

News about threadshare

a plugin collection to increase scalability

GStreamer Conference 2025

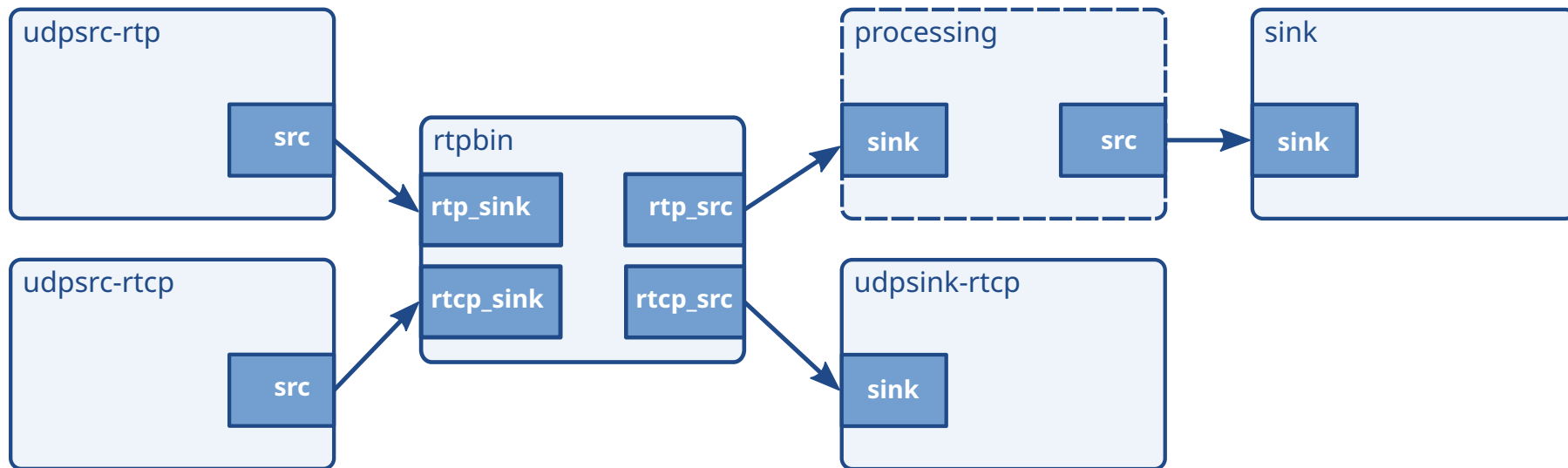
Who am I?

- François — *fengalin*
- Contributor to GStreamer since 2017, mostly to `gst-rs` & `gst-plugins-rs`
- Work at Centricular — *francois at centricular dot com*

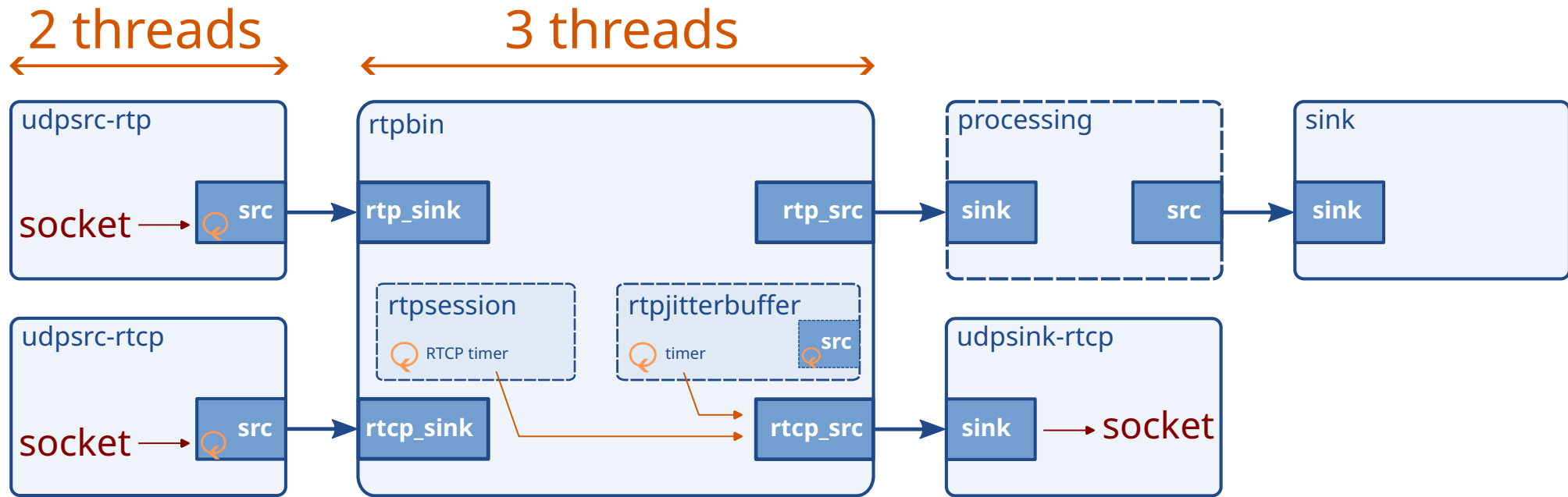
When adding more threads adds more problems - Sebastian @ Gst Conf 2018

Context: hundreds of live streams

Example using an RTP receiver

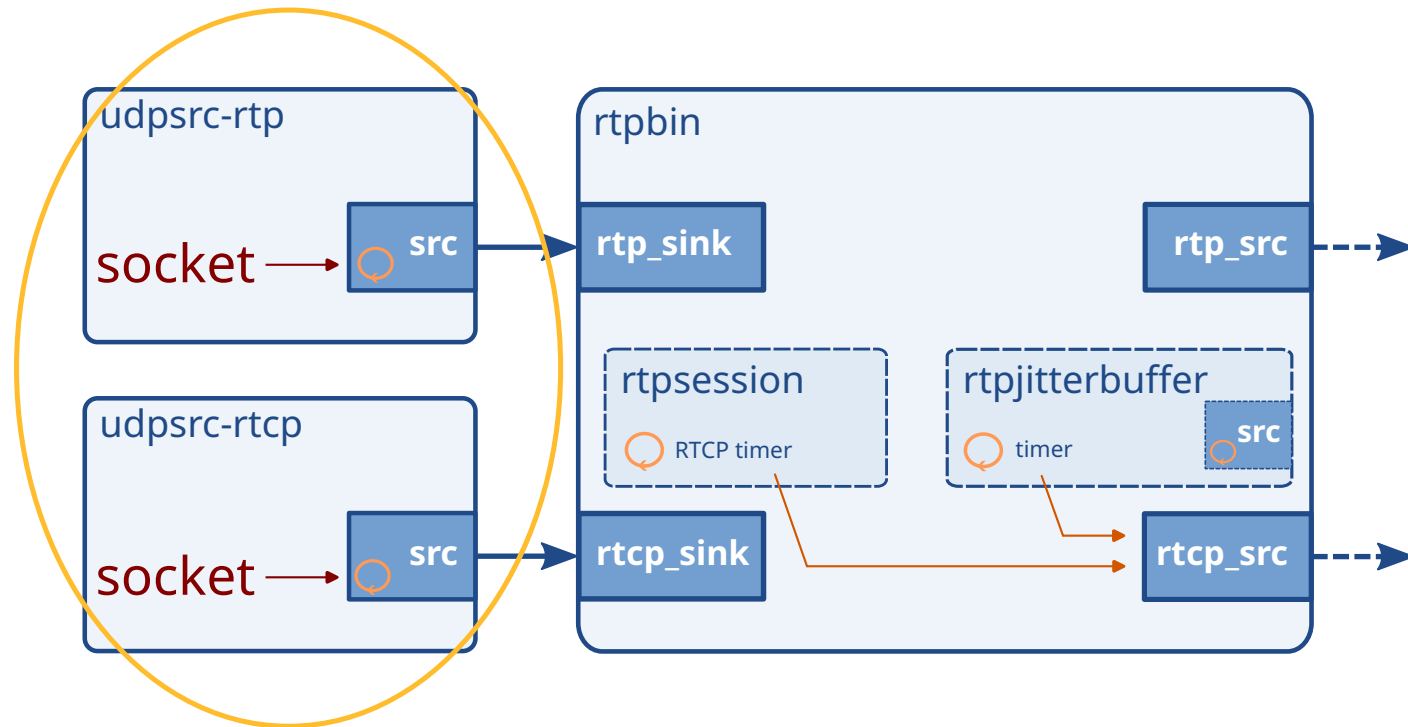


rtpbin receiver

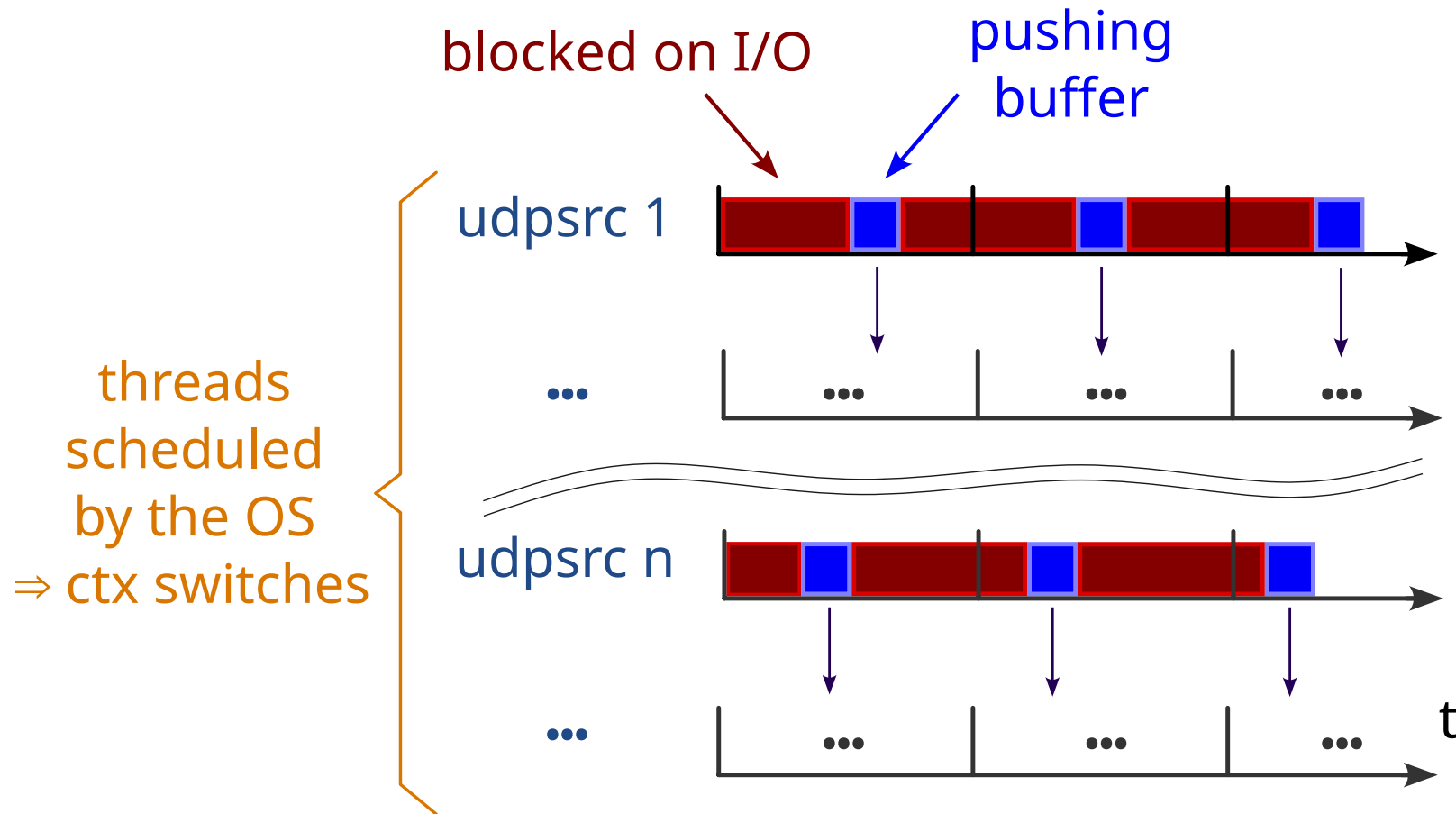


At least **5 threads** per session
=> 2,500+ threads for 500 sessions

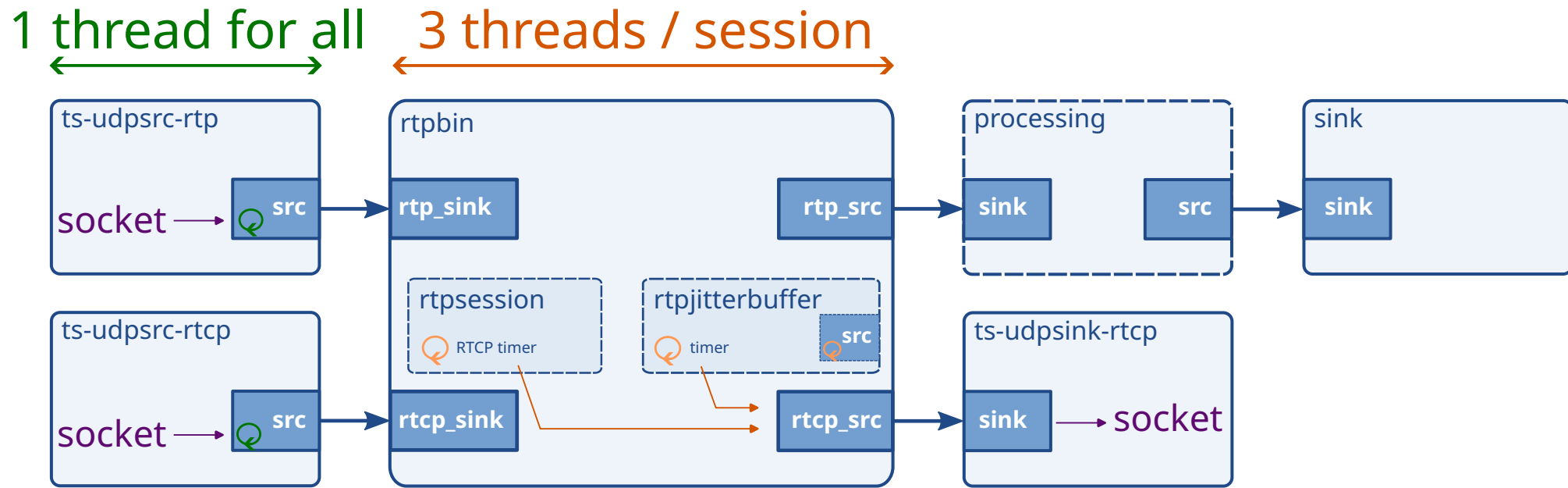
udpsrc focus



udpsrc timeline



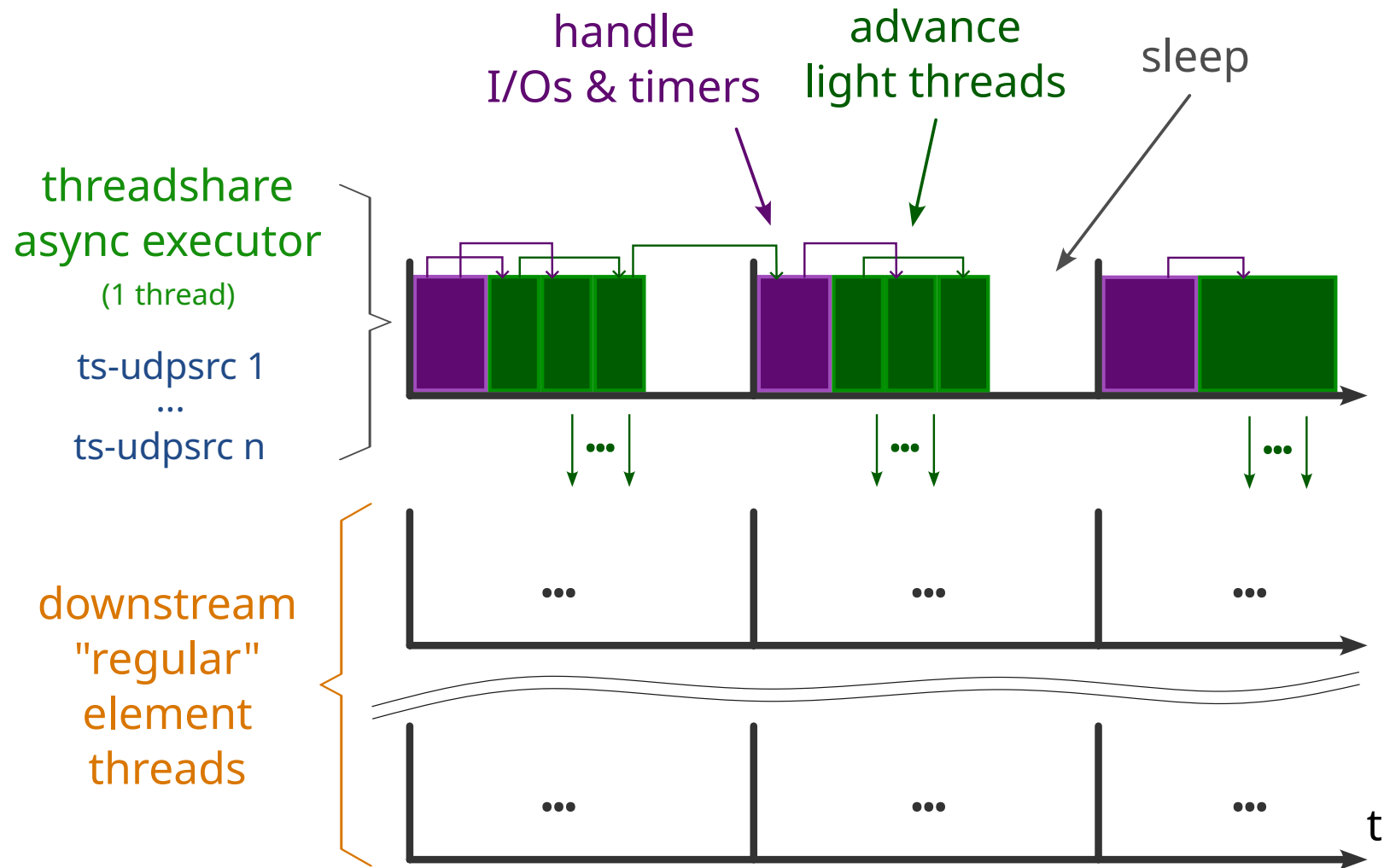
Threadshare **rtpbin** receiver



1 threads for all + at least 3 threads per session

⇒ 1,501+ threads for 500 sessions (999 saved)

Threadshare timeline



Light thread?

- A **Future** spawned onto an async executor $\Leftrightarrow$ *async task*

async function $\Rightarrow$ *Future*

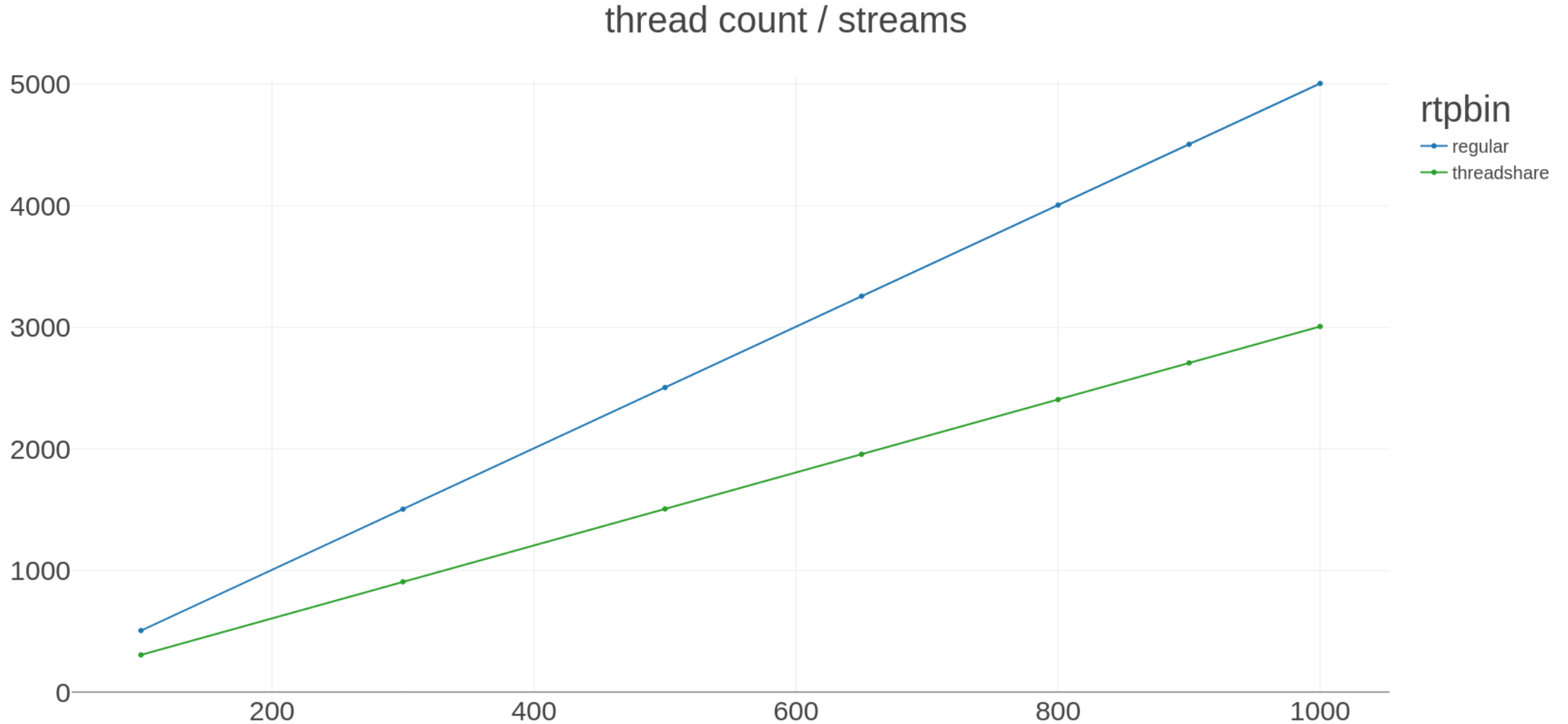
- either *spawned* onto an async executor
- or *awaited* by another *Future*

await points $\Rightarrow$ *Future* evaluation

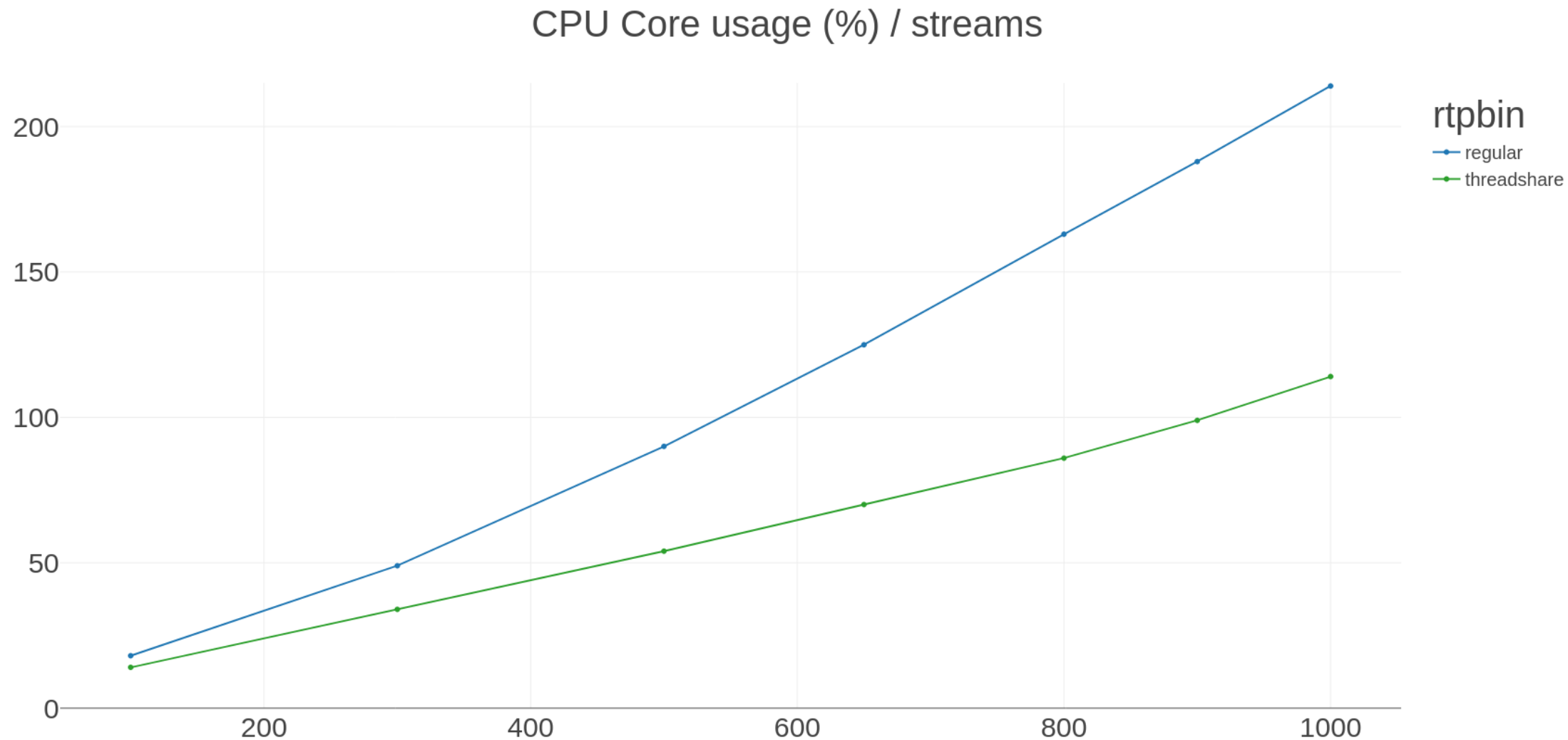
- when *Pending*
yield control to the executor
- when *Ready*
evaluate to the *Future's* result

```
async fn run(&self) {  
  loop {  
    let res = self.socket.try_next().await;  
  
    match res {  
      Ok(buffer) => self.handle_packet(buffer).await,  
      Err(_) => break,  
    }  
  }  
}
```

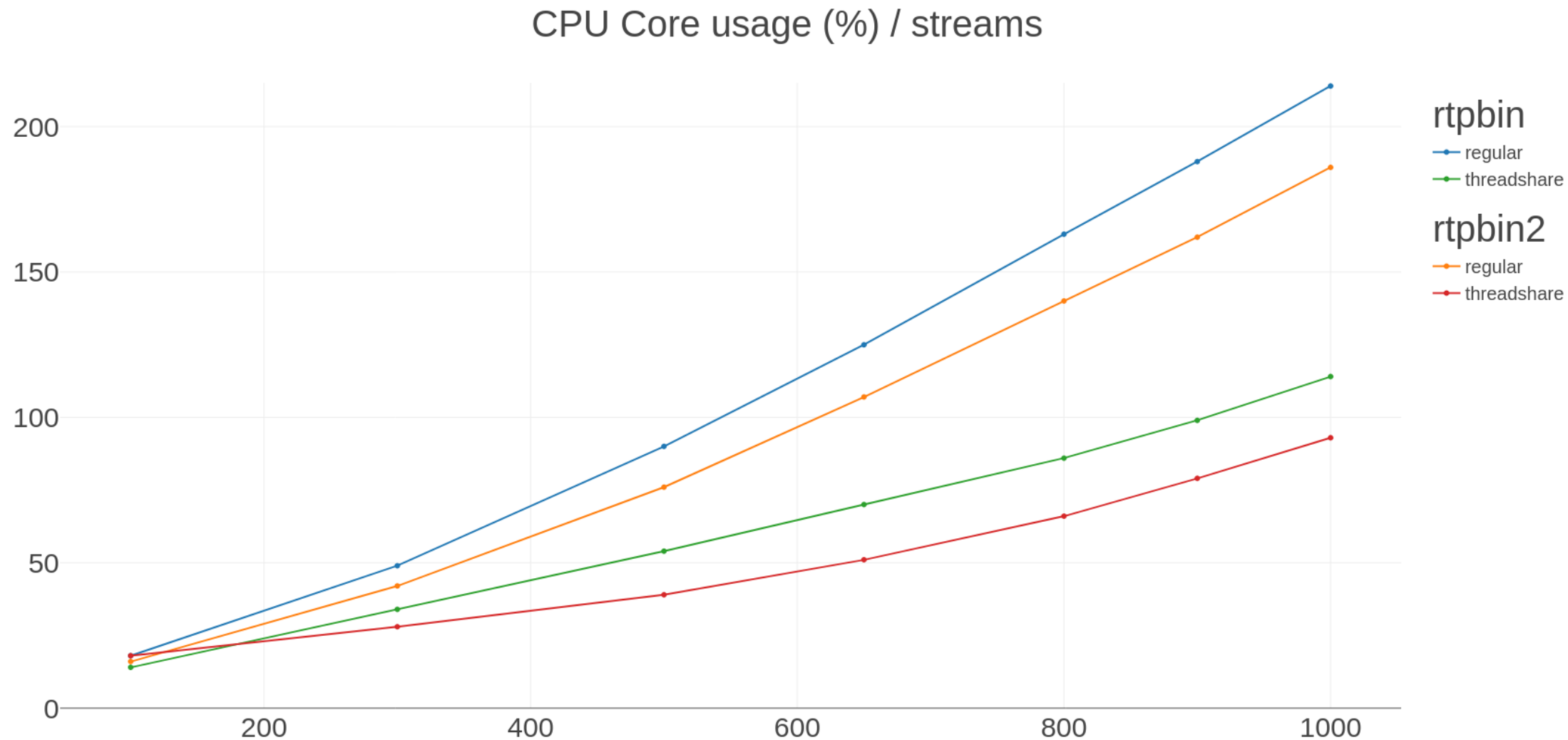
rtpbin thread count



rtpbin CPU usage



rtpbin2 CPU usage



Threadshare components — Context

- Interface to a particular threadshare async executor
- There can be several of them within an application each using its own thread
- Can be referred to by its name - `context` property of ts-elements
- Time slice duration configured at creation - `context-wait` property
- Can be used to `spawn` "light threads" from any threads
- Can be used to register I/O handlers or timers from any threads

Threadshare components — async Pads new

- threadshare `PadSrc` & `PadSink`
⇒ add an *async* "nature" to `gst::Pad`

```
async fn handle_packet(&self, buffer: gst::Buffer) -> Result<gst::FlowSuccess, gst::FlowError> {  
    ...  
    self.src_pad.push(buffer).await  
}
```

- `PadSrc` works equally with regular or ts downstream elements
- `PadSink` automatically adapts to regular or ts upstream elements

Threadshare components — Task new

- An **async event loop** for threadshare elements
- Similar to `gst::Task`
- Built-in state machine with user-defined transition actions
- Warning: name collision with async executor *async task* $\Leftrightarrow$ light thread

Threadshare elements — src / sink

- appsrc
- tcpclientsrc
- udpsrc
- udpsink - can sync on the clock
- audiotestsrc **new**
- rtpdtmfsrc **new**

Threadshare elements — inter-pipelines

Within the same process

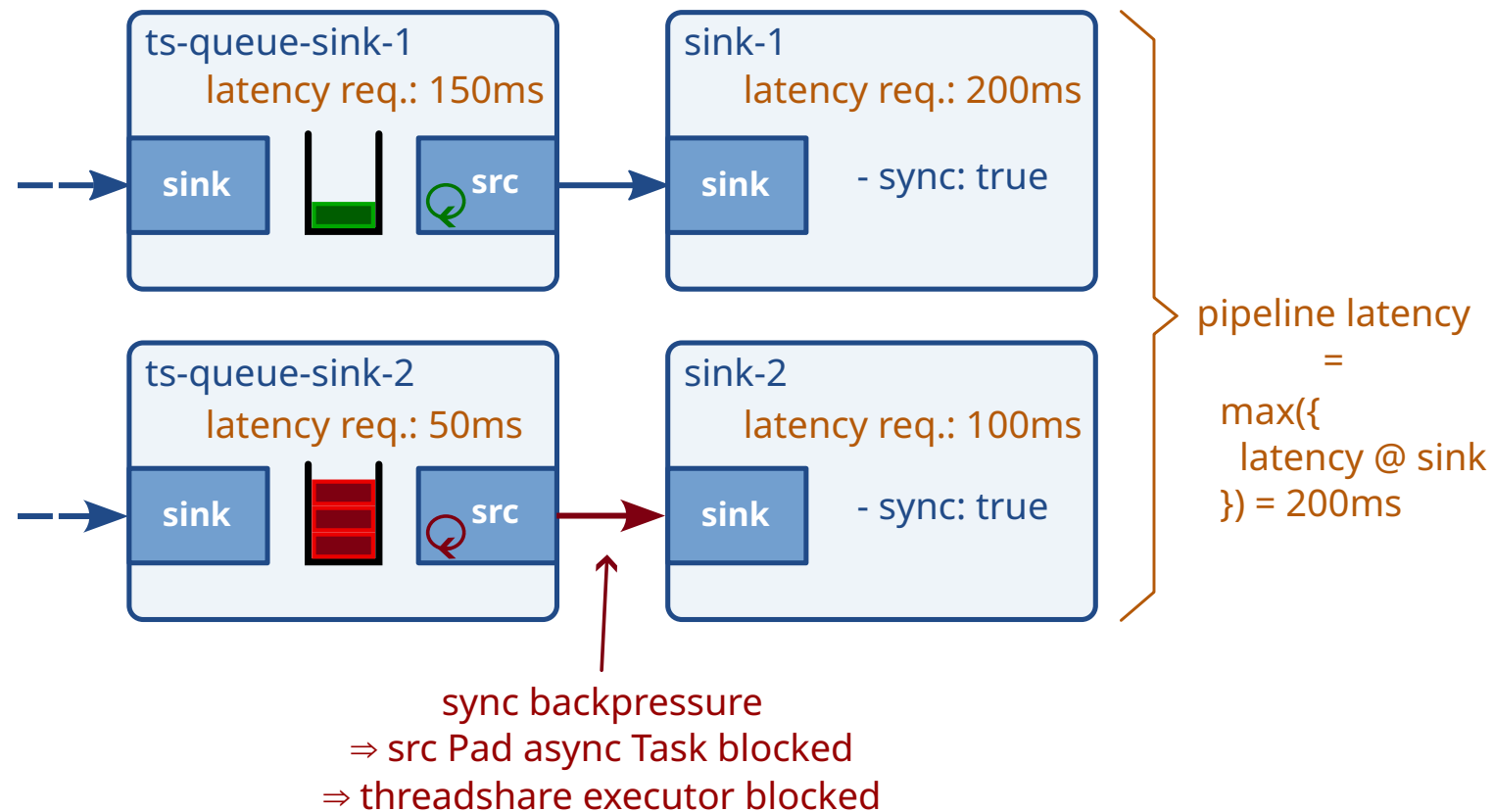
- `proxysink` & `proxysrc`
1:1 with backpressure
- `intersink new` & `intersrc new`
1:n without backpressure

Threadshare elements — misc

- `inputselector`
- `queue`
- `blocking-adapter` **new** - see next slides

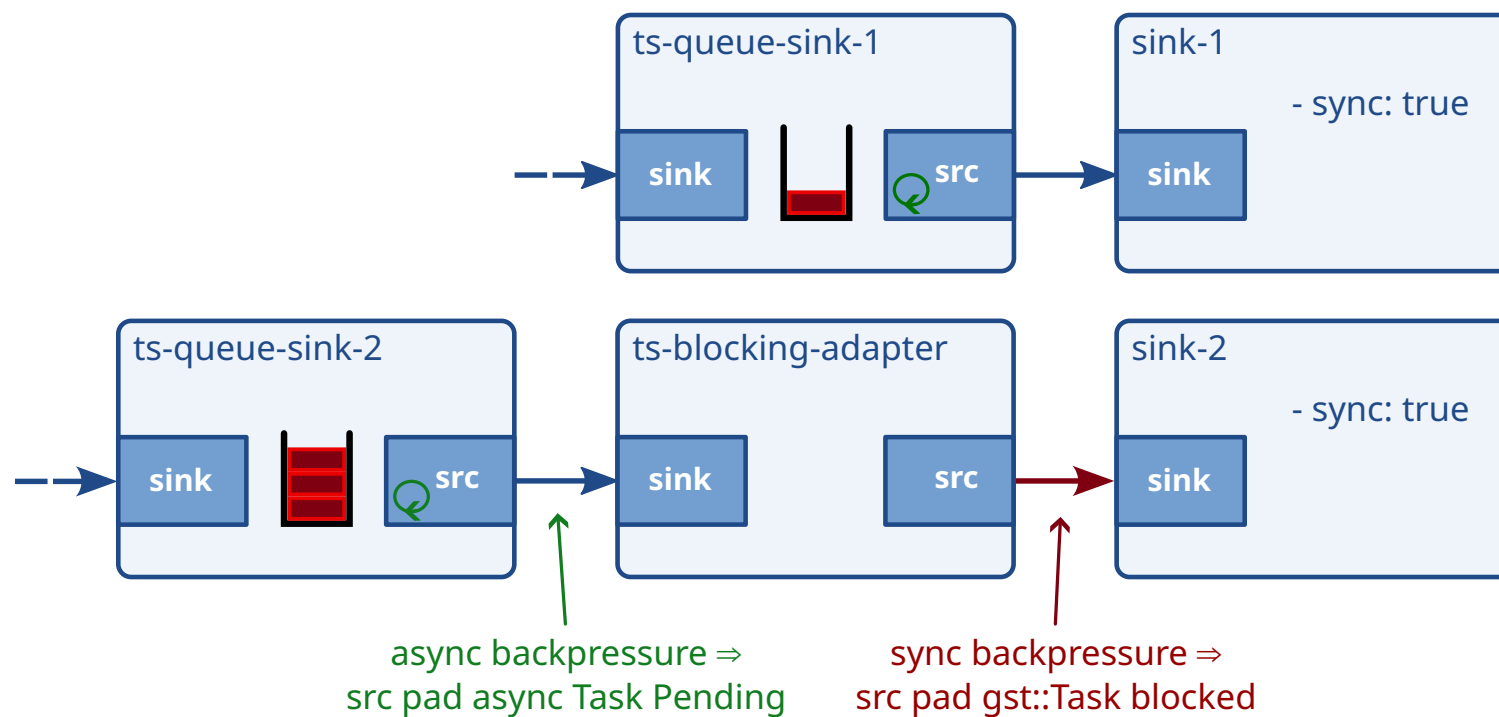
Threadshare **blocking-adapter**

Example: 2 branches with sync sinks & different latency requirements



Threadshare **blocking-adapter**

Solution



Limitations

- Applicable to live streams
- Threadshare elements usually add latency
- Challenging to set things up
- Rust only

Possible improvements

- Profiling
- Base classes
- Name adjustments

Previous talks about threadshare

- When adding more threads adds more problems
GStreamer conference 2018
<https://gstconf.ubicast.tv/videos/when-adding-more-threads-adds-more-problems-thread-sharing-between-elements-in-gstreamer/>
- Real-Time Network Audio with GStreamer on Windows
GStreamer conference 2024
<https://gstconf.ubicast.tv/videos/real-time-network-audio-with-gstreamer-on-windows/>

Thank you for your attention!

Any questions?