faust2clap: Integrating Faust DSP into the CLAP Ecosystem

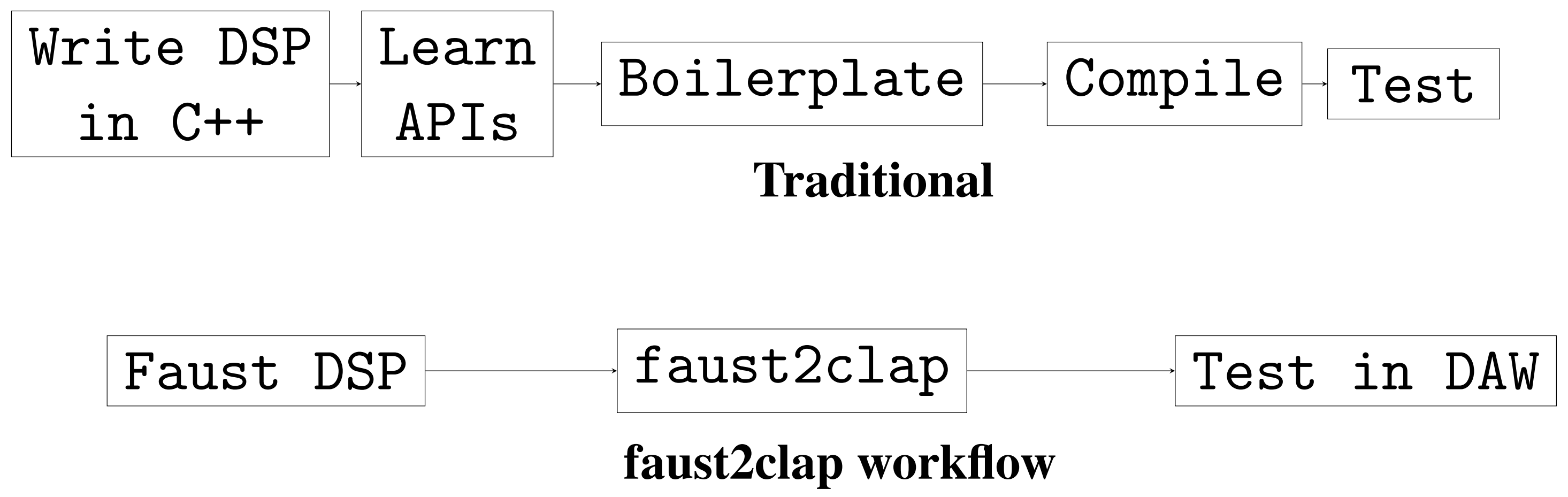
Facundo Franchino[1], *Stéphane Letz*[2], *Jatin Chowdhury*[3]

[1]*University of York, UK;* [2]*GRAME CNCM, France ;* [3]*Massachussetts Institute of Technology (MIT), USA.*

Introduction

faust2clap bridges high-level DSP code written in Faust with the modern CLAP plugin standard. It allows developers to turn Faust '.dsp' files into ready-to-use CLAP audio plugins with a single command, either as:

- **Static CLAP plugins:** compiled, optimised binaries ready for distribution
- **Dynamic CLAP plugins:** hot-reloadable plugins editable in real time while running inside a DAW



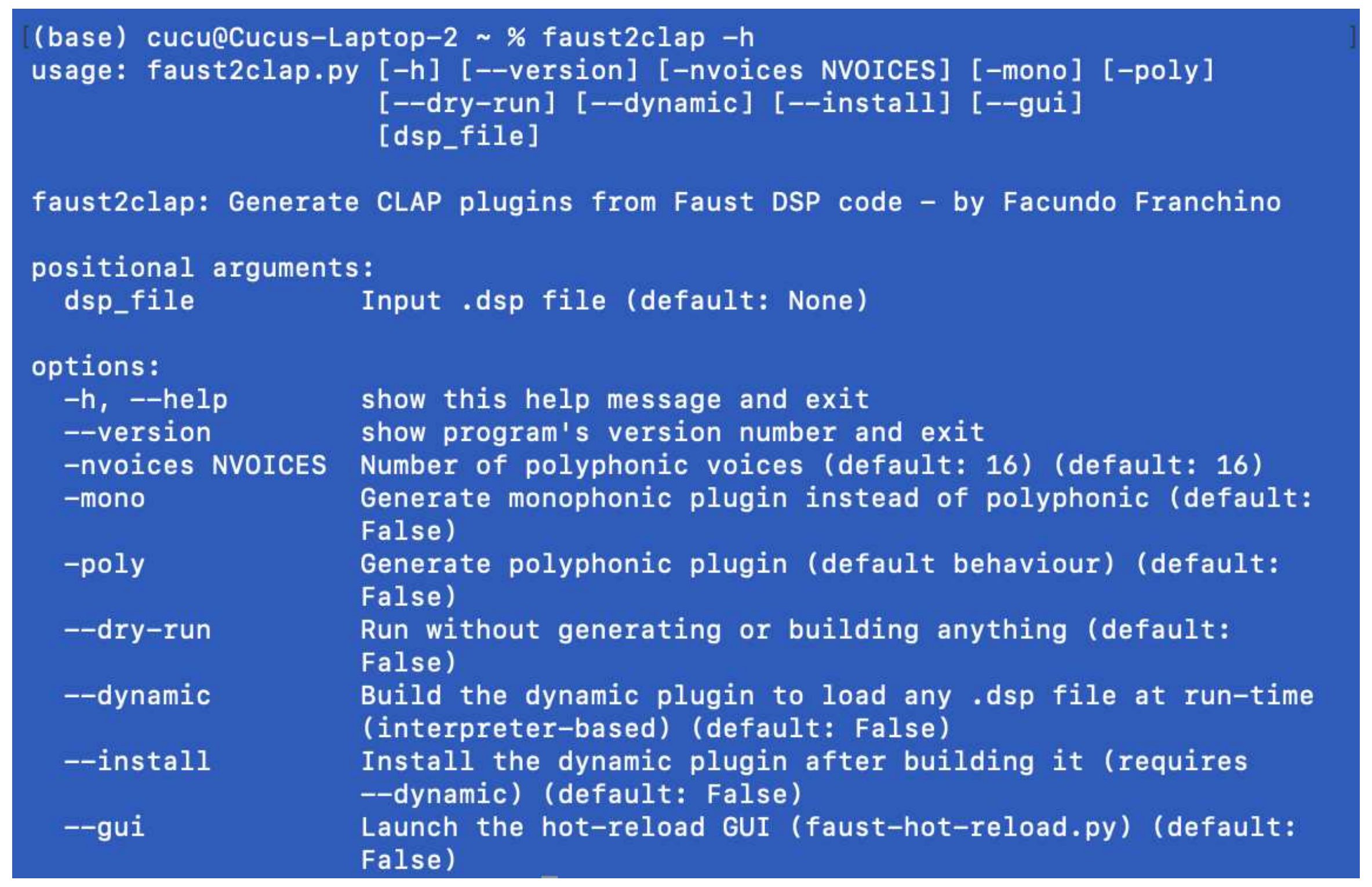
Traditional

faust2clap workflow

Figure 1: *Traditional C++ plug-in workflow vs. faust2clap workflow.*

System Overview

faust2clap is installed as a global CLI tool, allowing users to generate and manage CLAP plugins directly from Faust DSP code.

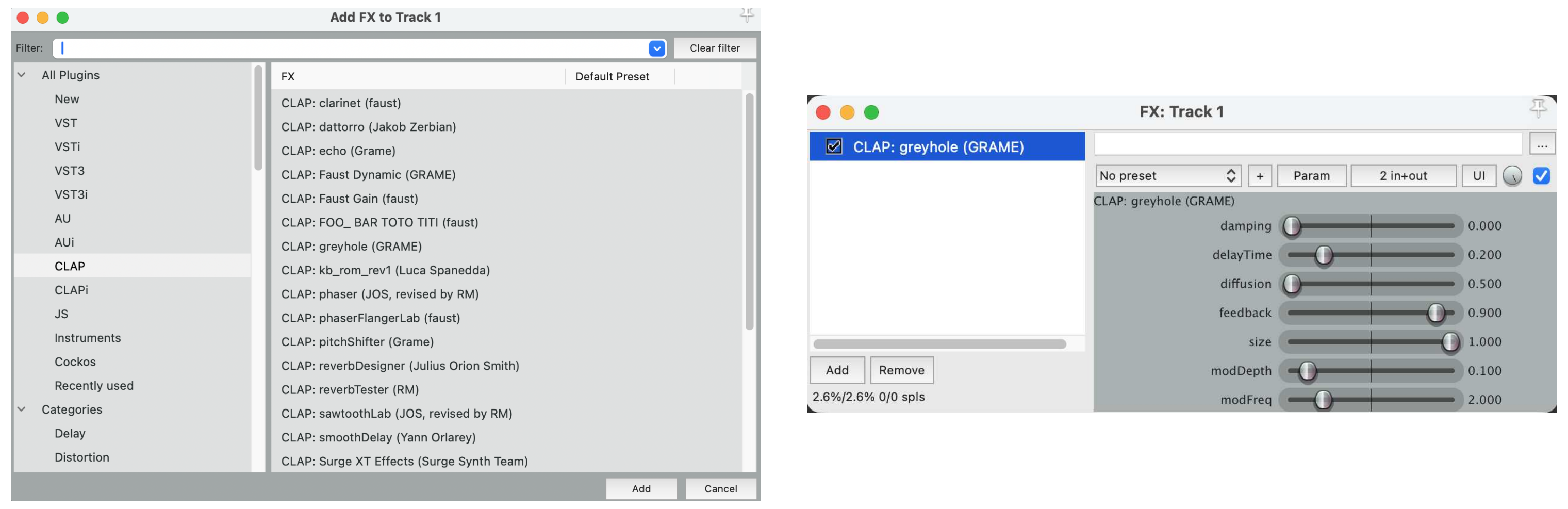


```
(base) cucu@Cucus-Laptop-2 ~ % faust2clap -h
usage: faust2clap.py [-h] [--version] [-nvoices NVOICES] [-mono] [-poly]
                     [--dry-run] [--dynamic] [--install] [--gui]
                     [dsp_file]

faust2clap: Generate CLAP plugins from Faust DSP code - by Facundo Franchino

positional arguments:
  dsp_file           Input .dsp file (default: None)

options:
  -h, --help         show this help message and exit
  --version          show program's version number and exit
  -nvoices NVOICES   Number of polyphonic voices (default: 16) (default: 16)
  -mono              Generate monophonic plugin instead of polyphonic (default:
                     False)
  -poly              Generate polyphonic plugin (default behaviour) (default:
                     False)
  --dry-run          Run without generating or building anything (default:
                     False)
  --dynamic          Build the dynamic plugin to load any .dsp file at run-time
                     (interpreter-based) (default: False)
  --install          Install the dynamic plugin after building it (requires
                     --dynamic) (default: False)
  --gui              Launch the hot-reload GUI (faust-hot-reload.py) (default:
                     False)
```

Figure 2: *faust2clap command–line interface (usage summary).*

Command Line Interface (CLI) Usage

Command	Description
`install-faust2clap.sh`	Install faust2clap globally
`faust2clap myeffect.dsp`	Build static CLAP plugin from DSP
`faust2clap --dynamic --install`	Install the dynamic plugin
`faust2clap --gui`	Launch GUI for hot-reloading DSPs
`faust2clap --help`	Show available options


Add FX to Track 1
Filter:
Clear filter
All Plugins
New
VST
VSTi
VST3
VST3i
AU
AUi
CLAP
CLAPi
JS
Instruments
Cockos
Recently used
Categories
Delay
Distortion
FX
Default Preset
CLAP: clarinet (faust)
CLAP: dattorro (Jakob Zerbian)
CLAP: echo (Grame)
CLAP: Faust Dynamic (GRAME)
CLAP: Faust Gain (faust)
CLAP: FOO_ BAR TOTO TITI (faust)
CLAP: greyhole (GRAME)
CLAP: kb_rom_rev1 (Luca Spanedda)
CLAP: phaser (JOS, revised by RM)
CLAP: phaserFlangerLab (faust)
CLAP: pitchShifter (Grame)
CLAP: reverbDesigner (Julius Orion Smith)
CLAP: reverbTester (RM)
CLAP: sawtoothLab (JOS, revised by RM)
CLAP: smoothDelay (Yann Orlarey)
CLAP: Surge XT Effects (Surge Synth Team)
Add
Cancel
FX: Track 1
☑ CLAP: greyhole (GRAME)
No preset
+
Param
2 in+out
UI
Add
Remove

2.6%/2.6% 0/0 spls

CLAP: greyhole (GRAME)

damping 0.000
delayTime 0.200
diffusion 0.500
feedback 0.900
size 1.000
modDepth 0.100
modFreq 2.000

Figure 3: *CLAP plug-ins generated by faust2clap — left: REAPER plug-in browser view, right: example plug-in instantiation.*

Architecture and Internals

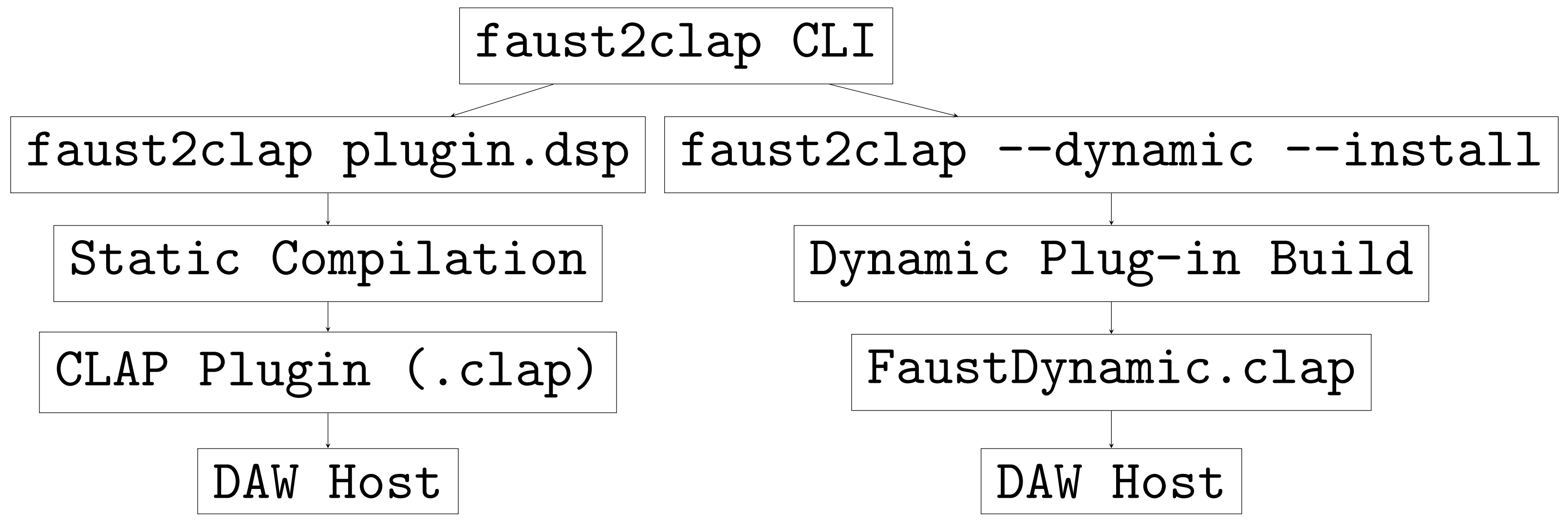


Figure 3: *Two ways of generating CLAP plug-ins from Faust code*

The Faust compiler translates `.dsp` source files into C++ classes, which are then wrapped by target-specific architecture files (handling audio I/O and UI) and deployed using `faust2xx` scripts as plug-ins or standalone programs.

But how does **faust2clap** enable *dynamic* plug-in compilation within a DAW, in just milliseconds?

This is possible thanks to *libfaust*, an embeddable version of the Faust compiler. It supports an LLVM IR backend and several others, including the **interpreter backend**, which powers dynamic support in `faust2clap`.

How the Interpreter Backend Works:

- The Faust compilation pipeline consists of the following stages:
  1. `.dsp` → Block Diagram
  2. → Signal Graph
  3. → Faust Imperative Representation
  4. → Typed Bytecode (via the interpreter backend)
- This bytecode is executed by a Virtual Machine (VM) embedded inside the plug-in.
- Dynamic compilation is initiated via `createInterpreterDSPFactory()`:
  - Accepts Faust code (file or string)
  - Returns an `interpreter dsp factory` containing the bytecode
  - `createDSPInstance()` then generates a DSP object with the embedded VM

Performance Notes

- Interpreter-based plug-ins are typically 3–10× slower than those compiled via LLVM.
- However, they allow on-the-fly editing and hot-reload, ideal for prototyping or live coding.

Conclusions

This project presents a novel contribution to the Faust and CLAP ecosystems: a full–featured toolchain for generating both static and dynamically reloaded CLAP plug–ins from Faust DSP code,all without leaving the DAW environment.

It shortens the path from idea to implementation. With hot-reload support, developers can test DSP changes live. With static compilation, they can produce high-performance binaries. One tool, one codebase, zero boilerplate.

Future Work:

- Add a native JUCE–based GUI inside the plug–in.
- On–screen DSP file selector and preset switching.
- Integrate CPU usage benchmarking across backends.
- Variable parameter system instead of a fixed 12 parameter one

Acknowledgements:

- Google Summer of Code, for supporting this project through the 2025 programme.
- Stéphane Letz, for the opportunity to contribute to Faust and his continuous guidance throughout the project.
- Jatin Chowdhury, for mentorship and deep insights into the CLAP standard.
- The CLAP and Faust communities, for invaluable feedback, discussion, and support.

GitHub