



UI Feedback optimizations in Lens



How to have efficient, large-scale feedback in a Dplug plug-in.



Meeting
Sep 27th 2022

Lens plug-in = more feedback than typical for us




GAIN MAP

Display energy estimate, and gain reduction for compressor and expander in gain map.

EQ

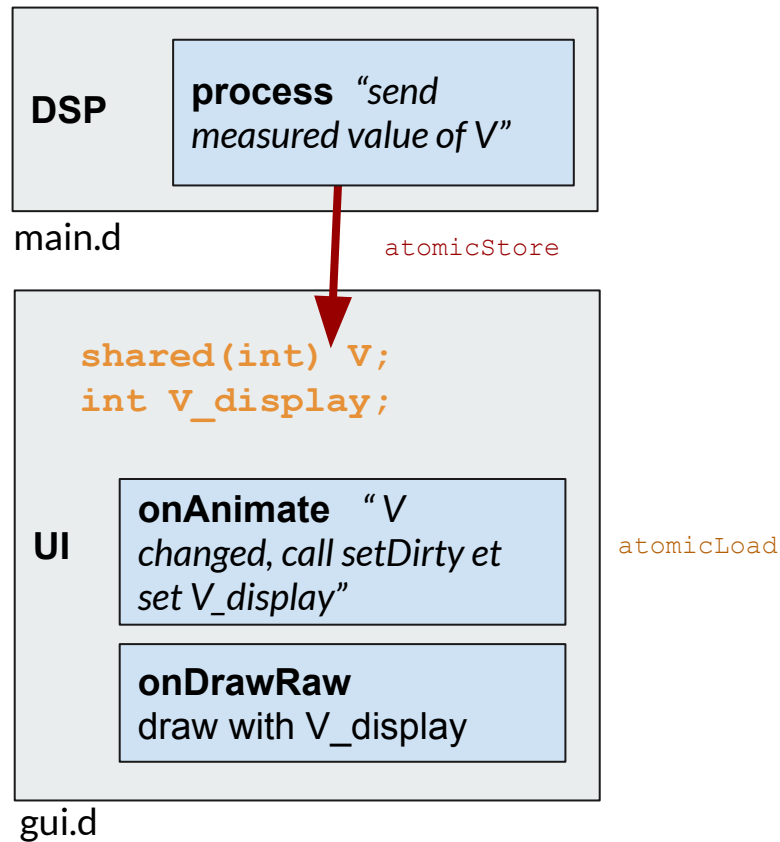
Each EQ has spectrogram, one of those display also gain reduction from compressor.



What are the desirable properties of DSP to UI feedback?

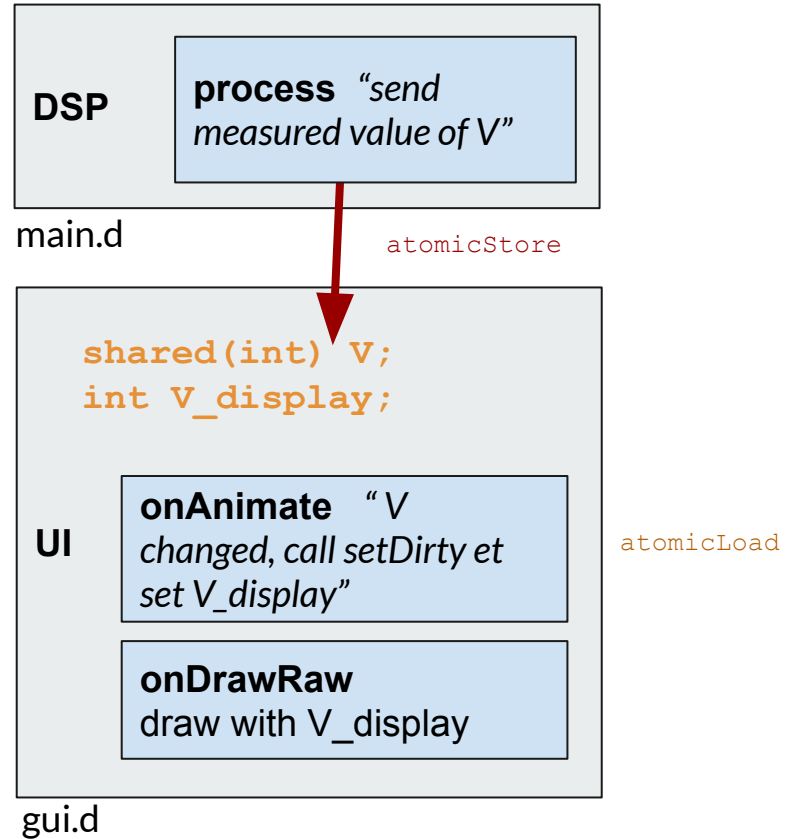
- **Fast.** It should be cheap enough to be activated at all time. (*Thankfully `dplug:canvas` is fast*)
- **Large.** *People want many things to be feedback visually.*
- **Non blocking.** *It's not worth holding back the audio if the UI struggles.*
- **Decoupled from DSP buffer size.**
- **Decoupled from DSP sampling rate.**
- **Sync.** Visual should approximately correspond to audio temporally.

Why not just use `core.atomic`?



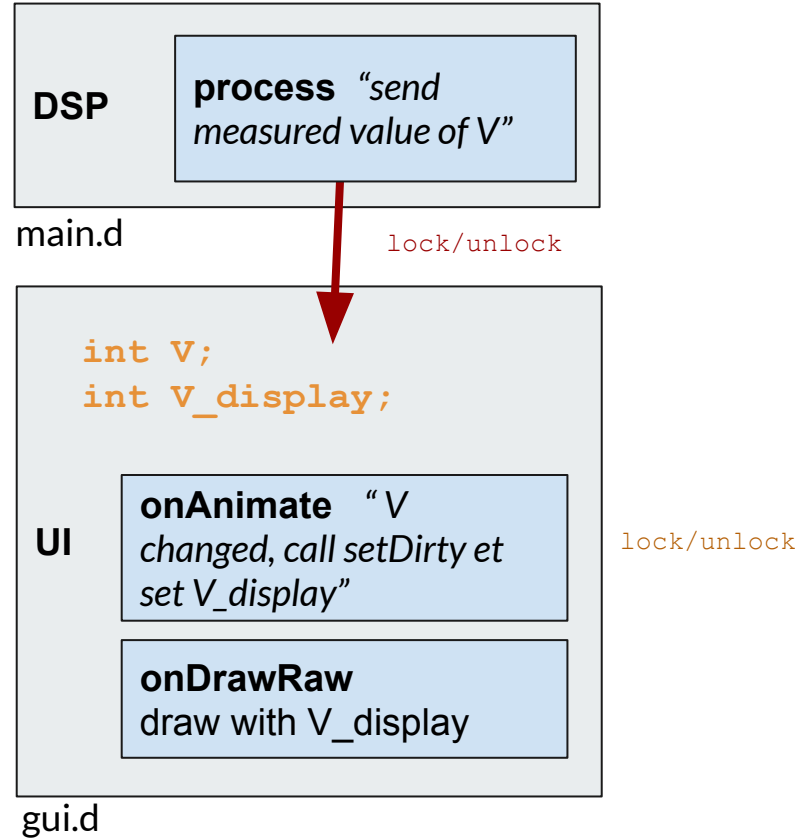
Why not just use `core.atomic`?

- Fast. ✓
- Large. ✗ *(it's just one scalar)*
- Non blocking. ✓
- Decoupled from DSP buffer size. ✗
- Decoupled from DSP sampling rate. ✗
- Sync. ✗



Why not just use *a mutex?*

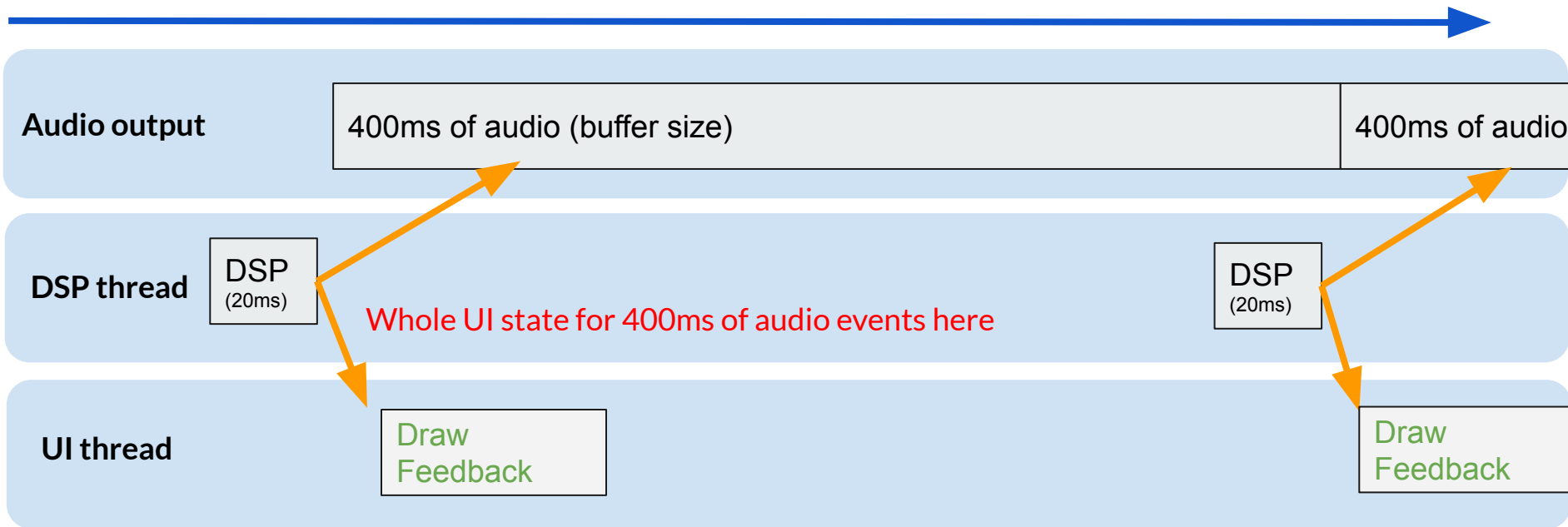
- Fast. ✓
- Large. ✓
- Non blocking. ✗
- Decoupled from DSP buffer size. ✗
- Decoupled from DSP sampling rate. ✗
- Sync. ✗



What's the problem with (large) buffer size?

Say we have 400ms buffer size, and the plugin is 20x realtime. Takes 20ms to process audio.

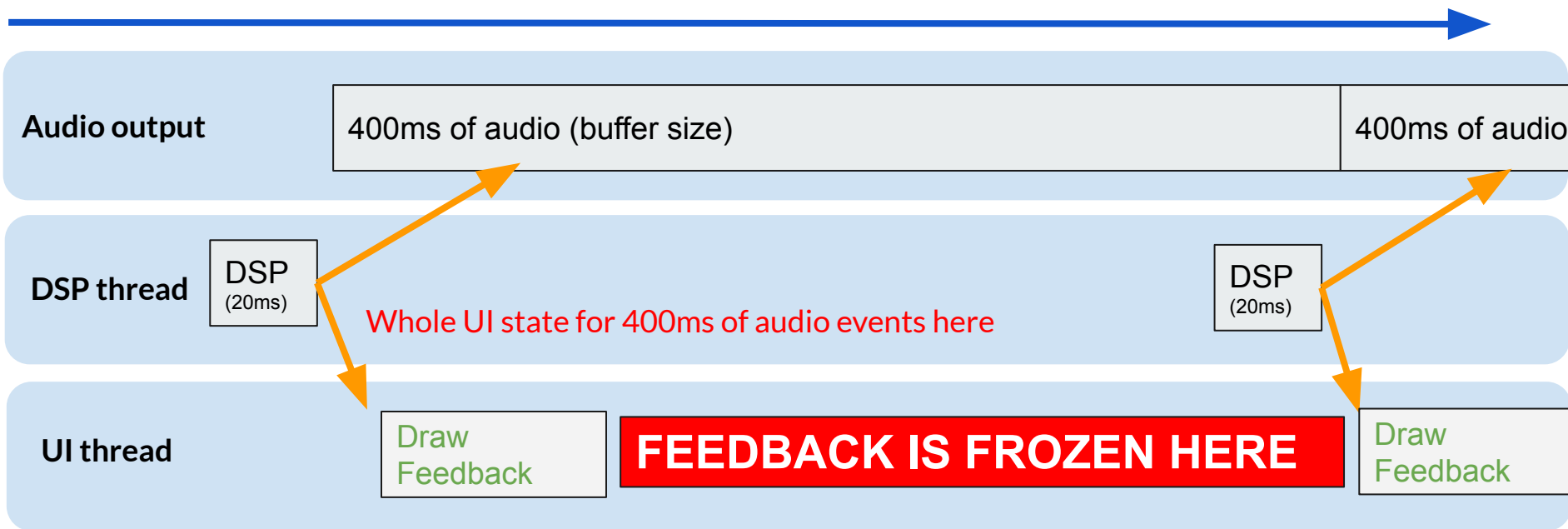
time



What's the problem with (large) buffer size?

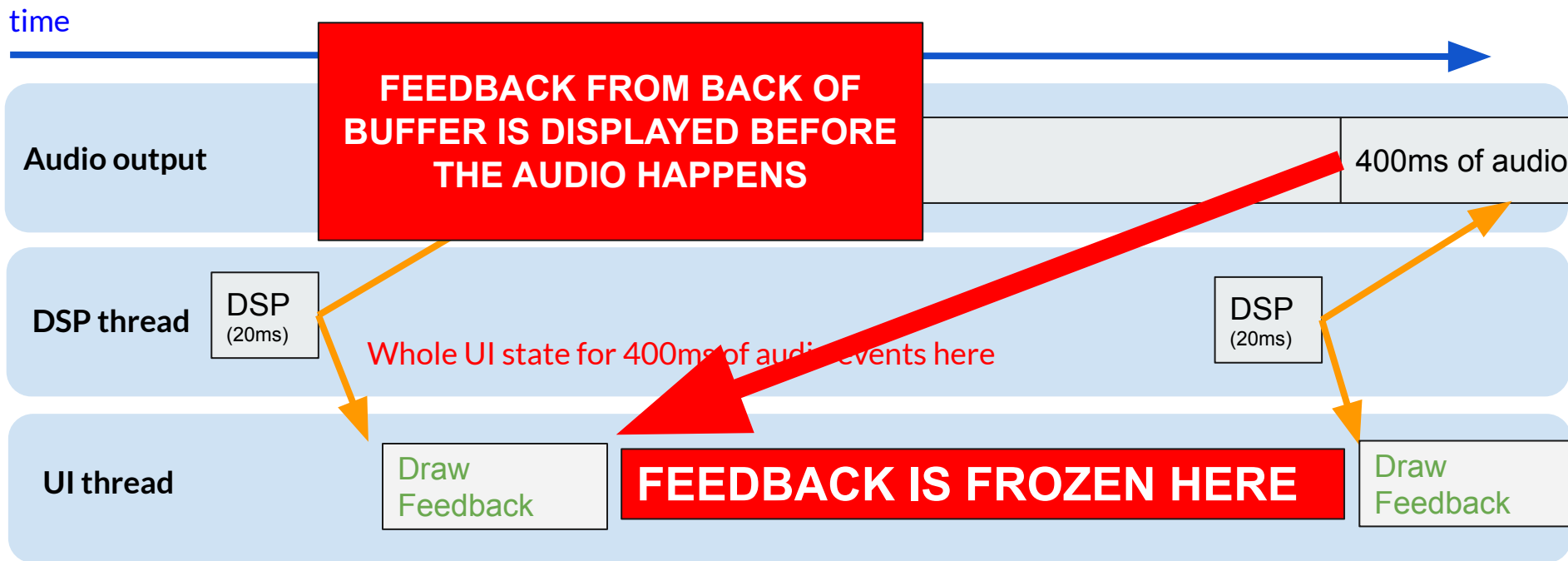
Say we have 400ms buffer size, and the plugin is 20x realtime. Takes 20ms to process audio.

time



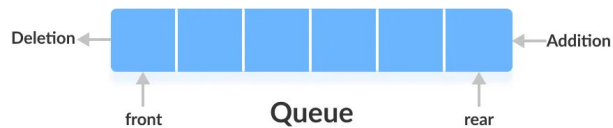
What's the problem with (large) buffer size?

Say we have 400ms buffer size, and the plugin is 20x realtime. Takes 20ms to process audio.





You need a queue somewhere.



You need a queue somewhere.

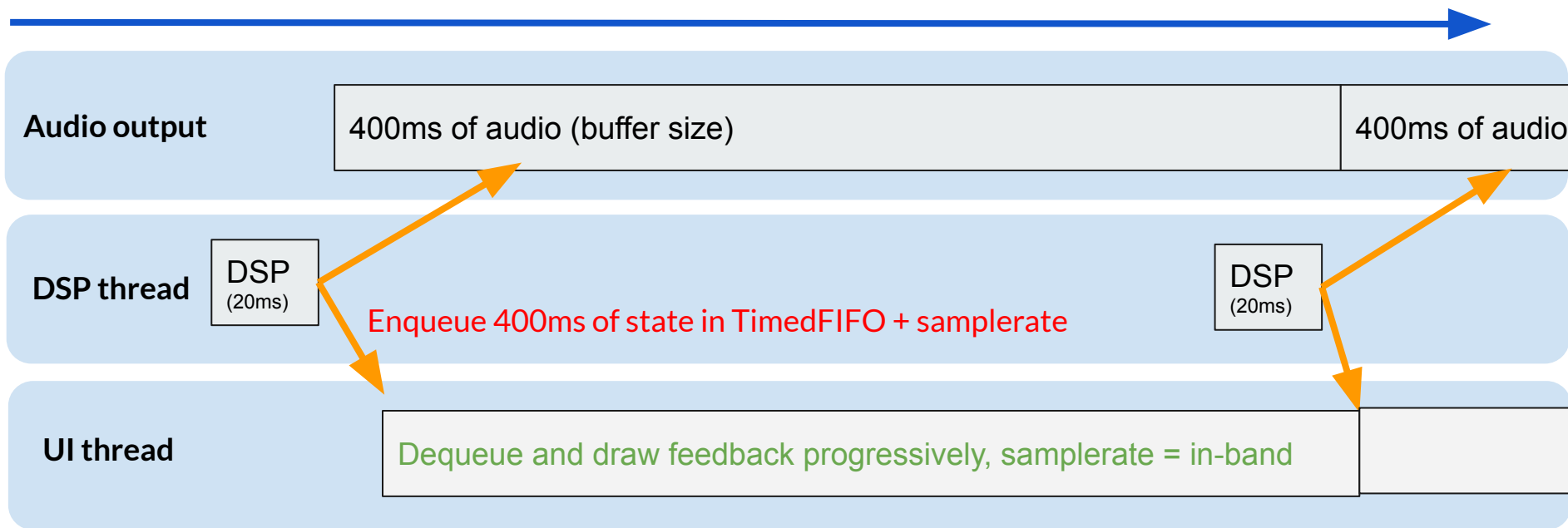
onAnimate should read the queue progressively,
and *at the right speed*.



That's where **TimedFIFO** originated (2016).

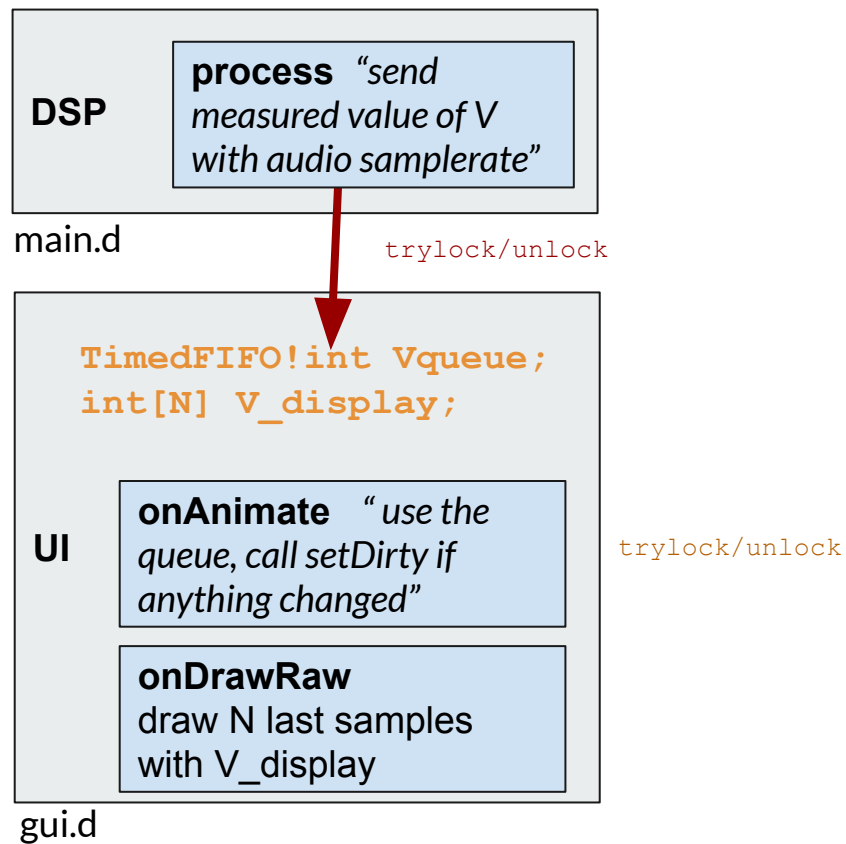
Using `dplug.core.ringbuf.TimedFIFO`

time



TimedFIFO

- Fast. ✓
- Large. ✓
- Non blocking => *almost*
- Decoupled from buffer size => *almost*
- Decoupled from sampling rate => ...no! ✗
- Sync. ✓



TimedFIFO problems

- **Samplerate.** Optimal queue size depends on sample rate, which changes everything again. At 96000 Khz, queue is emptied faster. But no **knowledge of sampling rate at queue startup.**
- **(typically) Buffer size.** Even if you gather feedback every 32 samples, if your buffer size is not multiple of 32, you will forget some.
- **Non-blocking.** One trylock is kinda ok for the whole feedback, but the problem is that typical plugin have *several* such TimedFIFO.

This is where we stopped for earlier Auburn Sounds plugins.

LENS compressor = more feedback than typical for us



Compressor Input volume (64 bands)
Expander Input volume (64 bands)
Compressor Gain reduction (64-bands)
Expander Gain reduction (64-bands)
Spectral volume for sidechain (64-bands)
Spectral volume for wet signal (64-bands)
etc...

The whole unit of feedback:

519 scalar values transit from DSP to UI,
measured 40x per second.

Feedback Tip #1: single FeedbackData struct

1. Have one single **FeedbackData** struct for all plugin feedback.
2. Have one single **TimedFIFO** for that
3. Profit from less synchronization

```
1
2 // Define the Feedback struct used by Lens.
3 // Lens has a particularly high amount of feedback, and doing it once per buffer
4 // and widget it not good.
5 // Hence it is regrouped in a single TimedFIFO.
6 // Feedback, unlike other plugins, is regrouped at given time intervals regardless of
7 // sampling-rate.
8 struct FeedbackData
9 {
10     // Last known sample rate.
11     float sampleRate;
12
13     // Last known number of bins.
14     int numBins;
15
16     //min-rate and max-rate
17     float minRateHz;
18     float maxRateHz;
19
20     bool listenMode; // Listen to sidechain
21     bool relativeMode; // Relative mode
22
23     // So that we don't need to dirty things if the generation is inferior.
24     long generation;
25
26     // ***** OTHER FEEDBACK ARRAYS *****
27
28     float[64] inputExpand_linear; // Volume of expander input, possibly normalized (relative).
29     float[64] outputExpandGR_linear;
30 }
31
```



Feedback Tip #2: compute feedback $40x / \text{sec}$,
and for a single sample.

in `processAudio` callback

```
1
2
3 // Should we collect feedback in this callback?
4 // and in which sample?
5 // If collectFeedback is true, this will be recomputed.
6 bool collectFeedback = false;
7 int feedbackSamplePos = 0;
8 if (_feedbackCounter == -1) //initialization
9 {
10     _feedbackCounter = 0;
11     collectFeedback = true;
12 }
13 _feedbackCounter += frames;
14 if (_feedbackCounter >= _collectFeedbackEverySamples)
15 {
16     collectFeedback = true;
17     _feedbackCounter = _feedbackCounter % _collectFeedbackEverySamples;
18     feedbackSamplePos = frames - 1 - _feedbackCounter;
19     assert(_feedbackCounter >= 0 && _feedbackCounter < frames);
20     assert(feedbackSamplePos >= 0 && feedbackSamplePos < frames);
21 }
22
23 // with _collectFeedbackEverySamples = cast(int)(sampleRate / FEEDBACK_DSP_HZ + 0.5f)
24
25
26
```

in `processAudio` callback

```
1 // Should we collect feedback in this callback?  
2 // and in which sample?  
3 // If collectFeedback is true, this will be recomputed.
```

```
4 bool collectFeedback = false;
```

```
5 int feedback;
```

```
6 if (_feedback
```

```
7 {
```

```
8     _feedback
```

```
9     collect
```

```
10 }
```

```
11 _feedbackC
```

```
12 if (_feedback
```

```
13 {
```

```
14     collect
```

```
15     _feedback
```

```
16     feedback
```

```
17     assert(
```

```
18     assert(
```

```
19 }
```

```
20 }
```

```
21 // with _collectFeedbackEverySamples = cast(int)(sampleRate / FEEDBACK_DSP_HZ + 0.5f)
```

```
22
```

```
23
```

```
24
```

```
25
```

```
26
```

ONLY COMPUTE THE FEEDBACK STRUCT IF
collectFeedback == true

AND THEN, ONLY FOR ONE SAMPLE
IN THE WHOLE SUB-BUFFER.


PASS THAT INFO IN ALL DSP THAT HAS
FEEDBACK.

in `processAudio` callback


```
if (collectFeedback)
{
    _feedbackData.sampleRate = _sampleRate;
    _feedbackData.listenMode = listenSidechain;
    _feedbackData.relativeMode = expandRelative;

    // GUI feedback
    if ( auto gui = cast(LensGUI) graphicsAcquire() )
    {
        gui.sendFeedback(_feedbackData);
        graphicsRelease();
    }
}
```


In lens, max subbuffer size is 512 samples,
and feedback period is 1102 samples at 44100Hz.
Feedback will only break down at 11025Hz.



```
void sendFeedback(FeedbackData data)
{
    data.generation = _writeFeedbackGeneration++;
    _timedFifo.pushData(data, FEEDBACK_DSP_HZ);
}
```



Instead of pushing the audio samplerate in-band, give the FIFO the *feedback sampling rate*. (here = 40Hz) FIFO created with 12 slots, corresponding to $12 * 1000/40$ ms of feedback independently of the audio sampling rate. 🎉



Annoying, but worth it.

- Fast. ✓
- Large. ✓
- Non blocking: *almost* ✓
- Decoupled from buffer size. ✓
- Decoupled from sampling rate. ✓
- Sync. ✓

Feedback Tip #3: accumulate delta time when `onAnimate` is called with small `dt`

```
override void onAnimate(double dt, double time)
{
    _rateLimitDt += dt;
    if (_rateLimitDt > minimumAnimationDelta)
    {
        bool dirty = rateLimitedAnimation(_rateLimitDt);
        if (dirty)
            setDirtyWhole();
        _rateLimitDt = 0;
    }
}
```


Feedback Tip #3: accumulate delta time when `onAnimate` is called with small `dt`

```
override void onAnimate(double dt, double time)
{
    _rateLimitDt += dt;
    if (_rateLimitDt > minimumAnimationDelta)
    {
        bool dirty = rateLimiteAnimation(_rateLimitDt);
        if (dirty)
            setDirtyWhole();
        _rateLimitDt = 0;
    }
}
```

100 ms

Save CPU
by avoiding
some redraw.

Basically = not worth it to redraw for too small a change.

Feedback Tip #4: Fix your timestep when needed.

- onAnimate is called repeatedly, but with any variable delta time (**dt**).
- Like in video games, this can be tricky for animation, especially **if you want points with trails**.
- But you can manually fix your timestep for some widgets.

can be small
or very large

```
override void onAnimate(double dt, double time)
{
```

Fixed animation rate howto

```
1  
2  
3  override void onAnimate(double dt, double time)  
4  {  
5      // Sub animation, with fixed frame-rate.  
6      _accumulatedDt += dt;  
7  
8      float decayAlpha = 1.0 - expDecayFactor(rmsDecayTime, 1.0 / animationStep);  
9  
10     while(_accumulatedDt > animationStep)  
11     {  
12         _accumulatedDt -= animationStep;  
13         if (animationFrame(decayAlpha))  
14             dirty = true;  
15     }  
16  
17     if (dirty == true)  
18         setDirtyWhole();  
19 }  
20  
21  
22
```

100ms

animationFrame called 10x per second

eventually hoist
computation out of the
fixed animation loop
(unsure gain here)



Feedback Tip #5: Drawing performance.

Same old advice.

- **Use `dplug:canvas`**, it write 4 pixels at once.
- **Do not update PBR layer for animation**, except for small widgets.
- *(advanced)* You can dirty only the graphics subpart of the widget that you know will be affected.
- Things will draw faster if update area rectangle is small and constrained. But, hard to do.



Questions?

Thanks for listening!