About Q

algebraic types and functions defined by equations

```
type MidiMsg = const note_on K N V, …;
transpose M (note_on K N V)
                = note_on K (N+M) V;
```

currying and higher-order functions

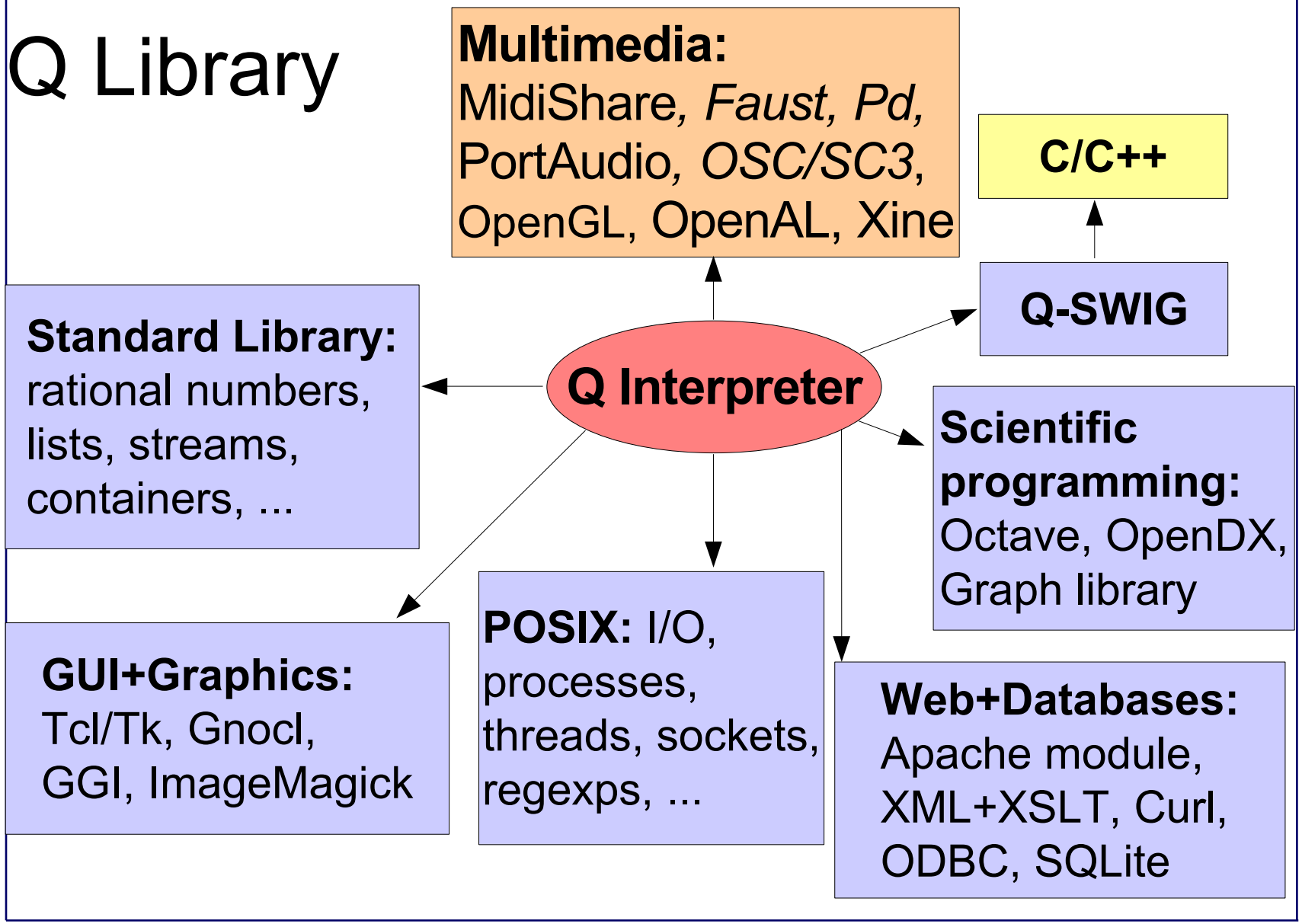
```
octave_up = map (transpose 12);
```

lazy evaluation and stream processing

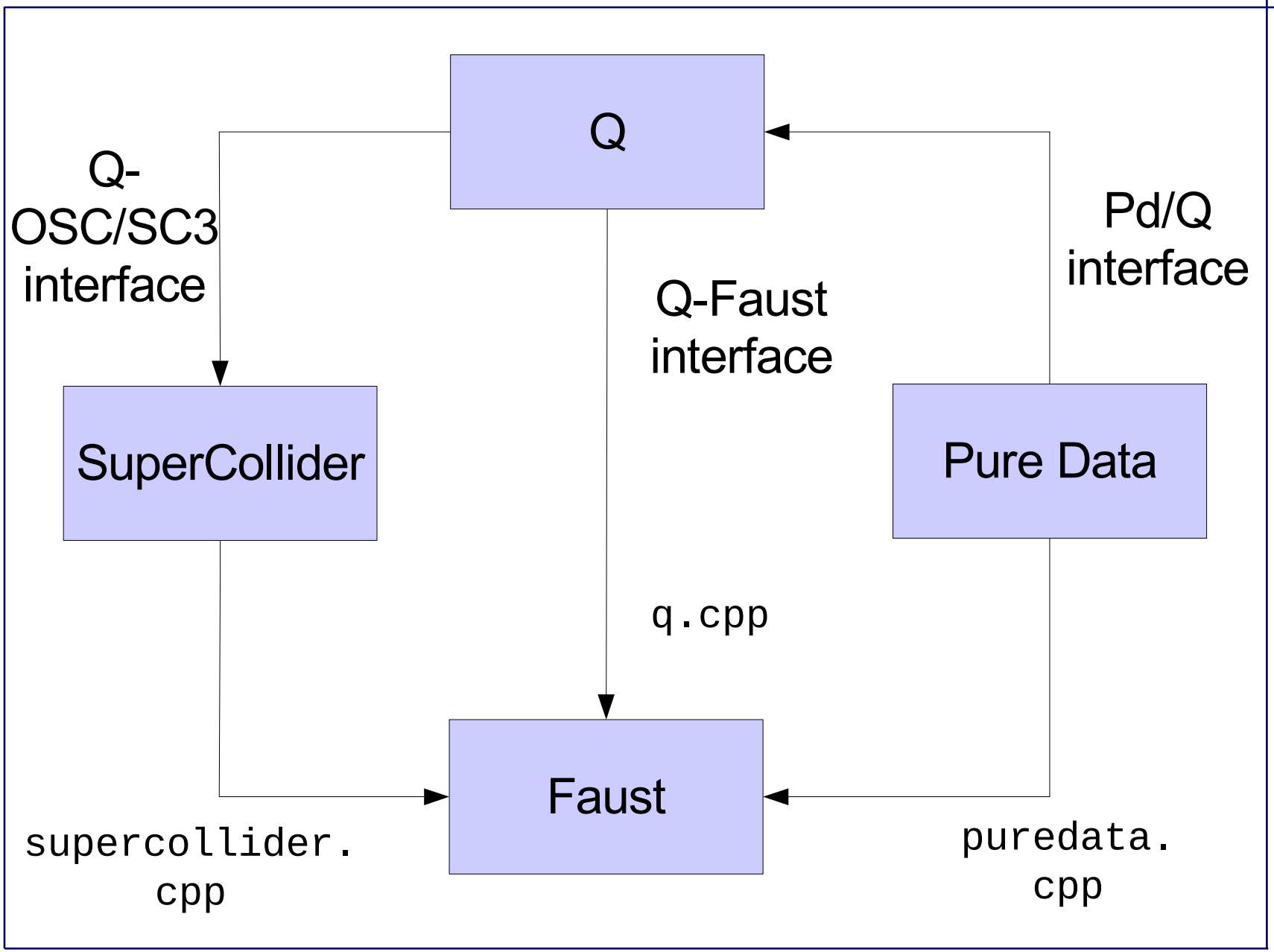
```
pattern   = repeat [60,60,choose [63,67]];
repeat X  = {X|repeat X};
choose Xs = Xs!rand 0 (#Xs-1);
```

- Haskell-like modern style syntax + Lisp-like dynamic typing and reflection capabilities
- multithreading and soft realtime processing, usable for “control stuff”
- extensive system and multimedia interfaces

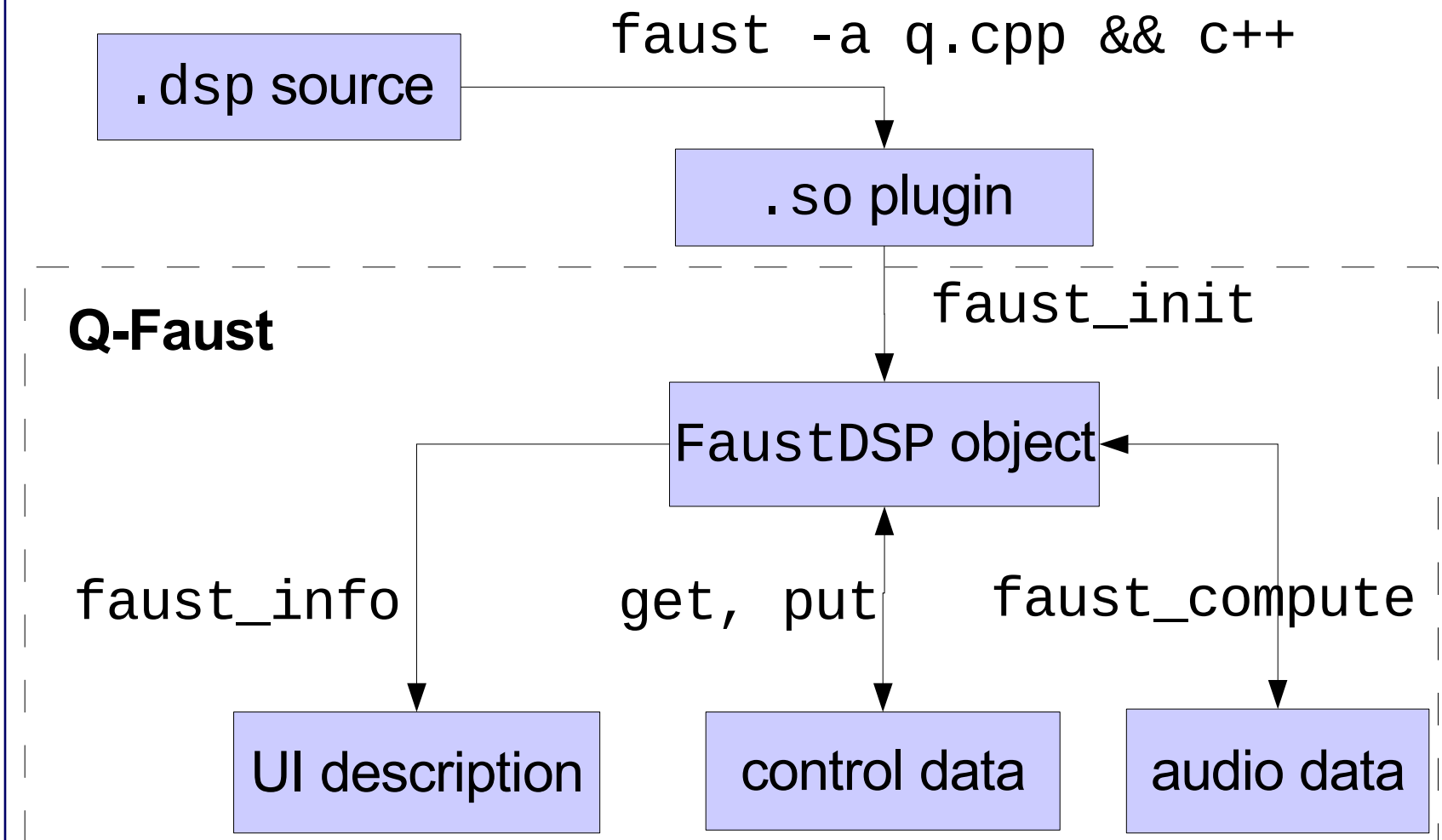
Q Library


Multimedia:
MidiShare, Faust, Pd,
PortAudio, OSC/SC3,
OpenGL, OpenAL, Xine
C/C++
Q-SWIG
Standard Library:
rational numbers,
lists, streams,
containers, ...
Q Interpreter
Scientific
programming:
Octave, OpenDX,
Graph library
GUI+Graphics:
Tcl/Tk, Gnocl,
GGI, ImageMagick
POSIX: I/O,
processes,
threads, sockets,
regexps, ...
Web+Databases:
Apache module,
XML+XSLT, Curl,
ODBC, SQLite

Q, Faust et al


Q
Q-
OSC/SC3
interface
Q-Faust
interface
Pd/Q
interface
SuperCollider
Pure Data
q.cpp
Faust
supercollider.
cpp
puredata.
cpp

Direct Faust Interface



A Simple Faust Organ

```
// 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 → Faust Example

```
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;
```

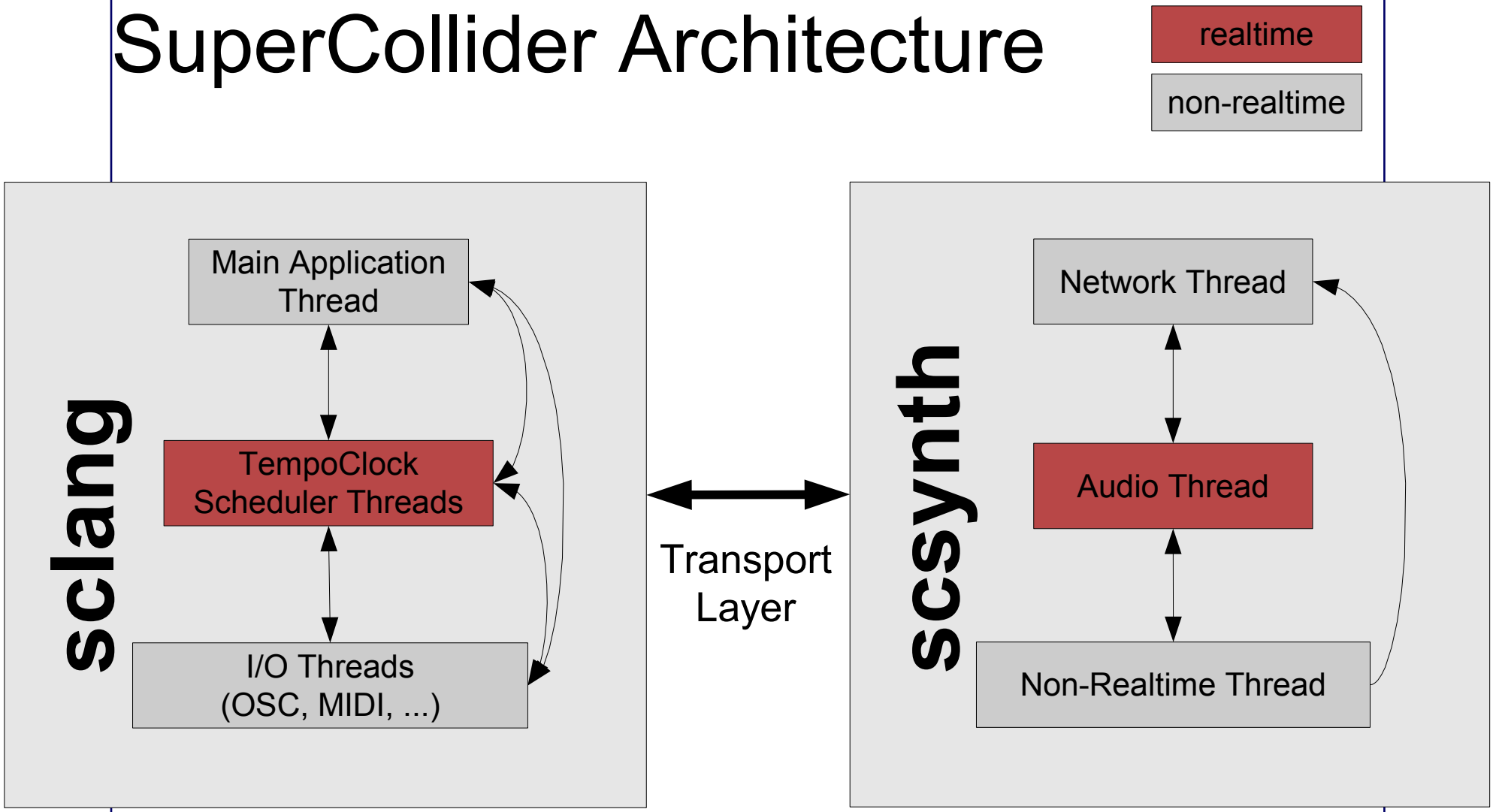
map MIDI note numbers and velocities

start and stop a note

process MIDI messages

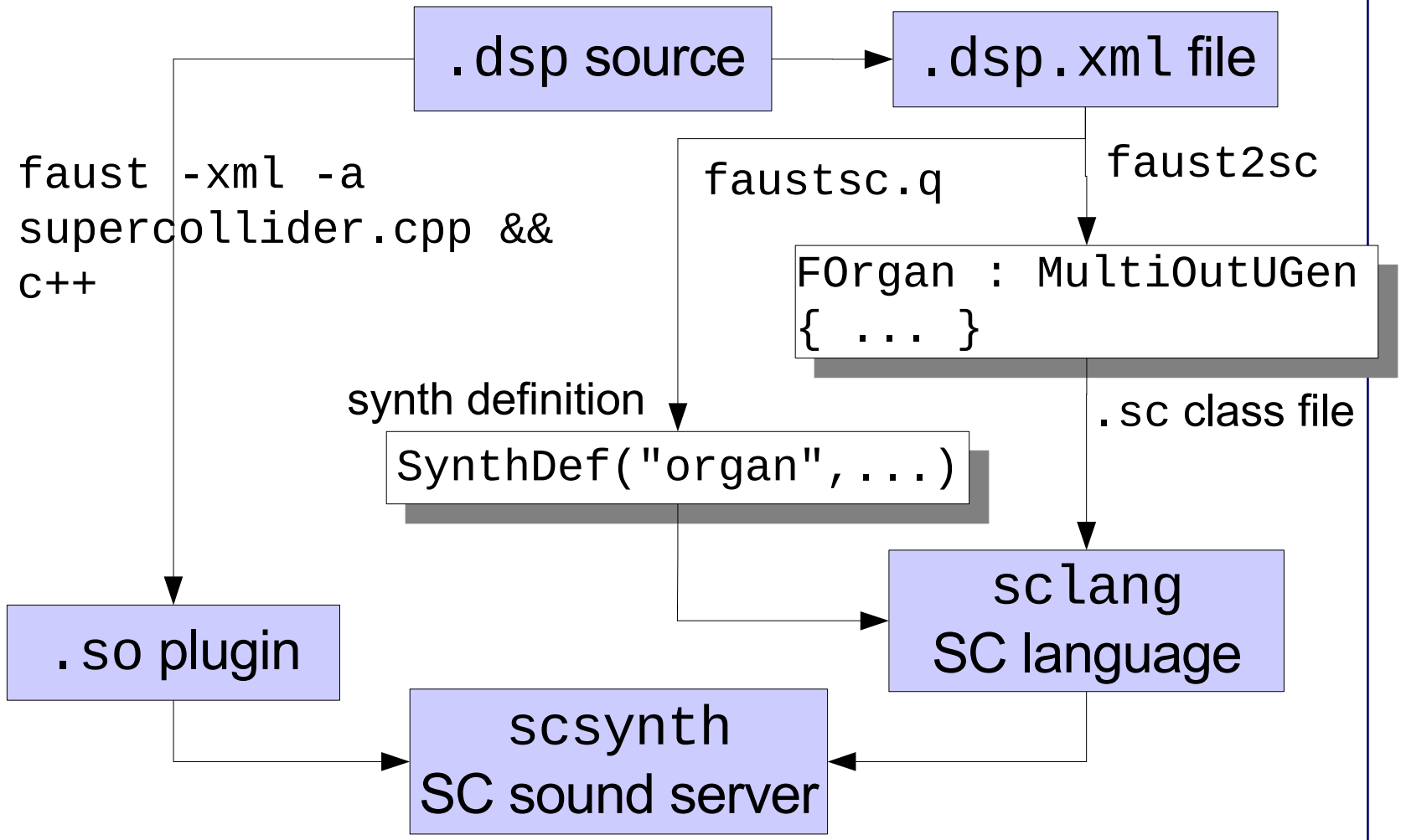
JOHANNES GUTENBERG UNIVERSITÄT MAINZ

SuperCollider Architecture



JOHANNES GUTENBERG UNIVERSITÄT MAINZ

SuperCollider Interface



Q → SC → Faust Example

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)); })
```

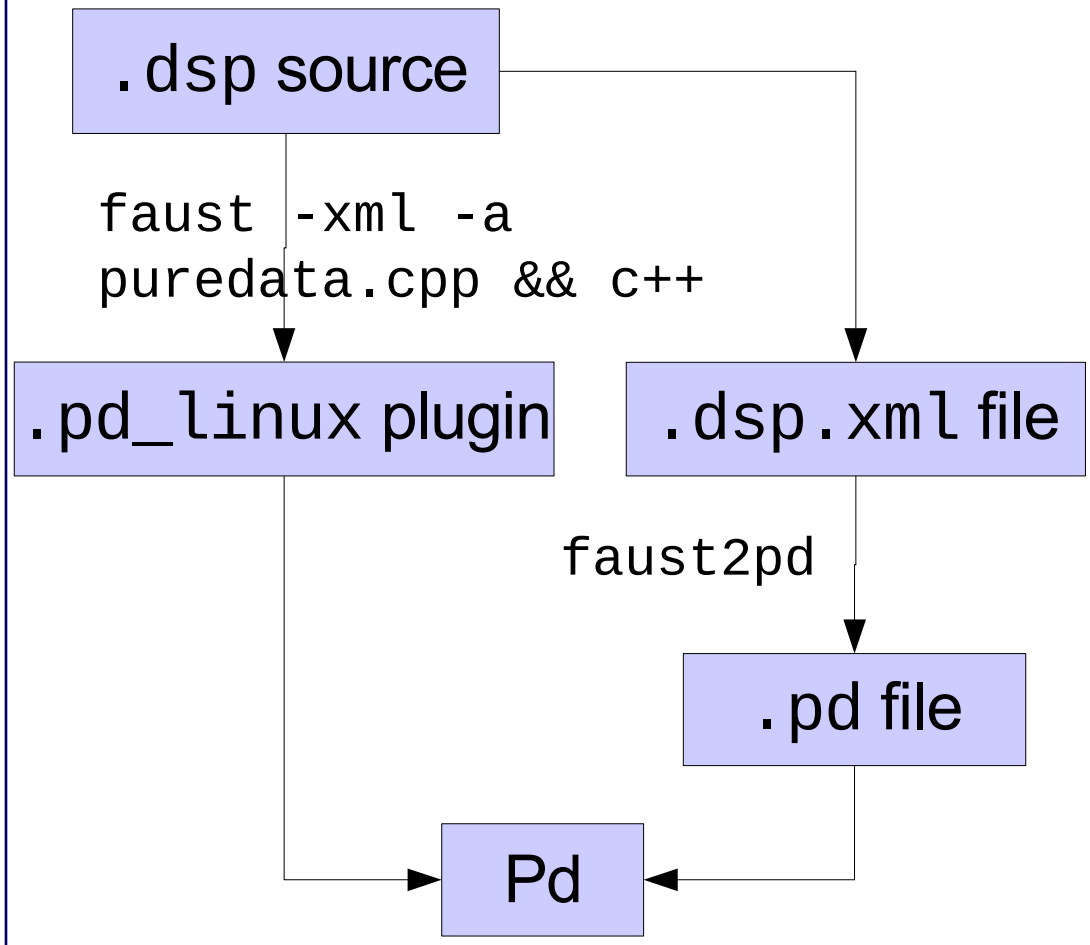
Q

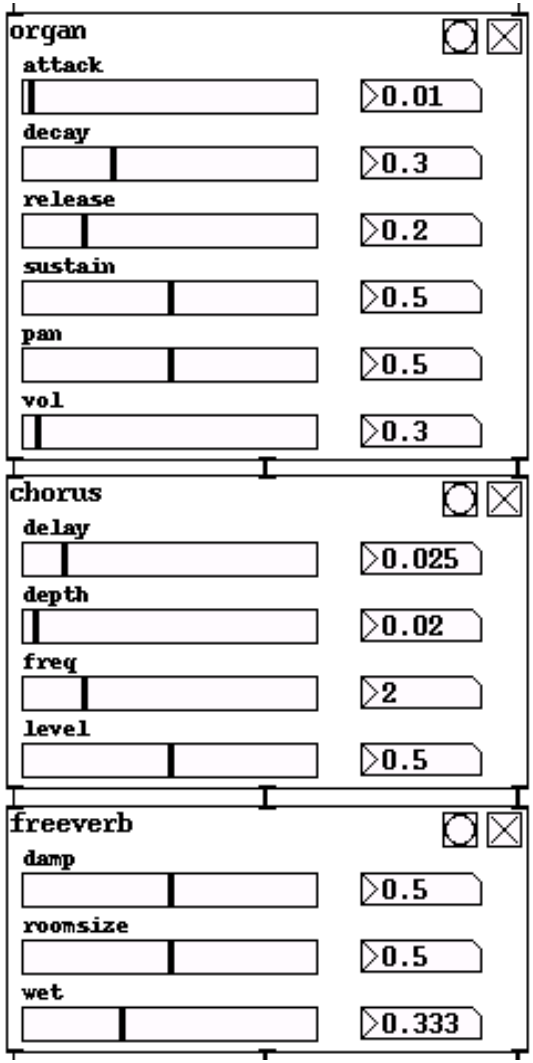
```
n_set ARGS = sc_send (osc_message "/n_set" ARGS);

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

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

Pd Interface





Conclusion

- Faust makes developing DSP modules for different environments easy
- Q+Faust together with a realtime engine (SC3, PD, ...) gives you a complete functional programming environment for interactive computer music applications
- TODO: high-level Q API for algorithmic composition