GRCON26 WORKSHOP

Intro to GNU Radio 4

Josh Morman

Chief Scientist, Altio Labs
joshua.morman@altiolabs.com

President, GNU Radio
jmorman@gnuradio.org

`https://github.com/altiolabs/grcon26-intro-to-gnuradio4`

In the next hour

Understand → **Build** → **Run** → **Deploy** → **Extend**

Take some basic blocks and graphs

Explore Several ways to work with them in GR4

INTRODUCTION

Why do I care about GR4

GNU Radio, re-engineered

If you know GNU Radio, you already know the basic idea of GR4

Blocks · ports · streams · settings · tags · messages

A flowgraph composes signal processing applications

What changed underneath?

Modern C++ first

Typed blocks and ports

Runtime redesigned

Graphs, scheduling, execution

Tooling redesigned

Reflection and Studio

Deployment choices

Applications, embedded, remote UI

Why rewrite the machinery underneath?

Typing

Catch mismatches

Reflection

Build better tools

Composition

Fit the application

Modular scheduling

Choose execution policy

Efficient execution

Bulk work + SIMD paths

Allow usage of GNU Radio in places it was not possible before

Getting GR4: build the workspace

```
git clone https://github.com/gnuradio/gnuradio4.git
cd gnuradio4
cmake --preset full
cmake --build --preset full
source build/full/activate.sh
gr4-studio
```

Prepare before the workshop: C++23 (GCC 14+, Clang 20+) · CMake 3.27+ · Ninja · Node 22+

Another route: containerized GR4

```
docker pull ghcr.io/gnuradio/gnuradio4-studio:latest
docker run --rm -p 8080:8080 \
  ghcr.io/gnuradio/gnuradio4-studio:latest
```

Open **localhost:8080**

podman instead of docker works just as well

The Repos

`github.com/gnuradio/gnuradio4-*`

gnuradio4

top level, entrypoint

core

runtime framework APIs

library

non-block things

blocks

the blocks.

control-plane

plugin based REST host

studio

interactive UI

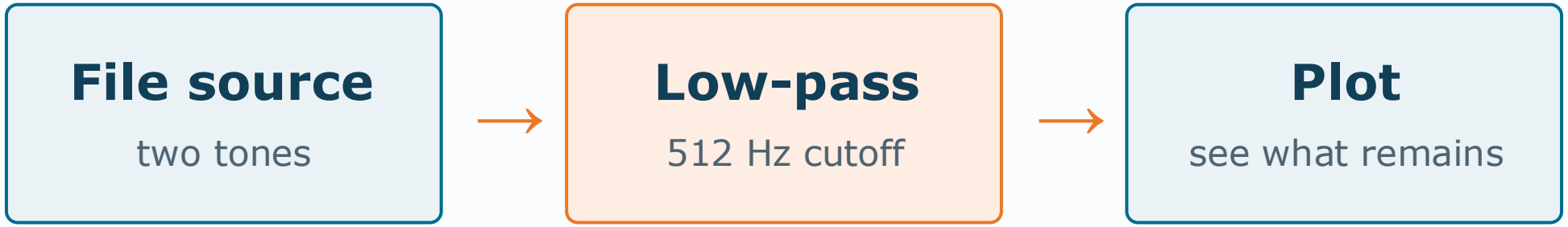
LICENSING PHILOSOPHY

Enable free and open source software radio for as many people as possible

Permissive core and standard blocks reduces friction

Copyleft derived blocks and applications encouraged

One graph for the rest of the workshop



128 Hz + 2048 Hz · 8192 samples/s

Same blocks.
Different ways to assemble them.

01

Static C++

Compile flowgraph and blocks

02

Runtime / plugins

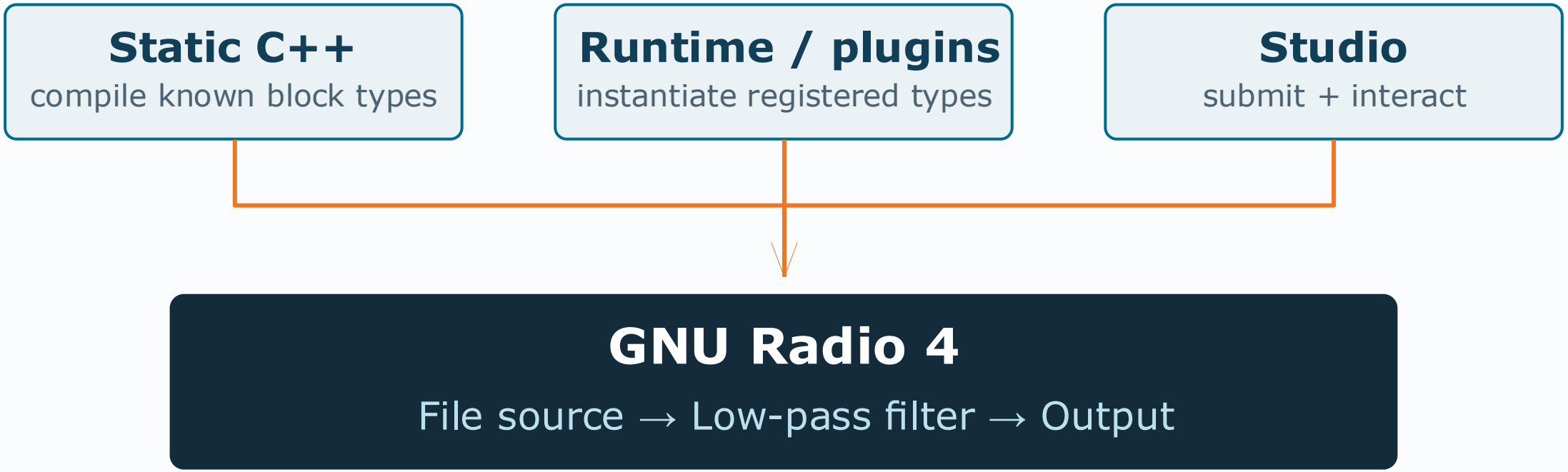
Create registered block types

03

Studio

Build visually.
Run and interact.

The important part



The signal-processing model does not change.

GRC renders code for your application
Studio operates interactively

GRC

Flowgraph design

↓

Generated Python / C++

↓

Application

Studio

Graph snapshot → session

↕

GR4 runtime - Control Plane [REST]

↕

Live settings + plots

The UI does not have to live beside the DSP

Desktop

Electron UI

Local gr4cp_server

Browser

WASM

Running entirely in browser

Remote

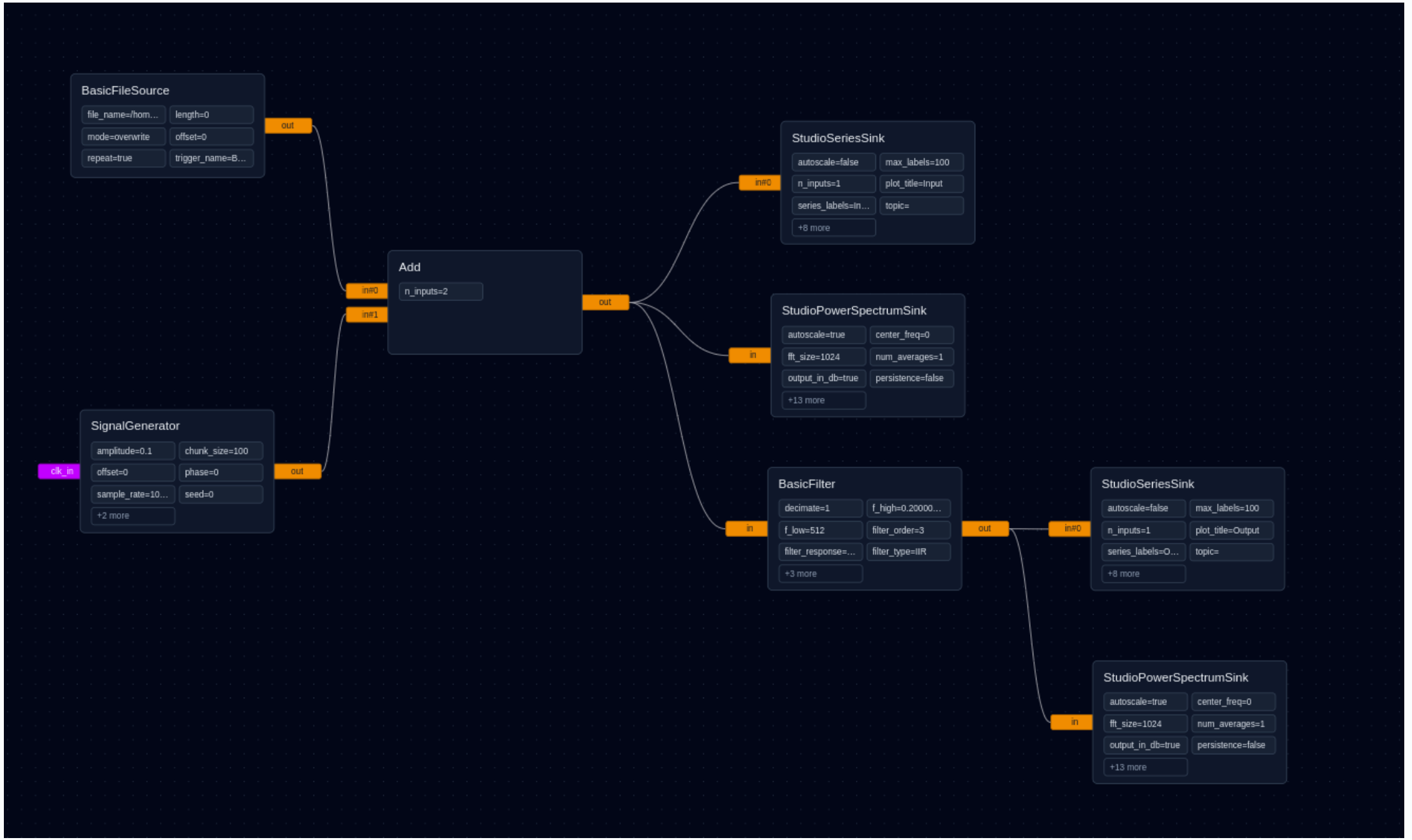
UI on your laptop

DSP on another host (e.g. embedded radio)

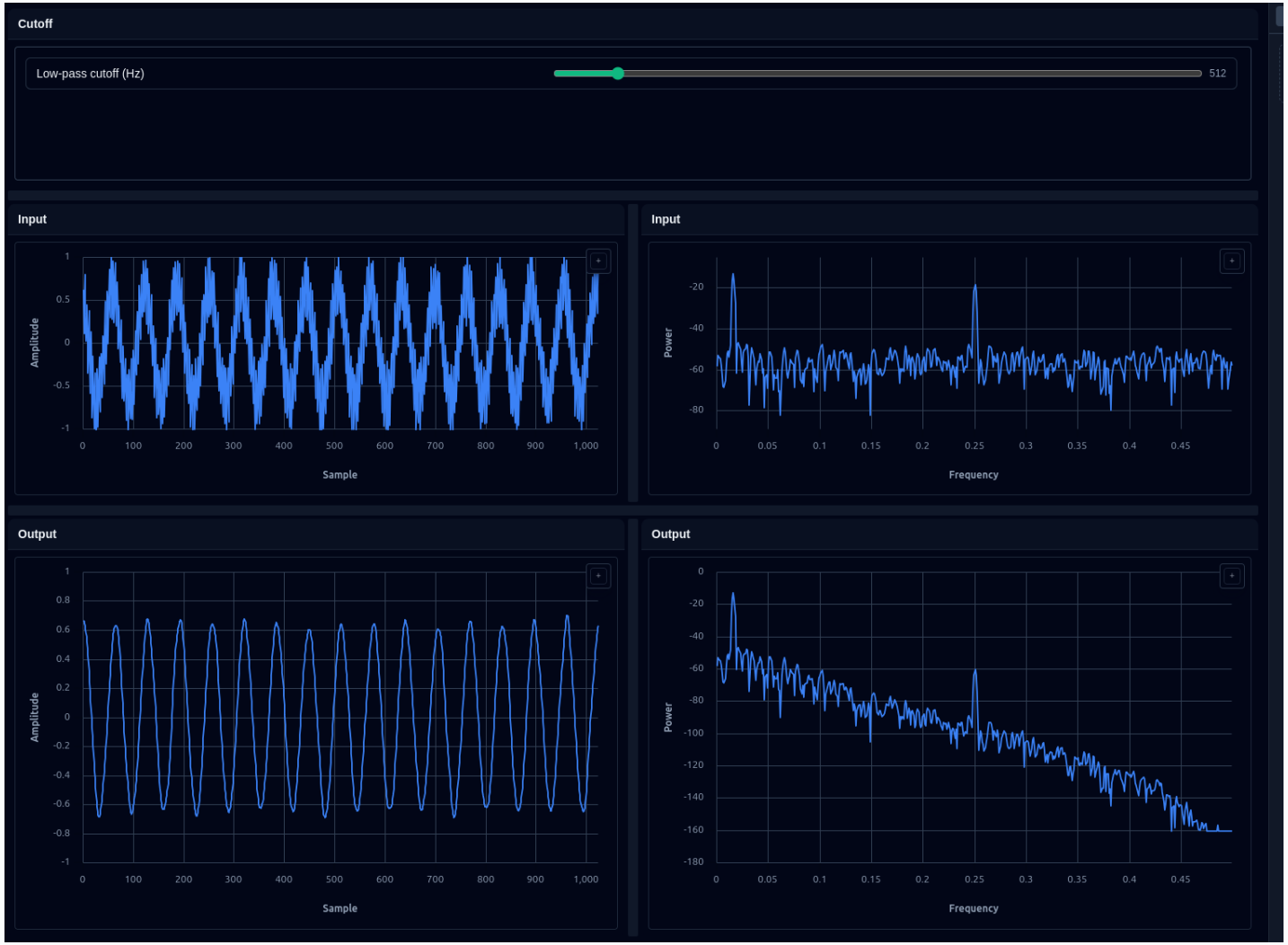
LIVE EXAMPLES

Intro Tutorial

https://github.com/altiolabs/grcon26-intro-to-gr4


BasicFileSource
file_name=/hom...
length=0
mode=overwrite
offset=0
repeat=true
trigger_name=B...
out
SignalGenerator
amplitude=0.1
chunk_size=100
clk_in
offset=0
phase=0
out
sample_rate=10...
seed=0
+2 more
Add
n_inputs=2
in#0
in#1
out
StudioSeriesSink
autoscale=false
max_labels=100
in#0
n_inputs=1
plot_title=Input
series_labels=In...
topic=
+8 more
StudioPowerSpectrumSink
autoscale=true
center_freq=0
in
fft_size=1024
num_averages=1
output_in_db=true
persistence=false
+13 more
BasicFilter
decimate=1
f_high=0.20000...
in
f_low=512
filter_order=3
out
filter_response=...
filter_type=IIR
+3 more
StudioSeriesSink
autoscale=false
max_labels=100
in#0
n_inputs=1
plot_title=Output
series_labels=O...
topic=
+8 more
StudioPowerSpectrumSink
autoscale=true
center_freq=0
in
fft_size=1024
num_averages=1
output_in_db=true
persistence=false
+13 more

What to notice



Now build the same graph without Studio

```
gr::Graph graph;
auto& source = graph.emplaceBlock<fileio::BasicFileSource<float>>(
    {{"file_name", input}});
auto& lowpass = graph.emplaceBlock<filter::BasicFilter<float>>(
    {{"sample_rate", 8192.0f}, {"f_low", 512.0f}});
auto& sink = graph.emplaceBlock<fileio::BasicFileSink<float>>(
    {{"file_name", output}});
checked(graph.connect<"out", "in">(source, lowpass));
checked(graph.connect<"out", "in">(lowpass, sink));
gr::scheduler::Simple scheduler;
checked(scheduler.exchange(std::move(graph)));
checked(scheduler.runAndWait());
```

Compile-tested excerpt · `checked` throws on errors · file output for comparison

What did we gain?

C++ graph → **Application** → **Deploy**

Explicit structure

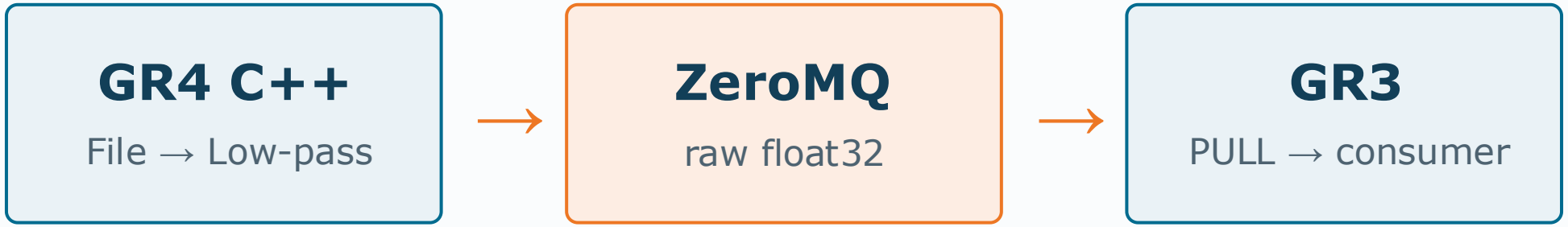
Known graph.
Repeatable processing.

Fits your system

Headless execution.
Part of a larger application.

Ship the required runtime libraries and data with your application.

GR4 does not have to own the whole application



8192 samples. Identical output.

Optional live demo · current ZeroMQ headers required

Data is not only scalar samples

Sample streams

float
complex<float>

Structured data

DataSet<T>
Tensor<T>

Packet-like items

Packet<T>
payload + metadata

The connected blocks must support the type.

The ecosystem is still the superpower

GNU Radio 4

Community

Reusable blocks
Shared expertise

Research

New algorithms
New instruments

Products

Public or private
OOT modules

Core + standard blocks: MIT. Studio application: GPL-3.0-or-later.

A GR4 block can be surprisingly small

```
struct Gain : gr::Block<Gain> {
    using gr::Block<Gain>::Block;
    using Description = gr::Doc<"Multiply each input sample by two.">;
    gr::PortIn<float> in;
    gr::PortOut<float> out;
    float factor = 2.0f;
    GR_MAKE_REFLECTABLE(Gain, in, out);
    [[nodiscard]] constexpr float processOne(float input) const noexcept {
        return input * factor;
    }
};
```

Next: **GR4 Block Development** · the full OOT workflow

Reflection changes the tooling model

Block definition

Ports + sample types
Reflected settings
Descriptions + constraints

GR4 reflection

↓

Block catalog + inspector

↓

Studio and other tools

Describe the block once.

Expose it to more than one tool.

What to remember

01 The flowgraph model is the same.

02 The runtime was re-engineered.

03 Compose in C++ or at runtime.

04 Studio interacts with a running system.

05 Start building and help shape GR4.

Try it. Build on it.
Help us improve it.

GNU Radio 4 source

Source + build instructions

GR4 documentation

→ Need help!!

GR4 Studio

UI + visualization blocks

GR4 technical discussion

Community on Matrix

Questions?

Next workshop: GNU Radio 4 Block Development