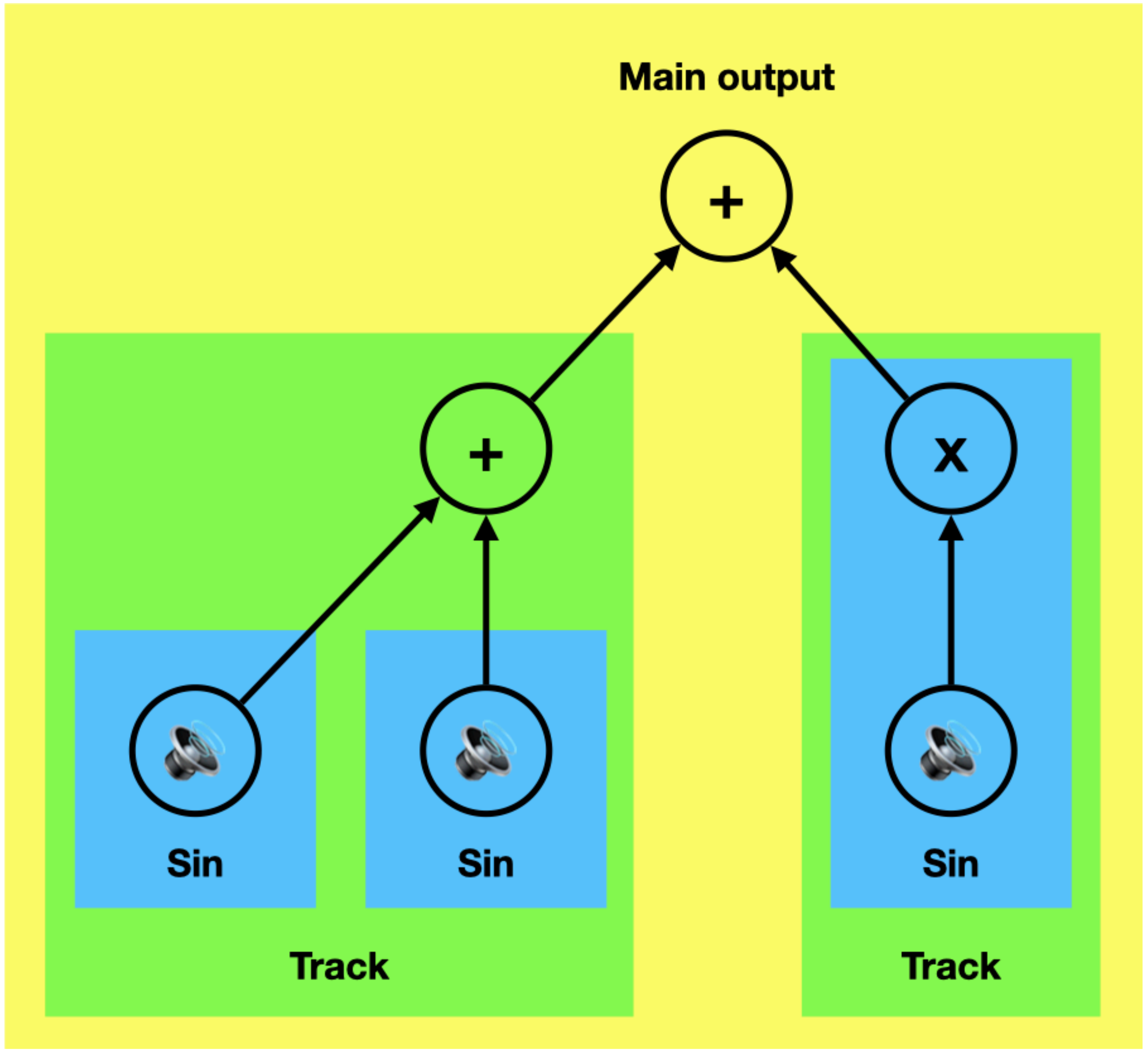
```cpp
class SinNode final : public Node
{
public:
    SinNode (float frequency, int numChannelsToUse)
        : numChannels (numChannelsToUse)
    {
        osc.setFrequency (frequency, true);
    }

    NodeProperties getNodeProperties() override
    {
        NodeProperties props;
        props.hasAudio = true;
        props.hasMidi = false;
        props.numberOfChannels = numChannels;

        return props;
    }

    bool isReadyToProcess() override      { return true; }

    void prepareToPlay (const PlaybackInitialisationInfo& info) override
    {
        osc.prepare ({ double (info.sampleRate), uint32_t (info.blockSize), (uint32_t) numChannels });
    }

    void process (const ProcessContext& pc) override
    {
        auto block = pc.buffers.audio;
        osc.process (juce::dsp::ProcessContextReplacing<float> { block });
    }

private:
    juce::dsp::Oscillator<float> osc { [] (float in) { return std::sin (in); } };
    const int numChannels;
};
```

```cpp
class SinNode final : public Node
{
public:
    SinNode (float frequency, int numChannelsToUse)
        : numChannels (numChannelsToUse)
    {
        osc.setFrequency (frequency, true);
    }

    NodeProperties getNodeProperties() override
    {
        NodeProperties props;
        props.hasAudio = true;
        props.hasMidi = false;
        props.numberOfChannels = numChannels;

        return props;
    }

    bool isReadyToProcess() override      { return true; }

    void prepareToPlay (const PlaybackInitialisationInfo& info) override
    {
        osc.prepare ({ double (info.sampleRate), uint32_t (info.blockSize), (uint32_t) numChannels });
    }

    void process (const ProcessContext& pc) override
    {
        auto block = pc.buffers.audio;
        osc.process (juce::dsp::ProcessContextReplacing<float> { block });
    }

private:
    juce::dsp::Oscillator<float> osc { [] (float in) { return std::sin (in); } };
    const int numChannels;
};
```


Main output
+
+
x
Sin
Sin
Sin
Track
Track