Q+Faust+SuperCollider (LAC 2006)

Albert Gräf

Dept. of Music Informatics

JOHANNES GUTENBERG UNIVERSITÄT MAINZ

Featuring: Recent advances in the functional programming language Q. Interfaces to Faust + SuperCollider. Collaboration with Yann Orlarey (Grame) and Stefan Kersten (TU Berlin).

JOHANNES GUTENBERG UNIVERSITÄT MAINZ

About Q

programs are collections of algebraic equations

```
sqr X = X*X;
```

expression evaluation by term rewriting

```
sqr 2+sqr (2+3) ⇒ 2*2+sqr (2+3)
  ⇒ 4+sqr (2+3) ⇒ 4+sqr 5
  ⇒ 4+5*5 ⇒ 4+25 ⇒ 29
```

conditional equations and tail recursion

```
gcdiv X Y = gcdiv Y (X mod Y) if Y>0;
          = X otherwise;
```

local variable definitions

```
fib N  = A where (A,B) = fibs N;
fibs N = (0,1) if N<=0;
       = (B,A+B) where (A,B) = fibs (N-1);
```

pattern matching and higher-order functions

```
qsort []     = [];
qsort [X|Xs] = qsort (filter (<X) Xs) ++
  [X] ++ qsort (filter (>=X) Xs);
```

Why Functional Programming?

- General FP advantages: powerful abstractions, mathematical elegance, higher-order functions, referential transparency.
- Added benefits of “modern-style” FP: Currying, algebraic data types, equational definitions, pattern matching, lazy evaluation. Better abstractions, less code.
- Special benefits for real-time multimedia applications: It's all about processing signals and event sequences, which can easily be modelled as higher-order functions and streams.

The Main Ingredients of Q

Term rewriting as a programming language (O'Donnell et al 1985)

+ rule priorities, conditions, local variable definitions

+ modern style FP syntax (infix application, currying, higher-order functions)

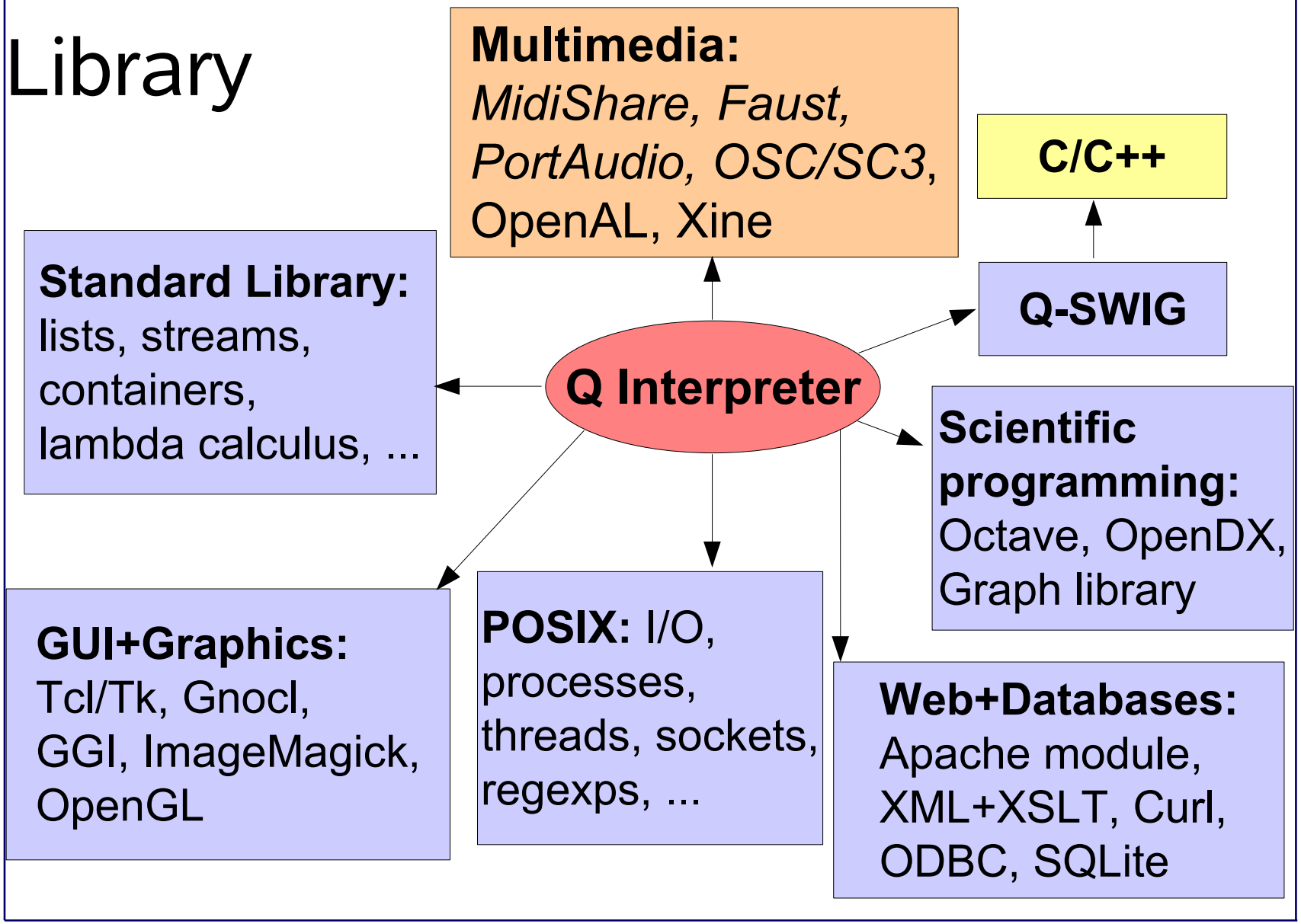
+ algebraic data types (free term algebra)

+ dynamic OO typing (Smalltalk style)

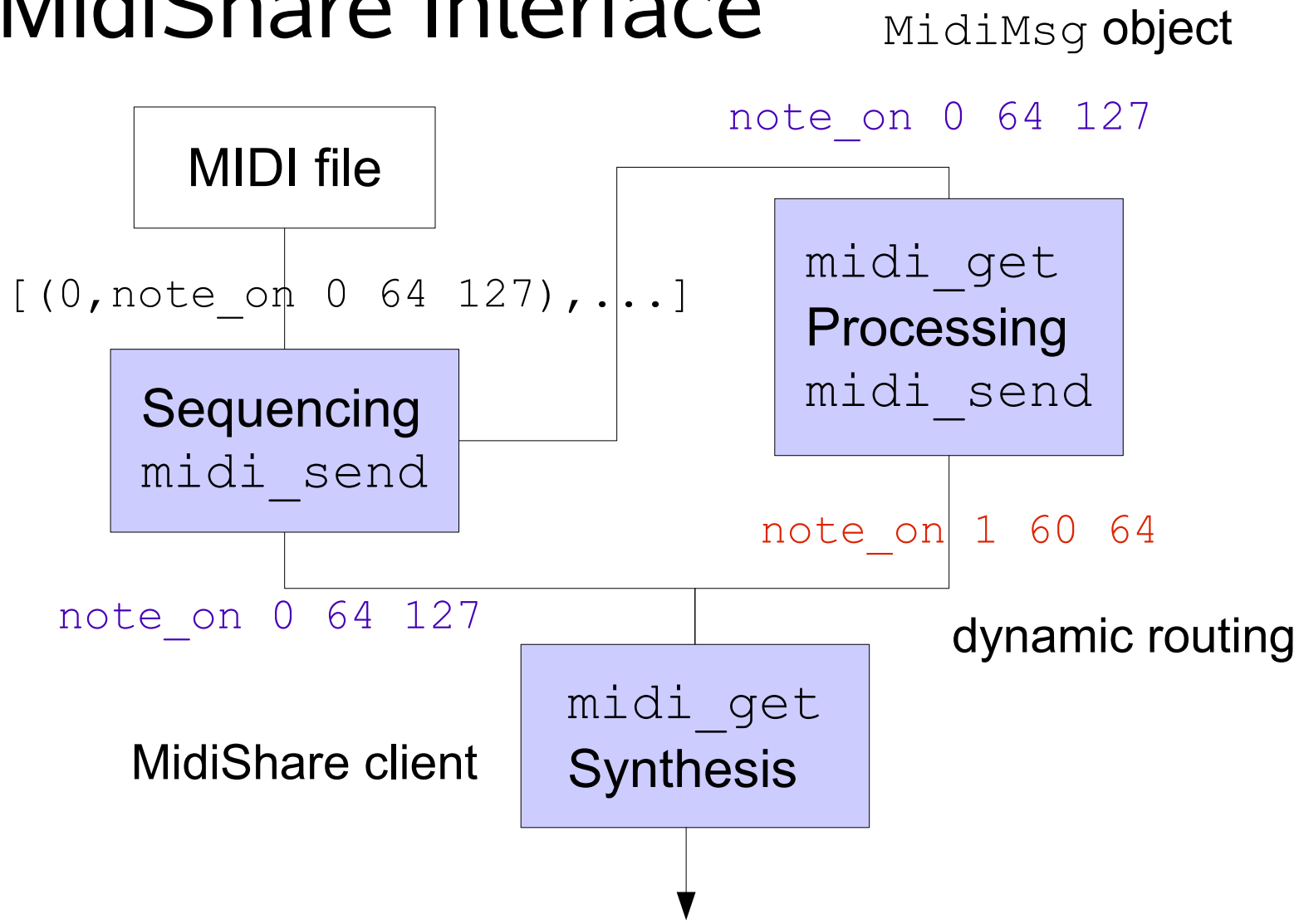
+ special forms (lazy evaluation)

JOHANNES GUTENBERG UNIVERSITÄT MAINZ

Library

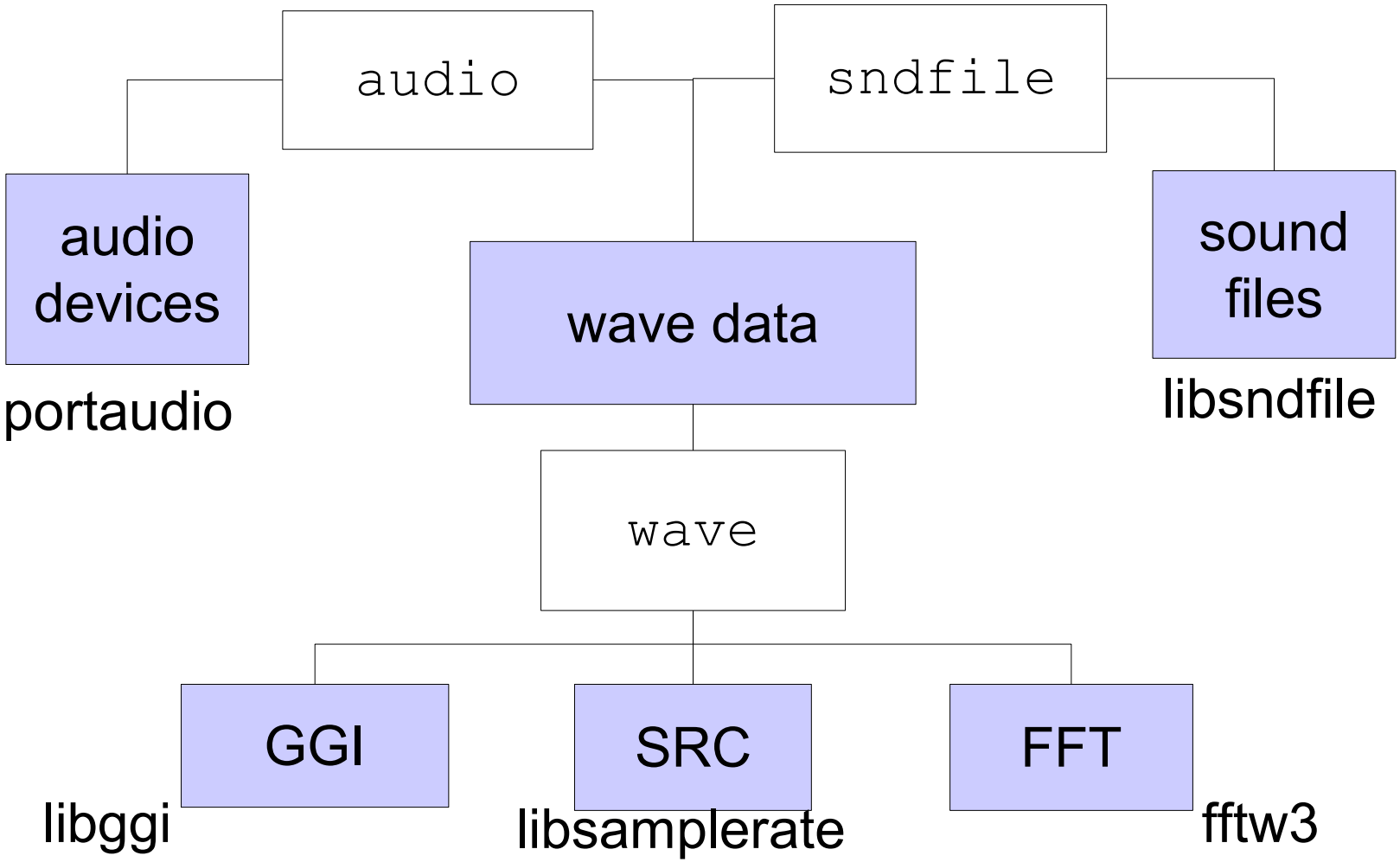


MidiShare Interface



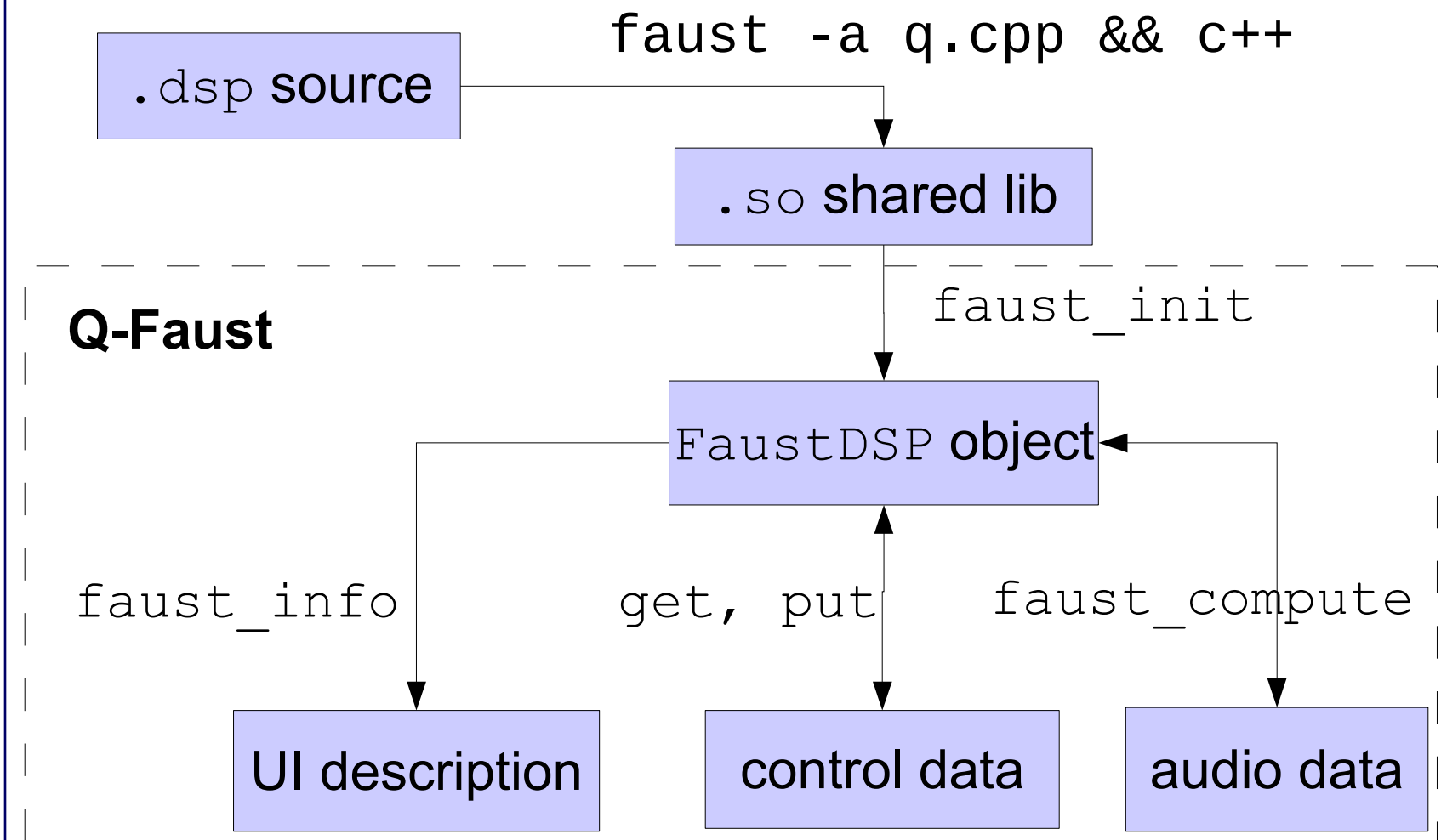
JOHANNES GUTENBERG UNIVERSITÄT MAINZ

Audio Interface



JOHANNES GUTENBERG UNIVERSITÄT MAINZ

Faust Interface



Faust Ex. 1: A Simple Organ

Demo

```
// control variables

vol     = nentry("vol", 0.3, 0, 10, 0.01);     // %
attack  = nentry("attack", 0.01, 0, 1, 0.001); // sec
decay   = nentry("decay", 0.3, 0, 1, 0.001);   // sec
sustain = nentry("sustain", 0.5, 0, 1, 0.01);  // %
release = nentry("release", 0.2, 0, 1, 0.001); // sec
freq    = nentry("freq", 440, 20, 20000, 1);   // Hz
gain    = nentry("gain", 1.0, 0, 10, 0.01);    // %
gate    = button("gate");                      // 0/1

// additive synth: 3 sine oscillators with adsr envelop

process = (osc(freq)+0.5*osc(2*freq)+0.25*osc(3*freq))
  * (gate : adsr(attack, decay, sustain, release))
  * gain * vol;
```

Q Ex. 1: Monophonic Synthesizer

Demo

```
def [FREQ,GAIN,GATE] = map (CTLD!)
  ["freq","gain","gate"];

freq N    = 440*2^((N-69)/12);
gain V    = V/127;

play N V  = put FREQ (freq N) || put GAIN (gain V) ||
            put GATE 1;
damp      = put GATE 0;

process (_,_,_,note_on _ N V)
          = play N V if V>0;
          = damp if not isnum (get FREQ) or else
                    (freq N = get FREQ);

midi_loop = process (midi_get IN) || midi_loop;
```

map MIDI note numbers and velocities

start and stop a note

process MIDI messages

Faust Ex. 2: Eight Voices

```
freq(i)  = nentry("freq%i", 440, 20, 20000, 1); // Hz
gain(i)  = nentry("gain%i", 0.3, 0, 10, 0.01);  // %
gate(i)  = button("gate%i");                    // 0/1

// one synth per voice

voice(i) = (osc(freq(i))+0.5*osc(2*freq(i))+
            0.25*osc(3*freq(i)))
  * (gate(i) : adsr(attack, decay, sustain, release))
  * gain(i);

nvoices = 8;

process = sum(i, nvoices, voice(i)) * vol;
```

voice index

Q Ex. 2: Polyphonic Synthesizer

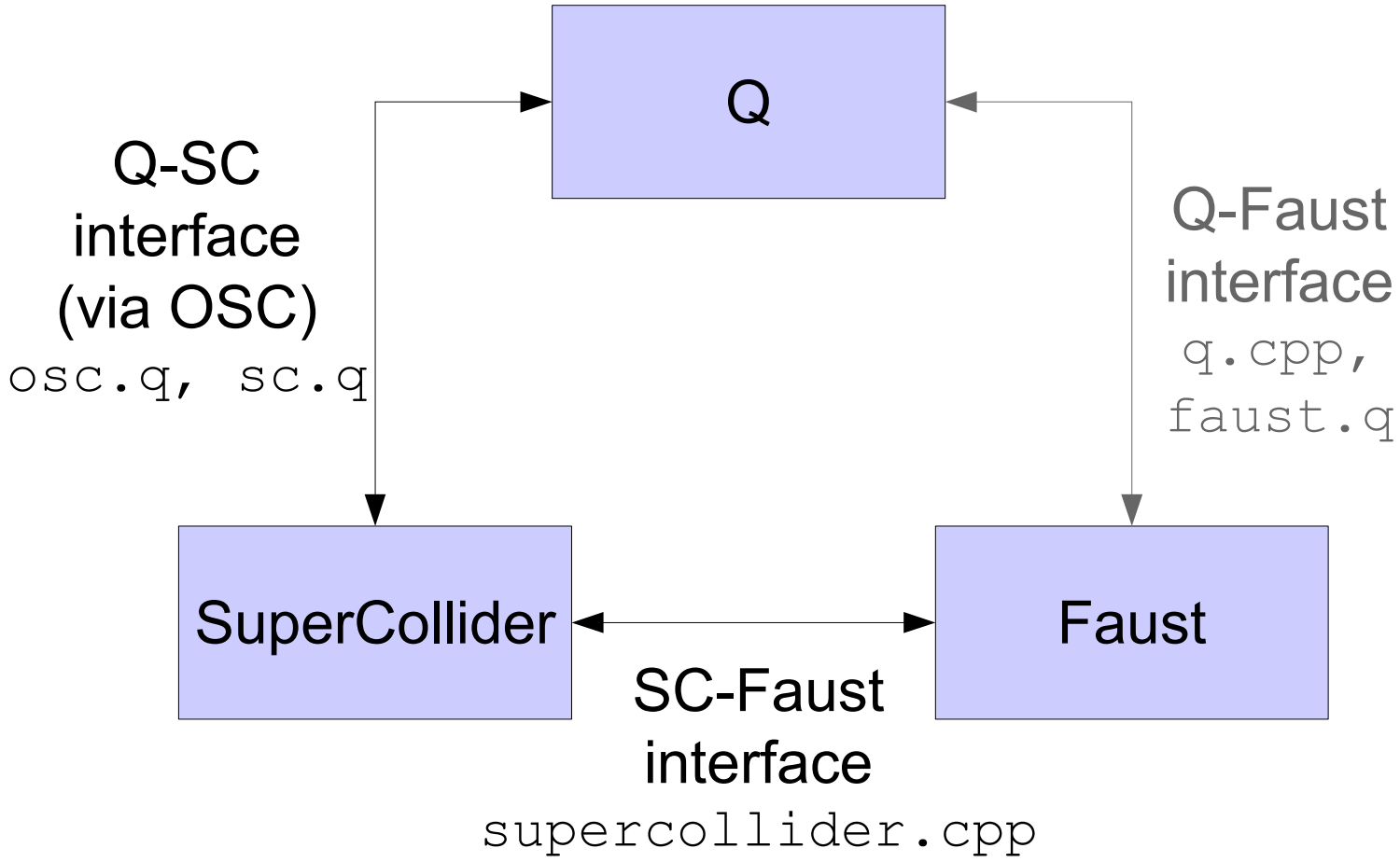
Demo

```
/* Simple voice allocation algorithm. */

init_voices  = ([],[0..7]);

new_voice (P,Q) N V
             = play I N V || (P,Q)
                 where [I|Q] = Q, P = append P (I,N);
             = damp I || sleep 0.01 ||
               play I N V || (P,Q)
                 where [(I,_)|P] = P, P = append P (I,N);

free_voice ([(I,N)|P],Q) N
             = damp I || (P,append Q I);
free_voice ([X|P],Q) N
             = ([X|P],Q) where (P,Q) = free_voice (P,Q) N;
free_voice ([],Q) _
             = ([],Q);
```

"steal" a voice

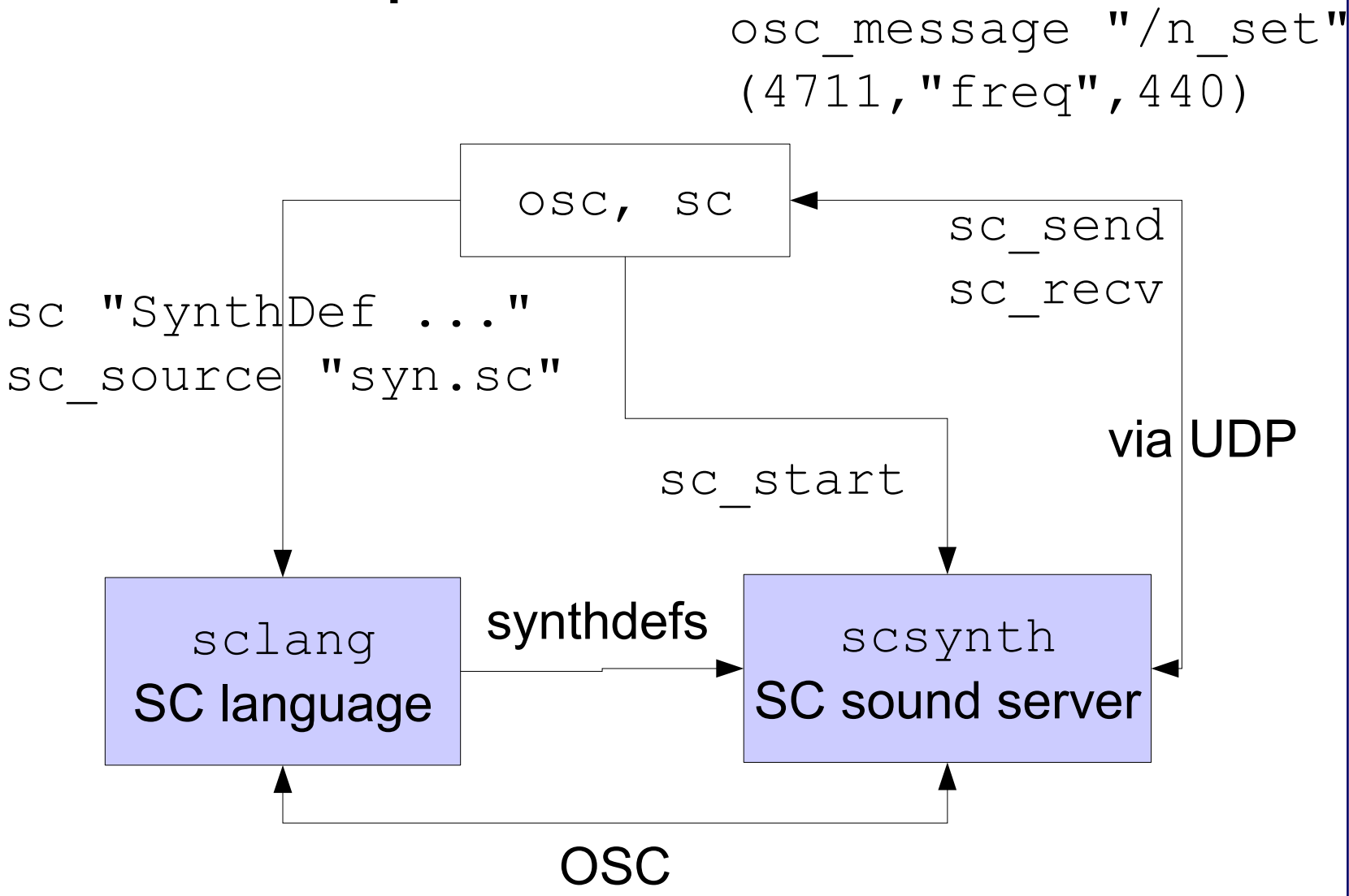
JOHANNES GUTENBERG UNIVERSITÄT MAINZ

Q+SuperCollider+Faust



JOHANNES GUTENBERG UNIVERSITÄT MAINZ

Q-OSC/SuperCollider Interface

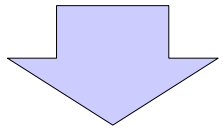


Q+Faust+SuperCollider

JOHANNES GUTENBERG UNIVERSITÄT MAINZ

Q

```
n_set ARGS = sc_send (osc_message "/n_set" ARGS);

play I N V = n_set (I,"freq",freq N,"gain",gain V,
                    "gate",1);

damp I     = n_set (I,"gate",0);
```



SuperCollider

```
SynthDef("organ",
{ arg gate = 0, freq = 440, gain = 0.3, vol = 1.0;
  var sig;
  sig = FOrgan.ar(gate: gate, freq: freq, vol: vol);
  Out.ar(0, Pan2.ar(sig, 0, 1)); })
```

Demo

Advantages of the Q+SC+Faust Combo

You get the best of **three worlds**:

- SuperCollider can be extended with plugins written in Faust.
- All hard real-time processing is done in SuperCollider.
- Q can be used to do the high-level control and provide the system and user interface.
- Existing SuperCollider code can be reused in Q applications.