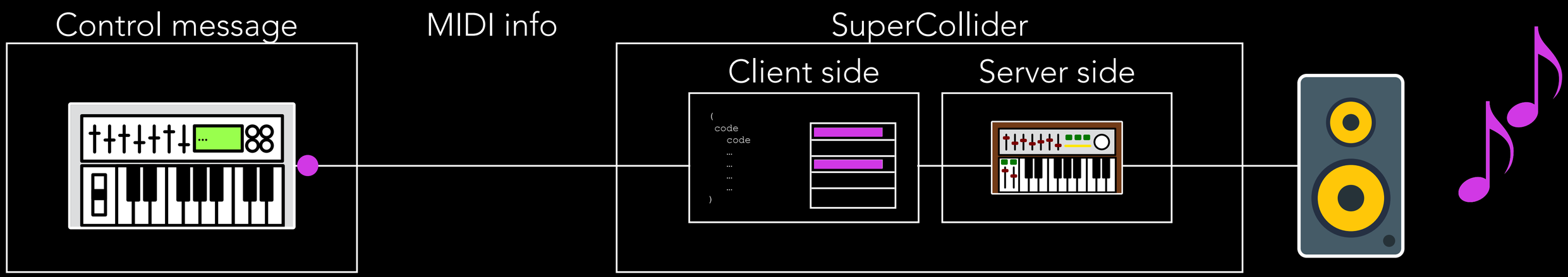
LET'S GET STARTED - MIDI

- MIDI is received by starting a MIDI client and connecting this to all MIDI inputs
- By using a MIDIFunc for noteOn messages, SuperCollider receives keypresses
- Every time a key is pressed a new synth instance is created on the server, and the reference to this is stored in an array (a list).



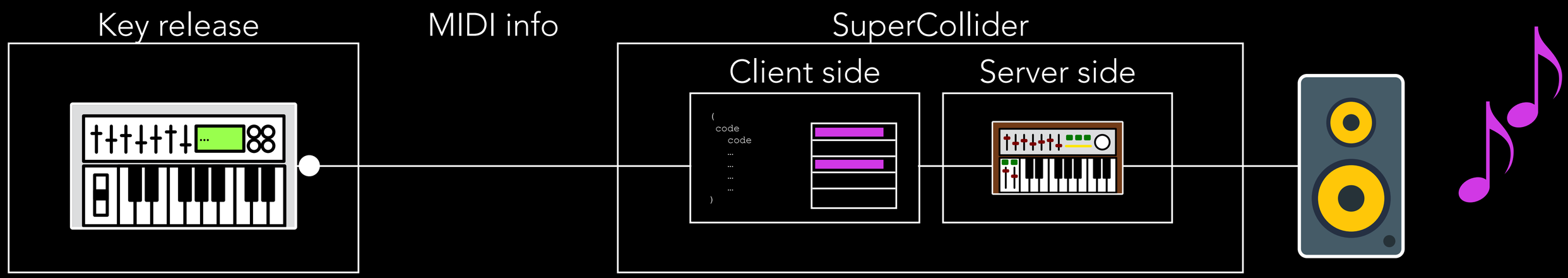
LET'S GET STARTED - MIDI

- If a control message are sent from the keyboard/MIDI controller
- The client side adjusts all synths available in the array in relation to the control message



LET'S GET STARTED - MIDI

- With a MIDIFunc for noteOff messages, SuperCollider receives key releases
- Every time a key is released the synth instance will be removed from the server and the array.



LET'S GET STARTED

- Workshop examples and extras
  https://www.itn.liu.se/~nikro27/am2023_ws/