gstreamer

GStreamer Conference 2016 Berlin

Profiling GStreamer pipelines

Kyrylo Polezhaiev `<kirushyk@gmail.com>`
`github.com/kirushyk`

- I tried to make this speech more fun than previous one

GStreamer Instruments

`github.com/kirushyk/gst-instruments`

What do we want to inspect?

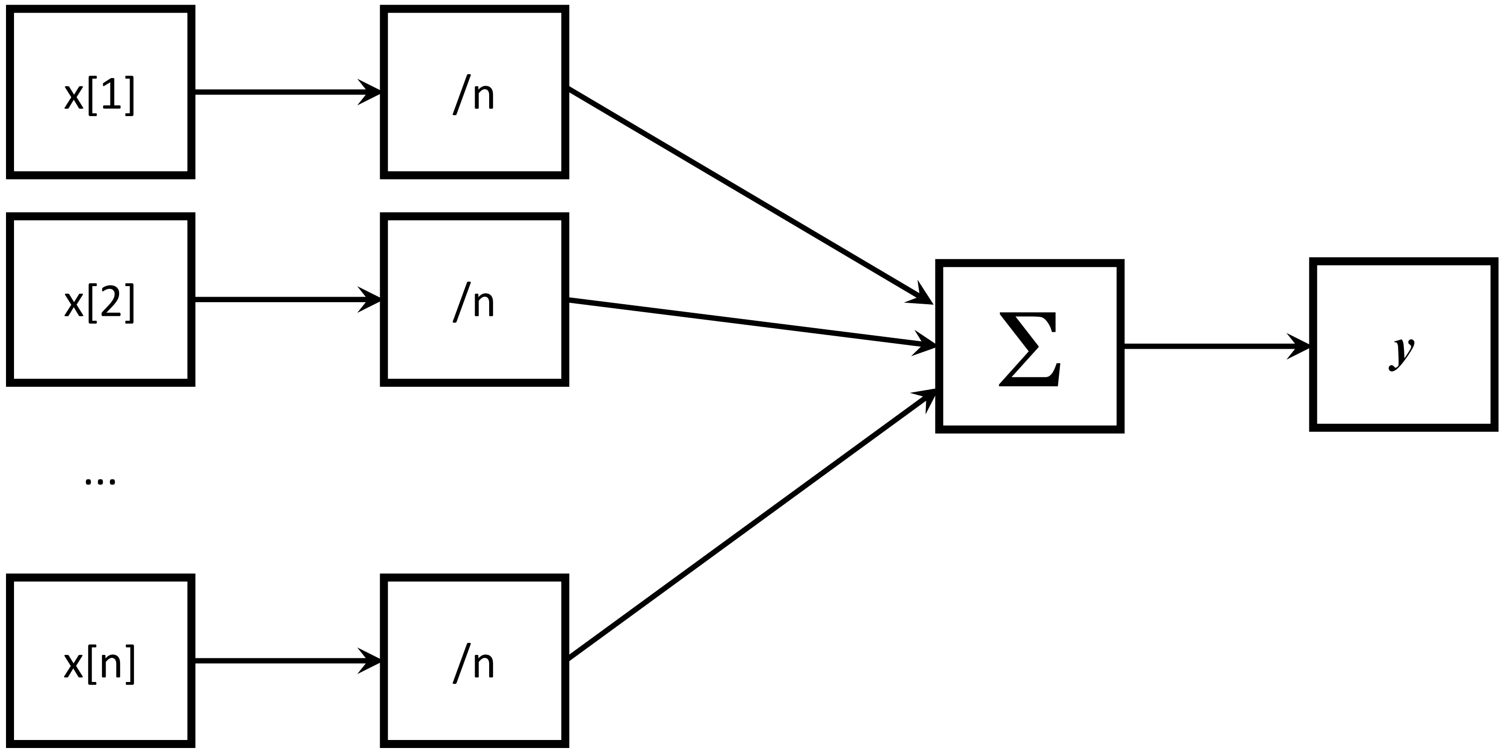
- Elements work
- Data pulling and pushing between elements

Abstraction in optimizations

- Let’s say we want calculate arithmetical mean for set of values:

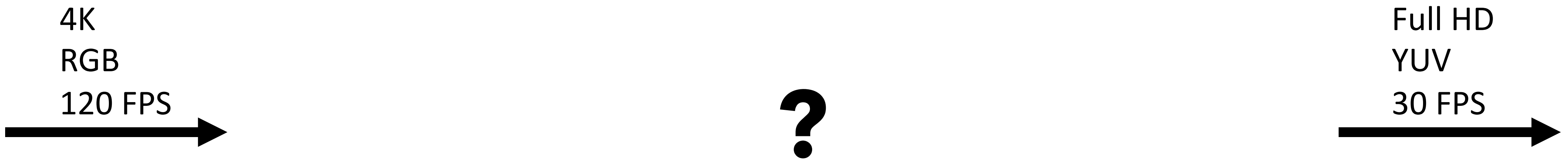
Abstraction in optimizations

- Let’s say we want calculate arithmetical mean for set of values:



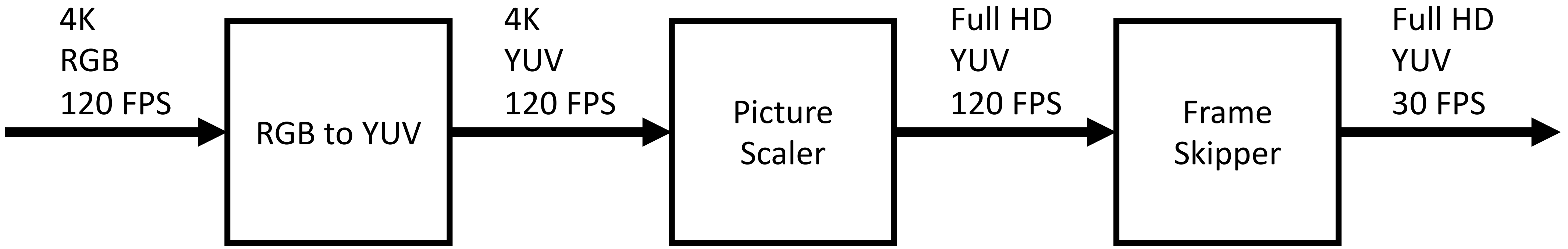
Abstraction in optimizations

- Let’s say we want to convert 4K RGB video stream at 120 FPS to 30 FPS Full HD YUV.



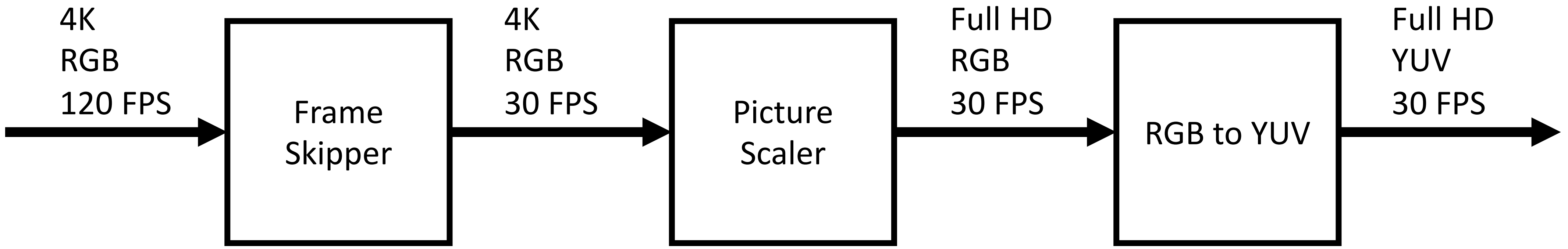
Abstraction in optimizations

- Let's say we want to convert 4K RGB video stream at 120 FPS to 30 FPS Full HD YUV.

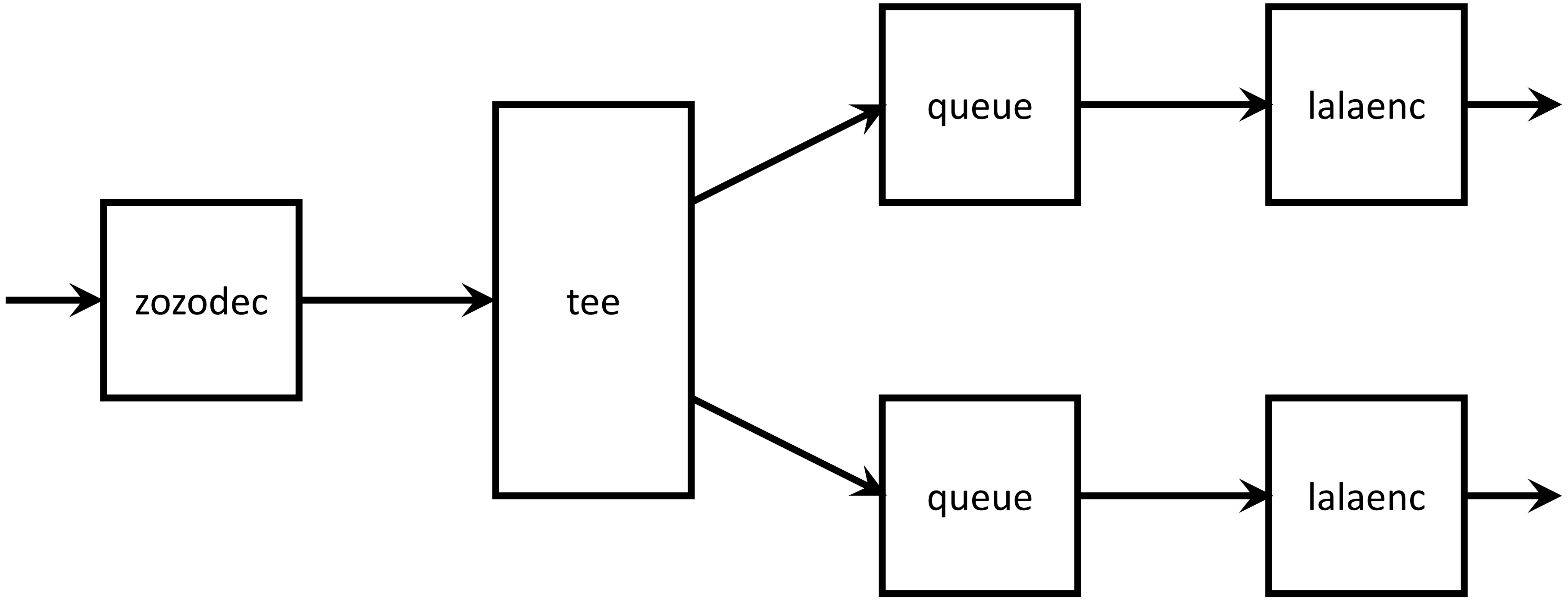


Abstraction in optimizations

- Let’s say we want to convert 4K RGB video stream at 120 FPS to 30 FPS Full HD YUV.



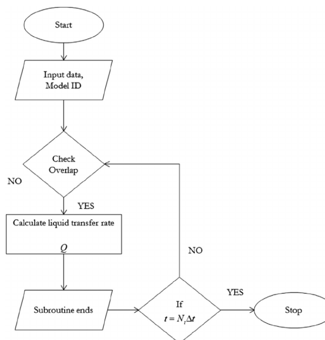
Pipeline is Abstraction

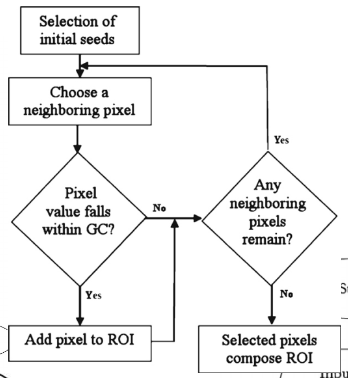


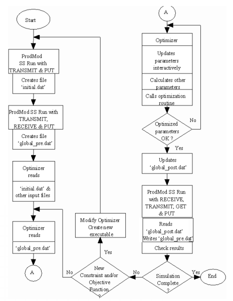
Chip or Tag: Abstractions for Instruction Pointer

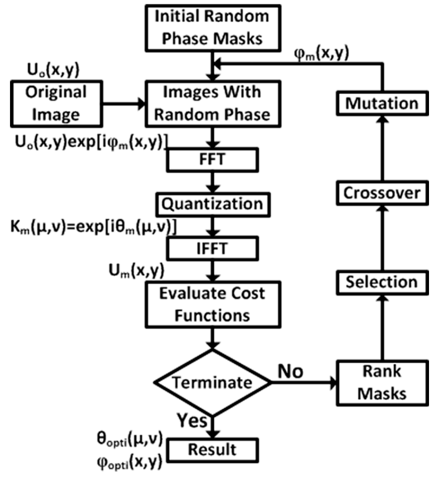
- There is a thing named Program Counter
- Processors have Instruction Pointer Register

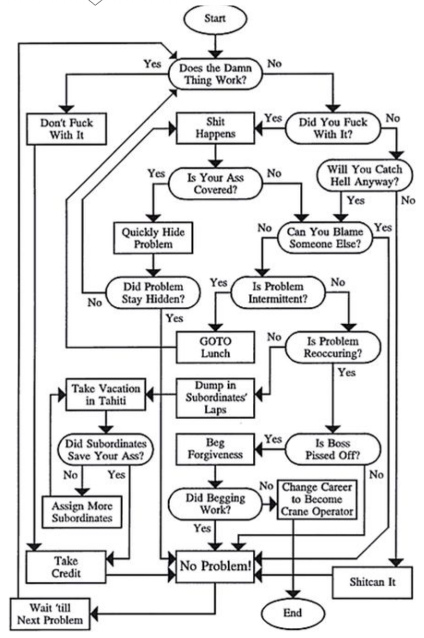
Abstractions for Instruction Pointer

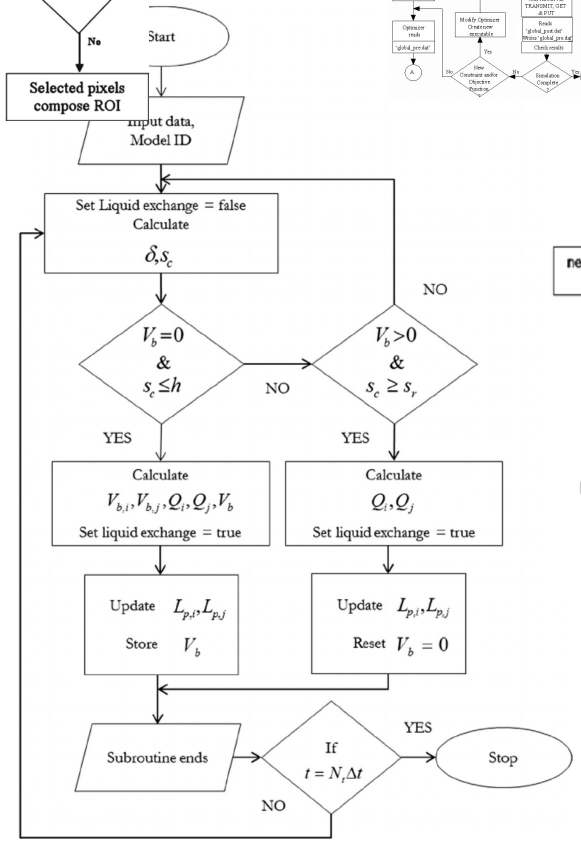


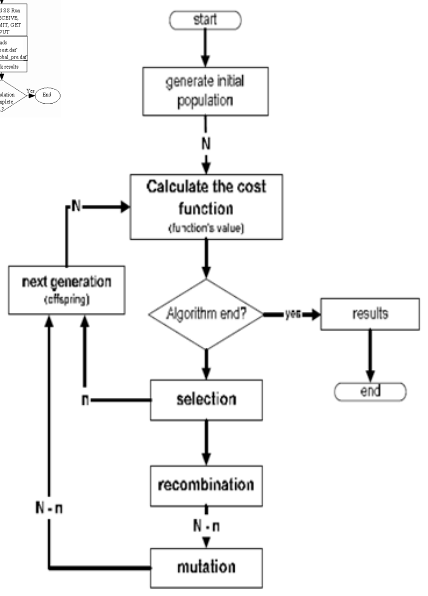

Selection of initial seeds
Choose a neighboring pixel
Pixel value falls within GC?
No
Yes
Any neighboring pixels remain?
Yes
No
Add pixel to ROI
Selected pixels compose ROI



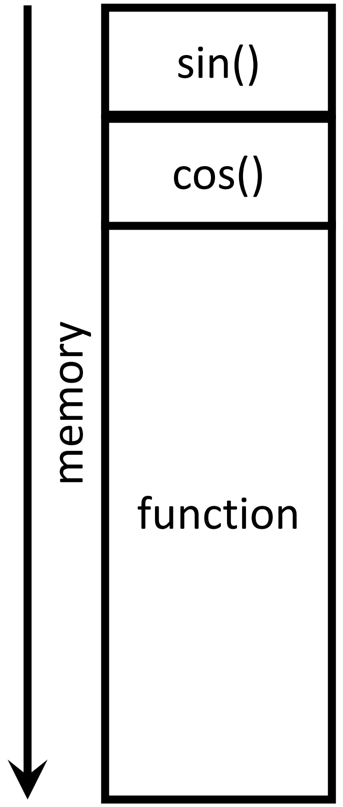






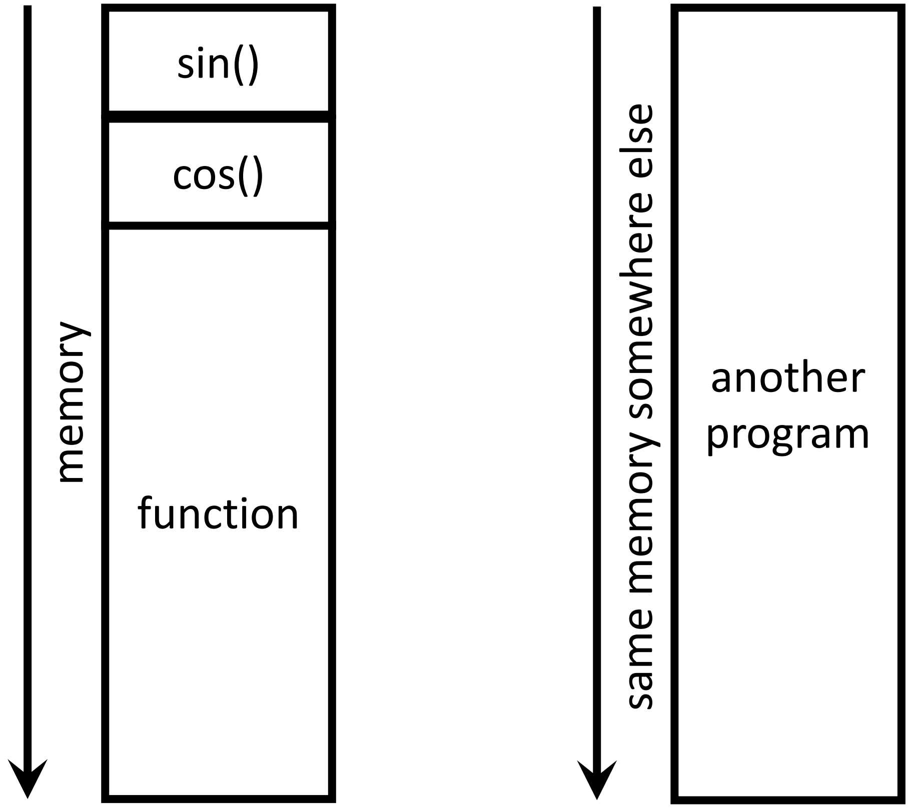


Instruction Pointer Abstraction



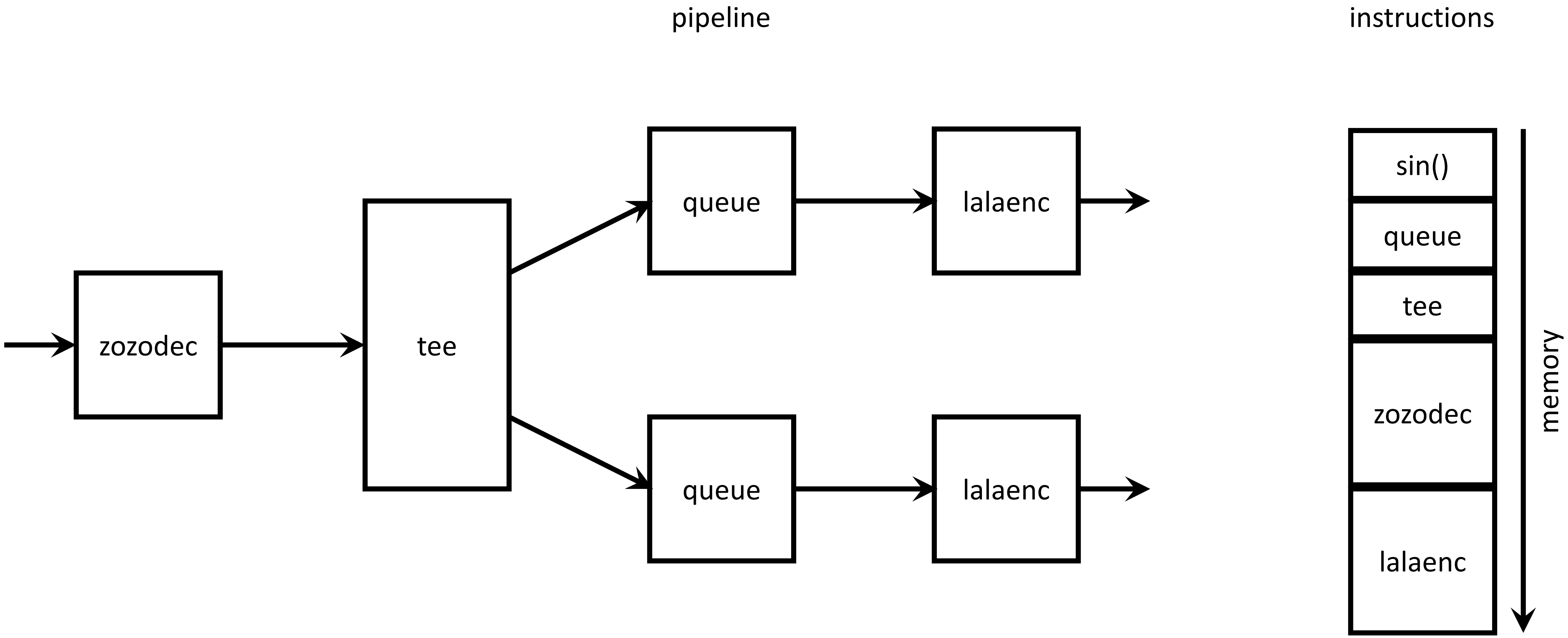
- We can travel via gotos or ifs
- We can call procedures

Instruction Pointer Abstraction

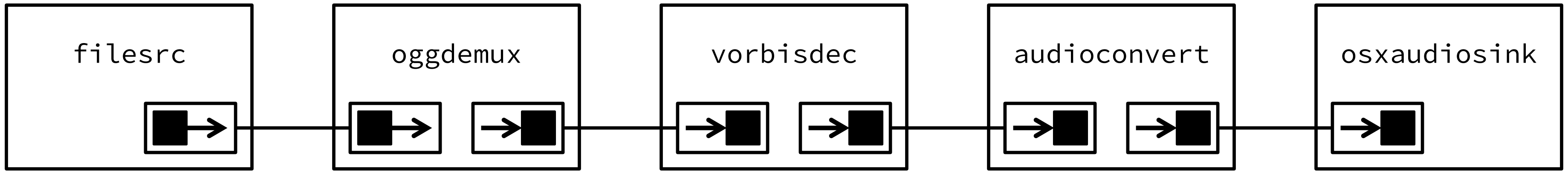


- OS can switch processes
- But we shouldn’t care about this, no goto help needed
- We may have multiple threads on single core

Pipeline is Abstraction

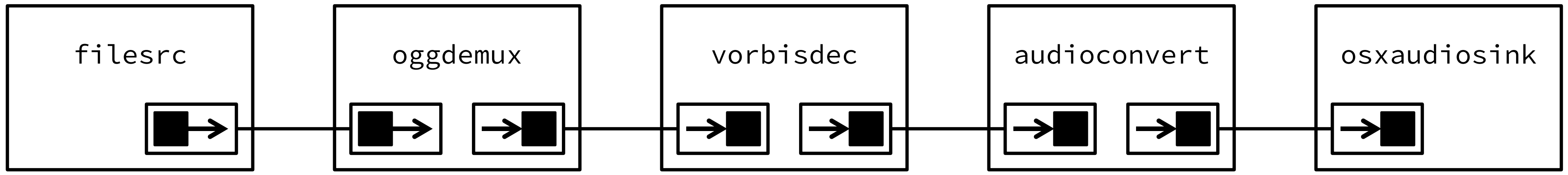


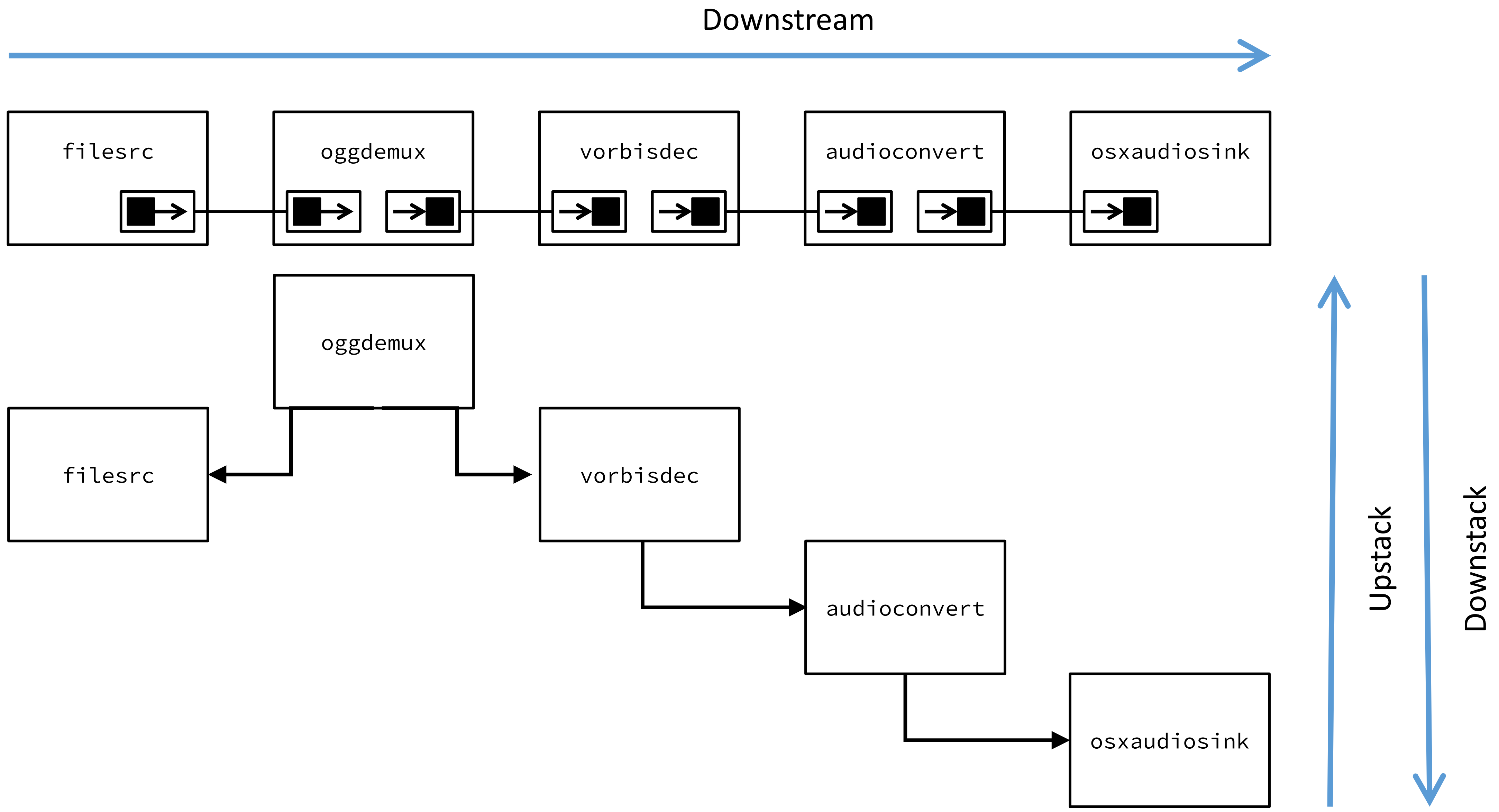
Upstream / Downstream

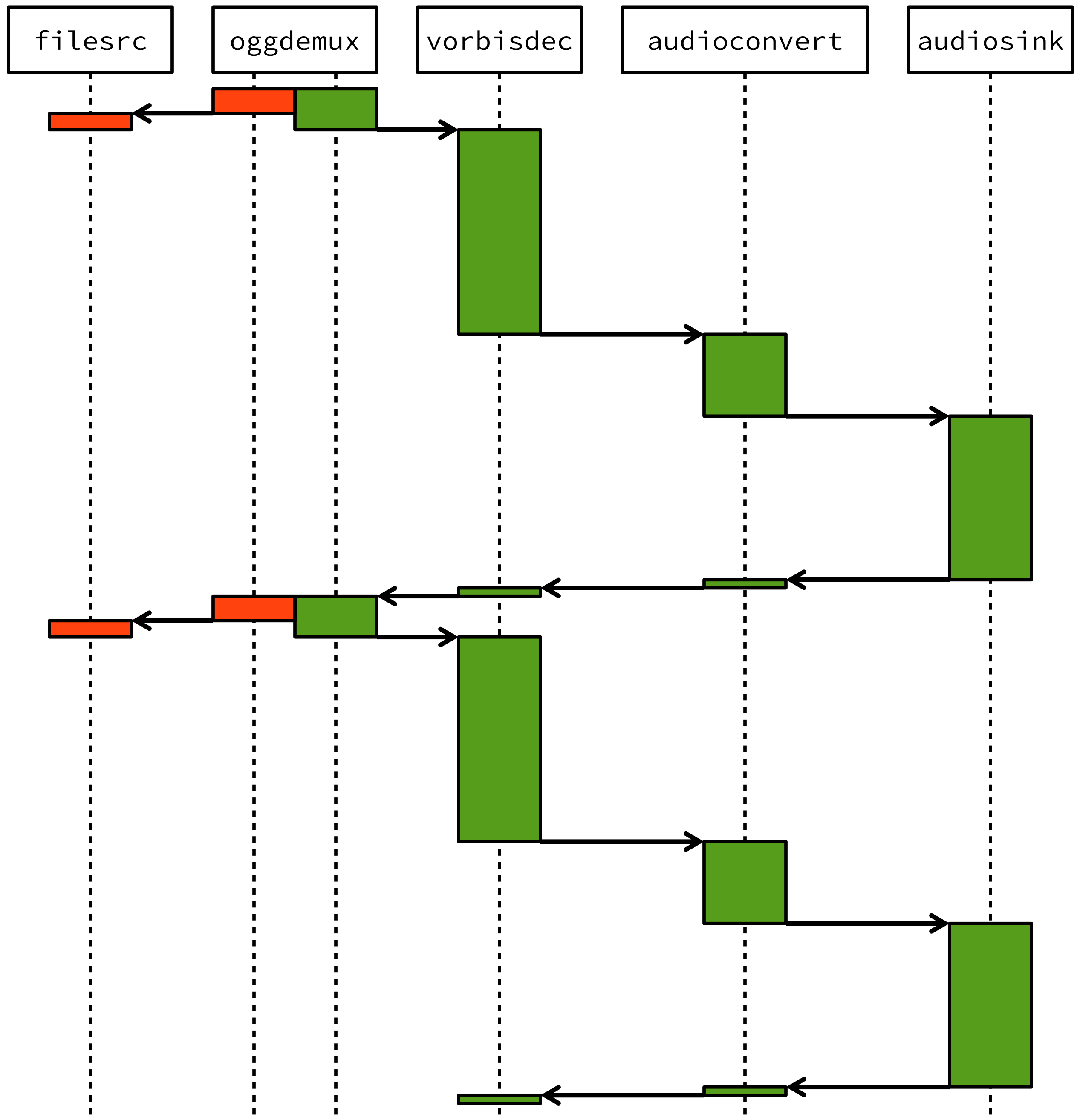


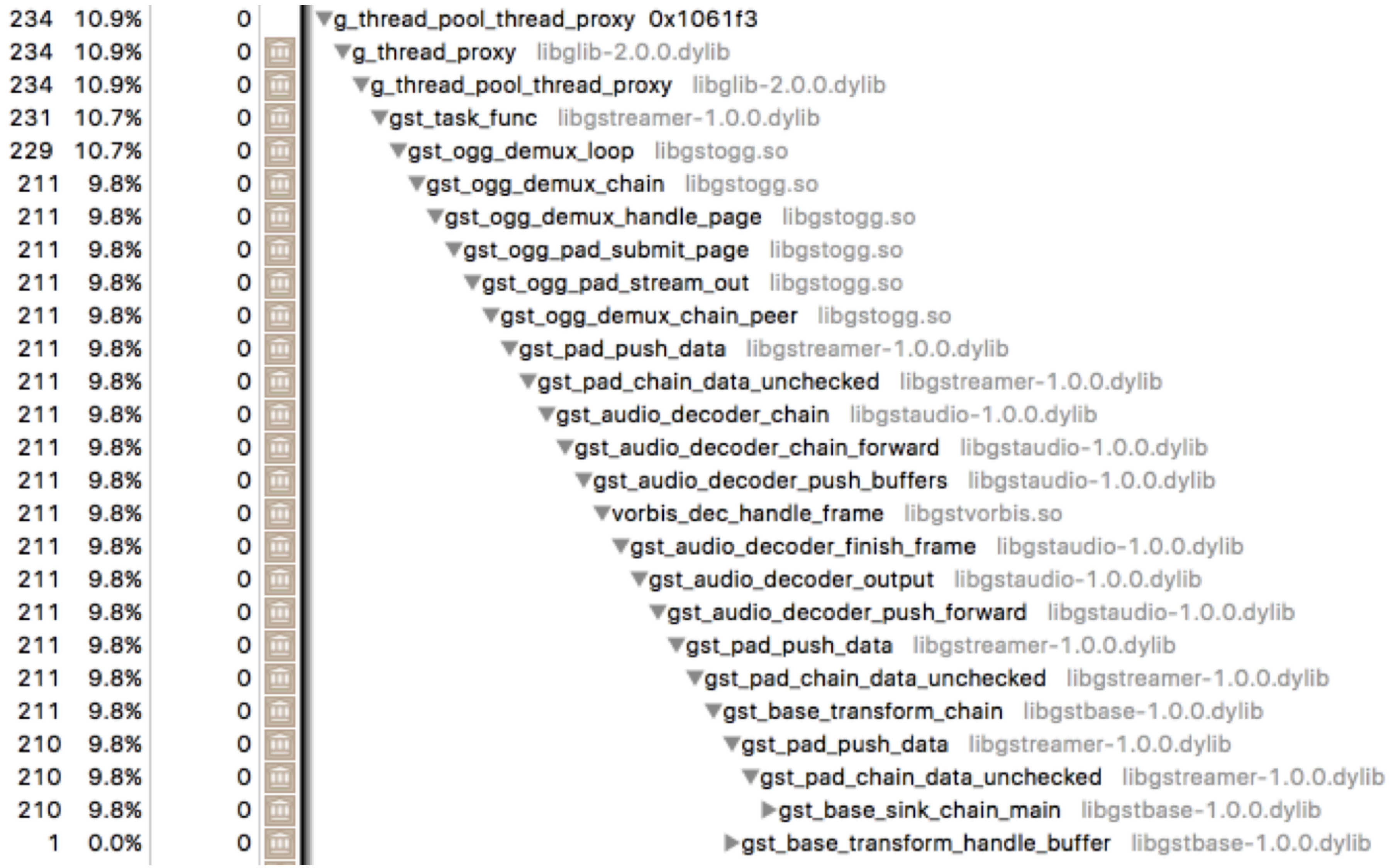
- Is there goto somewhere?
- No, function calls!

Upstream / Downstream



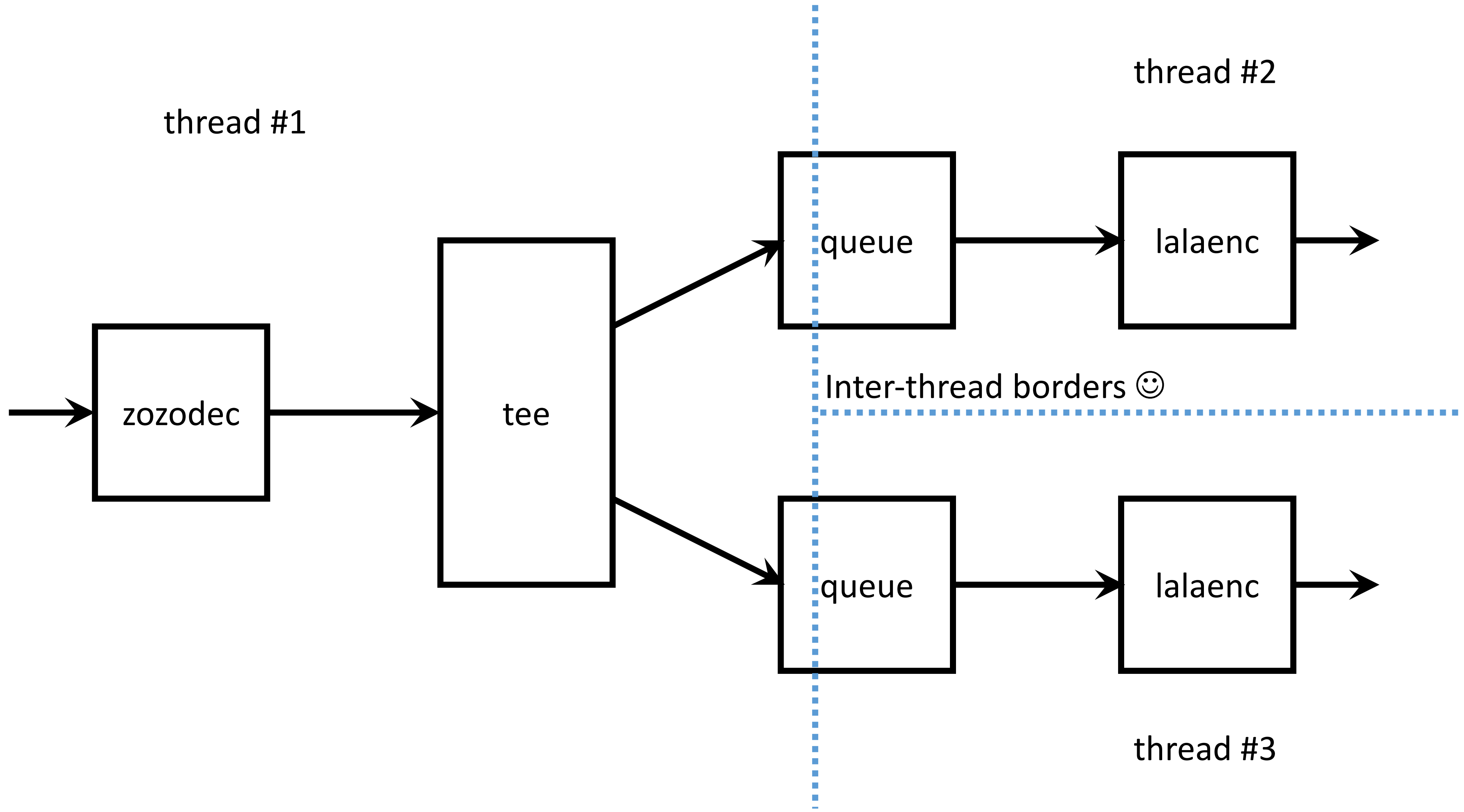

Downstream
filesrc
oggdemux
vorbisdec
audioconvert
osxaudiosink
oggdemux
filesrc
vorbisdec
audioconvert
osxaudiosink
Upstack
Downstack


filesrc
oggdemux
vorbisdec
audioconvert
audiosink

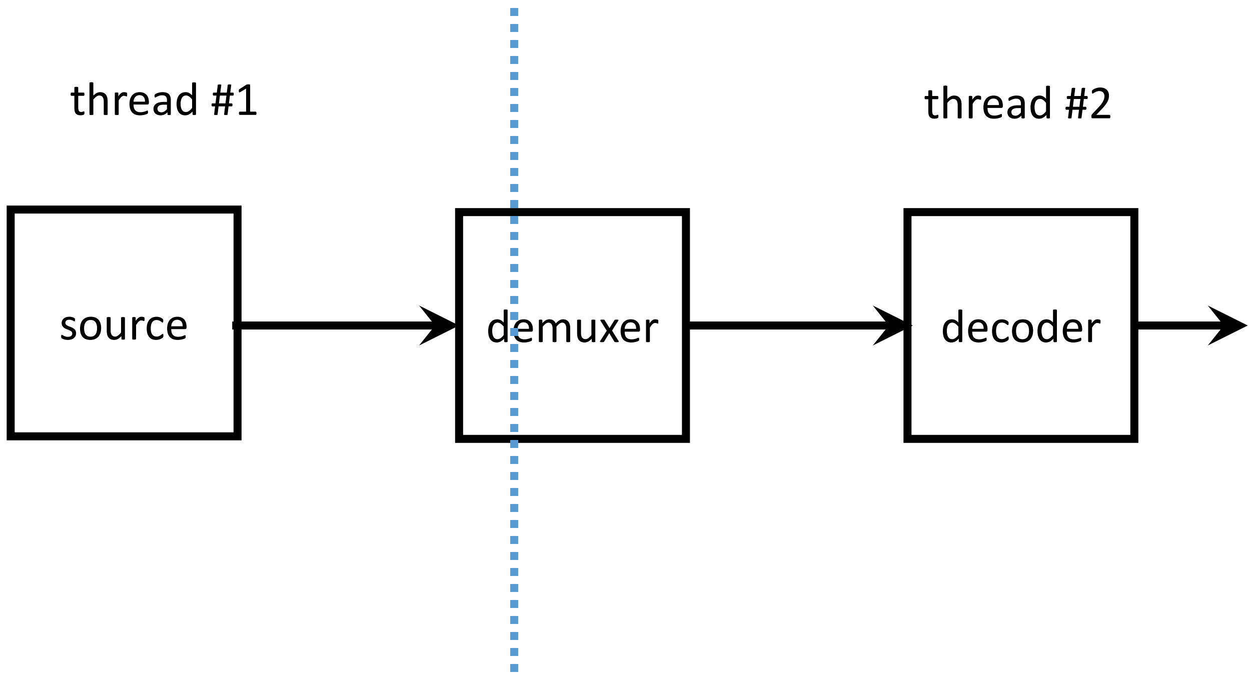


```
234  10.9%   0  ▼g_thread_pool_thread_proxy  0x1061f3
234  10.9%   0   ▼g_thread_proxy  libglib-2.0.0.dylib
234  10.9%   0    ▼g_thread_pool_thread_proxy  libglib-2.0.0.dylib
231  10.7%   0     ▼gst_task_func  libgstreamer-1.0.0.dylib
229  10.7%   0      ▼gst_ogg_demux_loop  libgstogg.so
211   9.8%   0       ▼gst_ogg_demux_chain  libgstogg.so
211   9.8%   0        ▼gst_ogg_demux_handle_page  libgstogg.so
211   9.8%   0         ▼gst_ogg_pad_submit_page  libgstogg.so
211   9.8%   0          ▼gst_ogg_pad_stream_out  libgstogg.so
211   9.8%   0           ▼gst_ogg_demux_chain_peer  libgstogg.so
211   9.8%   0            ▼gst_pad_push_data  libgstreamer-1.0.0.dylib
211   9.8%   0             ▼gst_pad_chain_data_unchecked  libgstreamer-1.0.0.dylib
211   9.8%   0              ▼gst_audio_decoder_chain  libgstaudio-1.0.0.dylib
211   9.8%   0               ▼gst_audio_decoder_chain_forward  libgstaudio-1.0.0.dylib
211   9.8%   0                ▼gst_audio_decoder_push_buffers  libgstaudio-1.0.0.dylib
211   9.8%   0                 ▼vorbis_dec_handle_frame  libgstvorbis.so
211   9.8%   0                  ▼gst_audio_decoder_finish_frame  libgstaudio-1.0.0.dylib
211   9.8%   0                   ▼gst_audio_decoder_output  libgstaudio-1.0.0.dylib
211   9.8%   0                    ▼gst_audio_decoder_push_forward  libgstaudio-1.0.0.dylib
211   9.8%   0                     ▼gst_pad_push_data  libgstreamer-1.0.0.dylib
211   9.8%   0                      ▼gst_pad_chain_data_unchecked  libgstreamer-1.0.0.dylib
211   9.8%   0                       ▼gst_base_transform_chain  libgstbase-1.0.0.dylib
210   9.8%   0                        ▼gst_pad_push_data  libgstreamer-1.0.0.dylib
210   9.8%   0                         ▼gst_pad_chain_data_unchecked  libgstreamer-1.0.0.dylib
210   9.8%   0                          ▶gst_base_sink_chain_main  libgstbase-1.0.0.dylib
  1   0.0%   0                        ▶gst_base_transform_handle_buffer  libgstbase-1.0.0.dylib
```

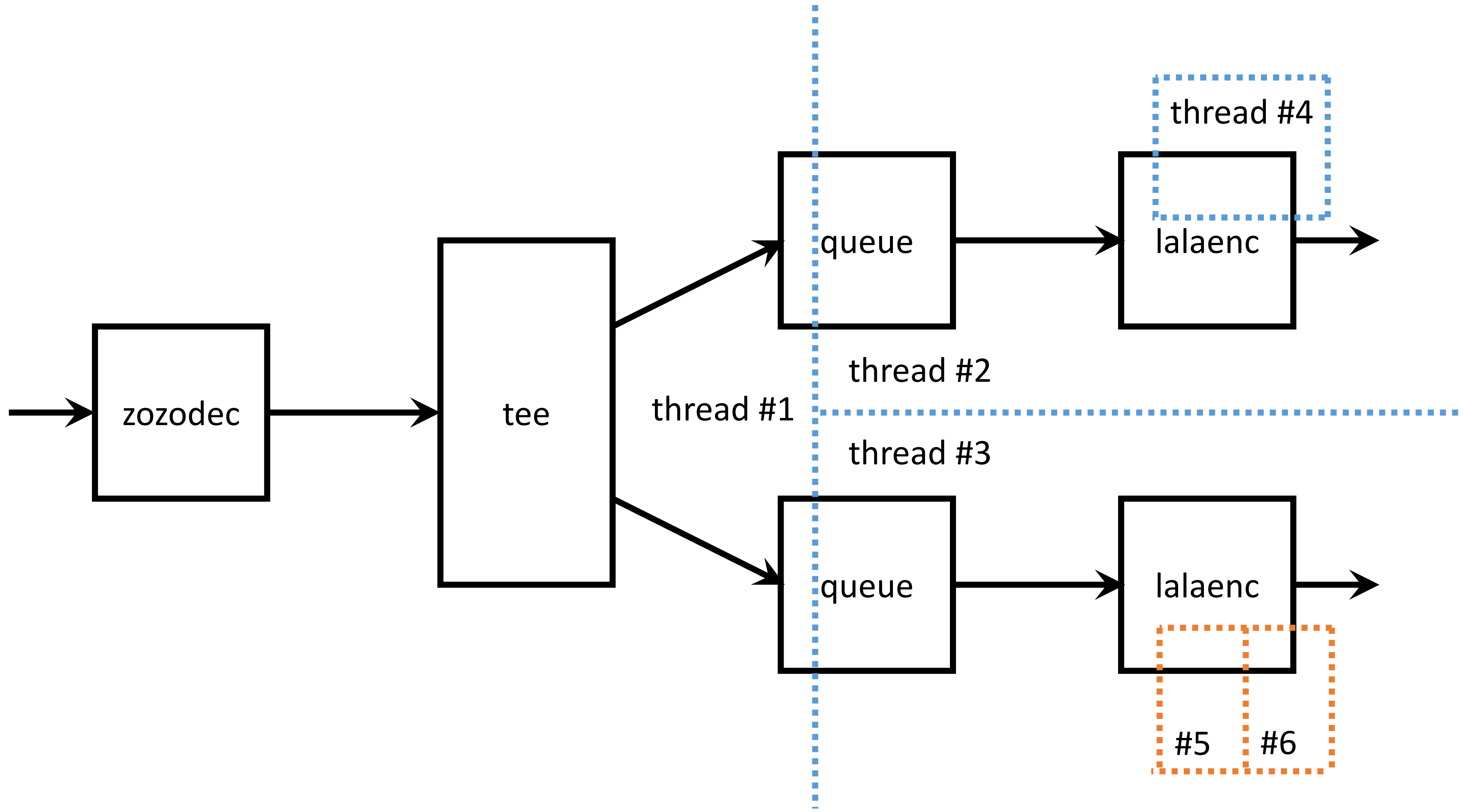
Threads’ realms



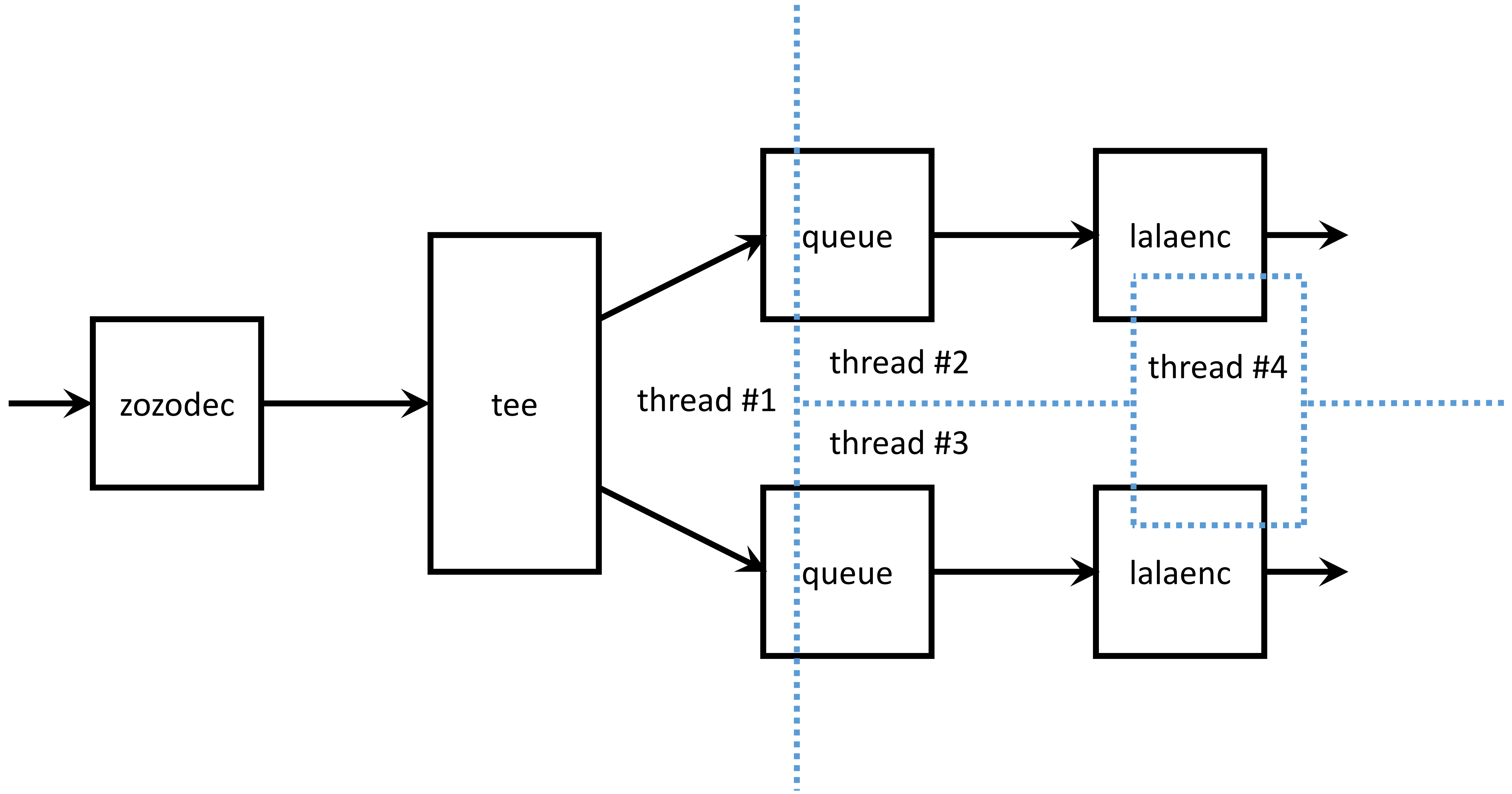
Threads' realms



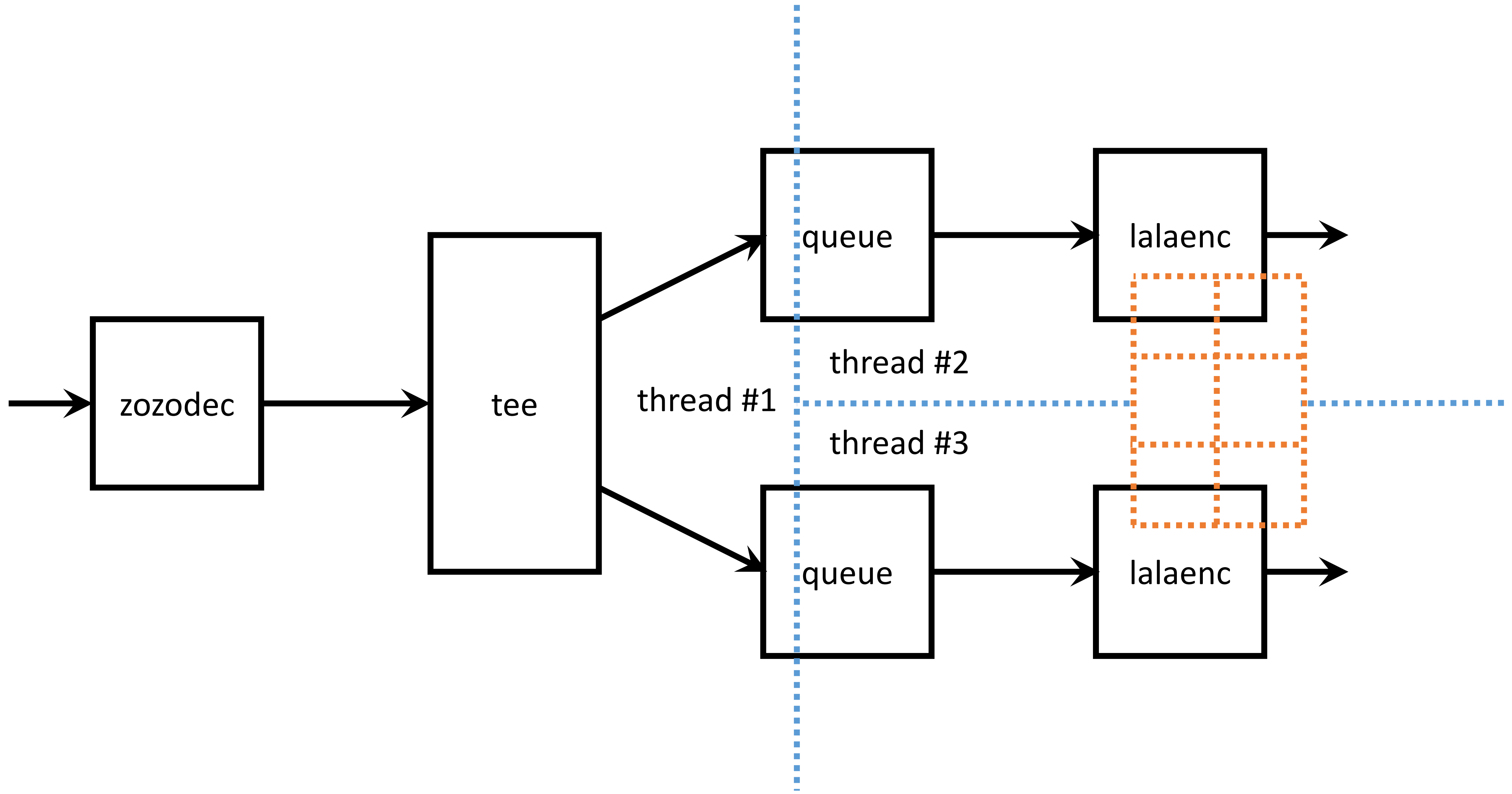
Threads' realms



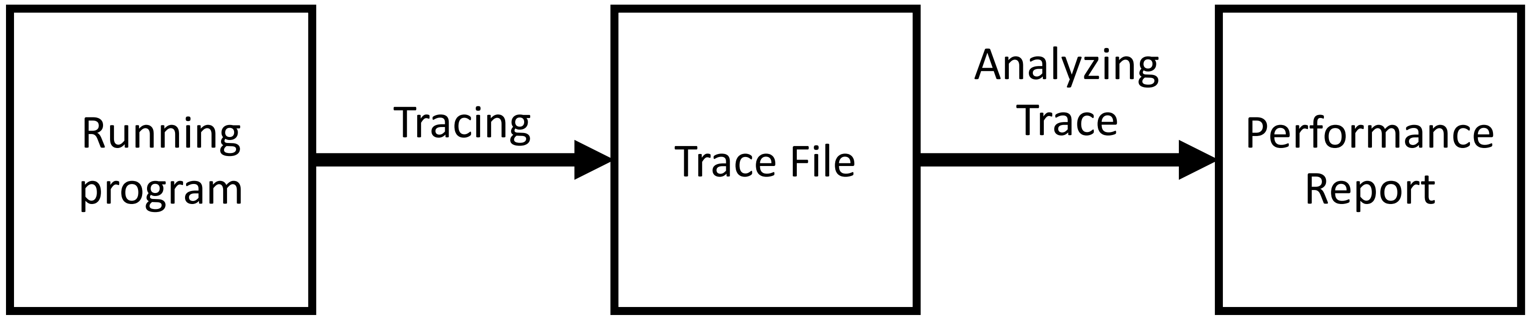
Threads’ realms



Threads' realms



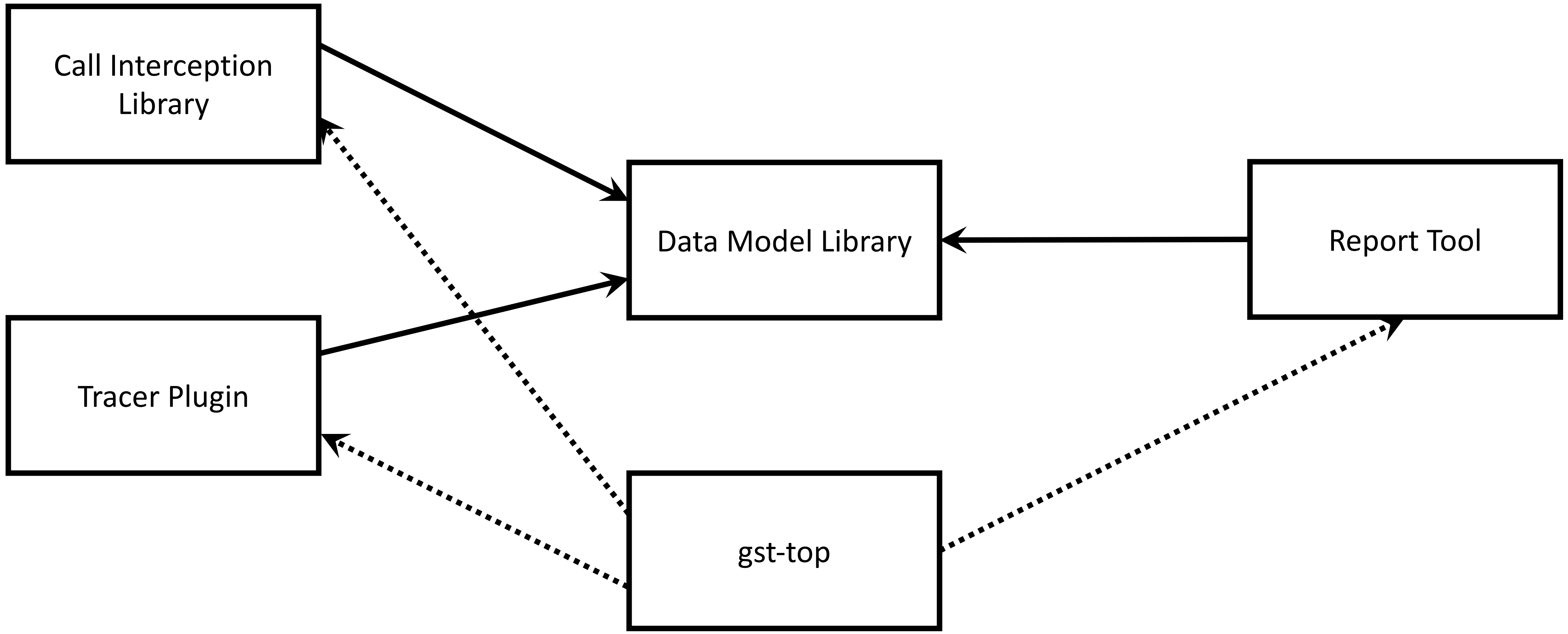
General Idea



Way to trace events in running app

- LD_PRELOAD
- DYLD_INSERT_LIBRARIES + symbol interpose
- GStreamer Tracing Subsystem

Components of GStreamer Instruments



Trick with Linux dynamic linker

- Create .so library containing functions with same names (gst_pad_push, gst_pad_pull_range, etc.)
- That functions can call original ones loaded via dlsym
- Run binary setting LD_PRELOAD environment variable

Trick with Linux dynamic linker

```
wrapper_function()
{
    start = ⏱
    log_event (ENTERINTOELEMENT, start)
    original_function()
    end = ⏱
    duration = end - start
    log_event (EXITFROMELEMENT, duration)
}
```

Trick with Linux dynamic linker

Problems:

- No statically-linked functions calls intercepted
- No way to subtract GTask-related work from upstack time
- No way to measure how many time pulling/pushing takes

Trick with macOS Dynamic Linker

Two kinds of DLLs on Mac:

- Bundle (.bundle or .so)
- Dynamic Library (.dylib)

Trick with macOS Dynamic Linker

- DYLD_INSERT_LIBRARIES instead of LD_PRELOAD
- Set DYLD_FORCE_FLAT_NAMESPACE

Trick with macOS Dynamic Linker

Statically link to library with functions we want to wrap +

```
# define INTERPOSE(_replacment, _replacee) \

__attribute__ ((used)) static struct { const void* replacment; const void* replacee; } _interpose_##_replacee \

__attribute__ ((section ("__DATA,__interpose"))) = { (const void*)(unsigned long)&_replacment, (const
void*)(unsigned long)&_replacee };

INTERPOSE (lgi_pad_push, gst_pad_push);

INTERPOSE (lgi_pad_push_list, gst_pad_push_list);

INTERPOSE (lgi_pad_push_event, gst_pad_push_event);

INTERPOSE (lgi_pad_pull_range, gst_pad_pull_range);

INTERPOSE (lgi_element_set_state, gst_element_set_state);

INTERPOSE (lgi_element_change_state, gst_element_change_state);
```

Trick with macOS Dynamic Linker

Problems:

- We have no enter time in stack
- We have no some hooks we want ☺

Using Tracing Subsystem

- Create library which listens for hooks to be hit

```
gst_tracing_register_hook (tracer, "pad-push-pre",
  G_CALLBACK (do_push_buffer_pre));
gst_tracing_register_hook (tracer, "pad-pull-range-pre",
  G_CALLBACK (do_pull_range_pre));
```

- Run program setting GST_TRACERS environment variable

Interesting events to log

Most interesting:

- Thread entered element
- Thread exited element

Also (less interesting):

- Hierarchy discovered (auxiliary event)
- Data sent (to measure data flows)

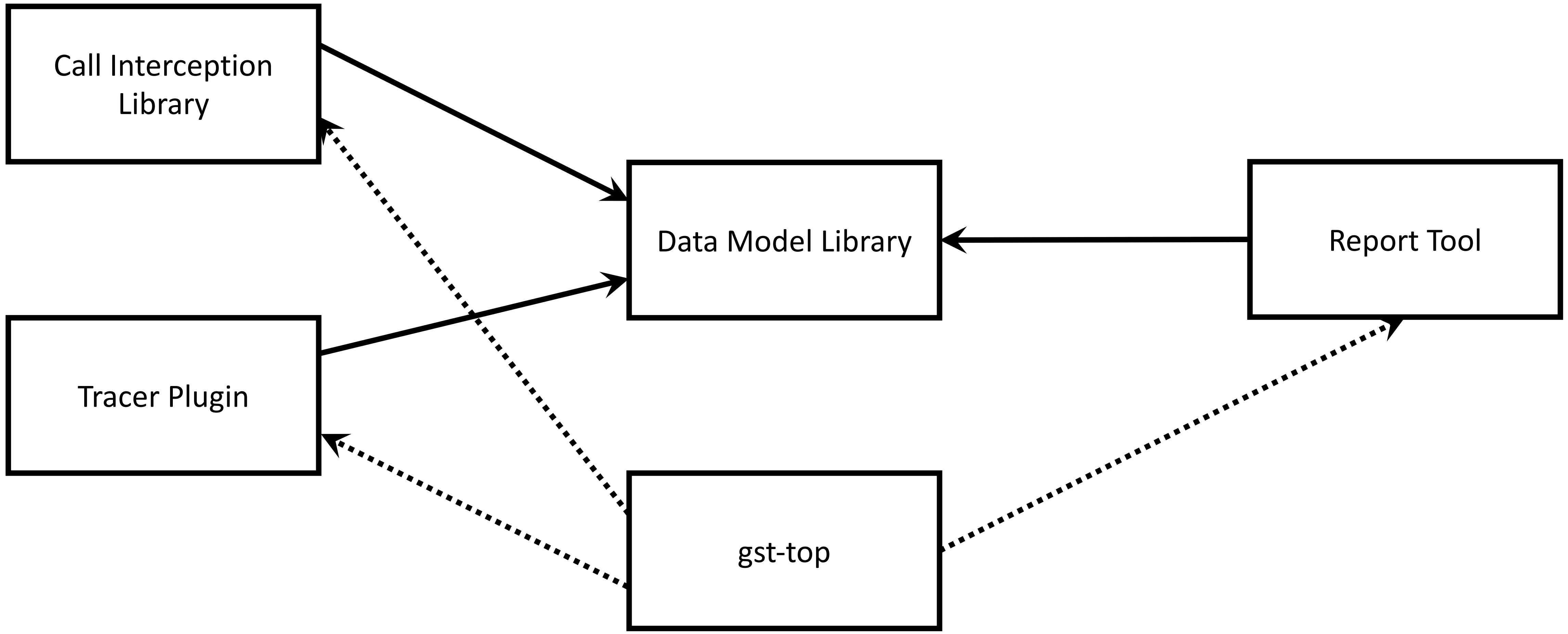
What can we measure?

- Thread execution time
- CPU cycles
- Real time spent

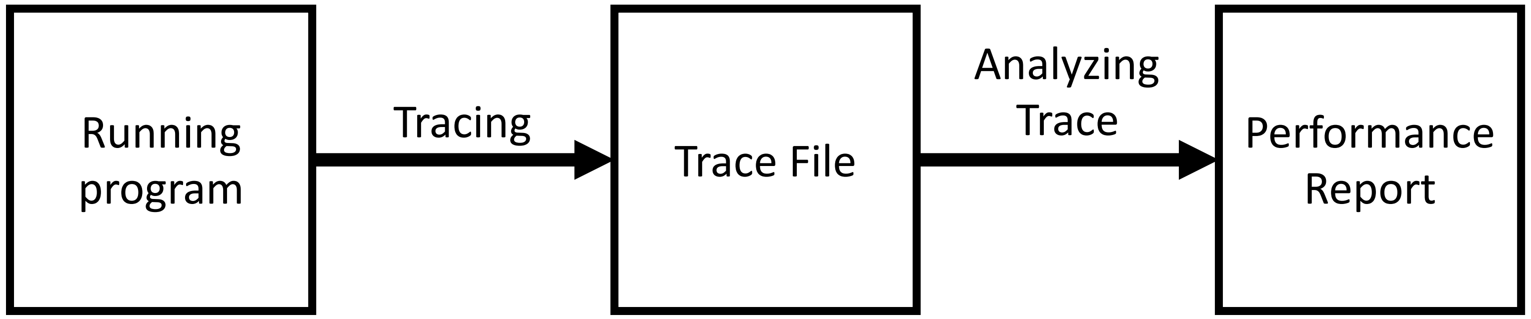
Thread ExecutionTime

- `thread_info (…, THREAD_EXTENDED_INFO, …, …)`
- `clock_gettime (CLOCK_THREAD_CPUTIME_ID, …)`
- `GetThreadTimes (…, …, …, …, …)`

Components of GStreamer Instruments



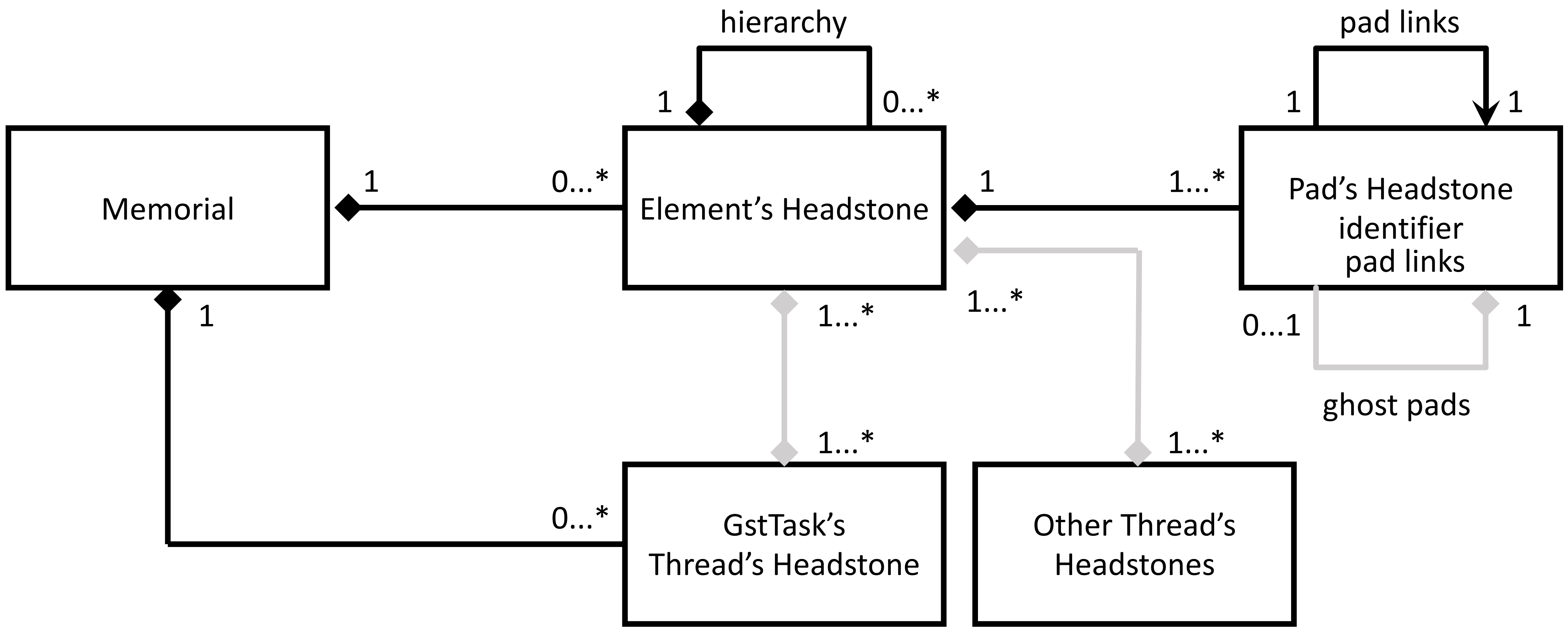
General Idea



Data Model

- Usually, no elements still alive when we do analysis
- Elements have no date of birth and death
- Element’s address can be used as identifier…
- But theoretically new element can be created at same address
- I didn’t think about names a lot

Data Model


hierarchy
pad links
1
0...*
1
1
Memorial
1
0...*
Element's Headstone
1
1...*
Pad's Headstone
identifier
pad links
1
1...*
1...*
0...1
1
ghost pads
1...*
1...*
0...*
GstTask's
Thread's Headstone
Other Thread's
Headstones

$$\begin{tikzcd} \boxed{1} \arrow[r, "push"] & \boxed{2} \arrow[r, "push"] & \boxed{3} \end{tikzcd}$$

- Element #1 ENTER
- Element #2 ENTER
- Element #3 ENTER
- Element #3 EXIT
- Element #2 EXIT
- Element #1 EXIT
- Element #1 ENTER
- …

Algorithm

- Read ENTER / EXIT events one by one
- Detect & add new Elements and Threads to DM

For ENTER events:

- Log thread time we were upstack
- Log element enters

For EXIT events

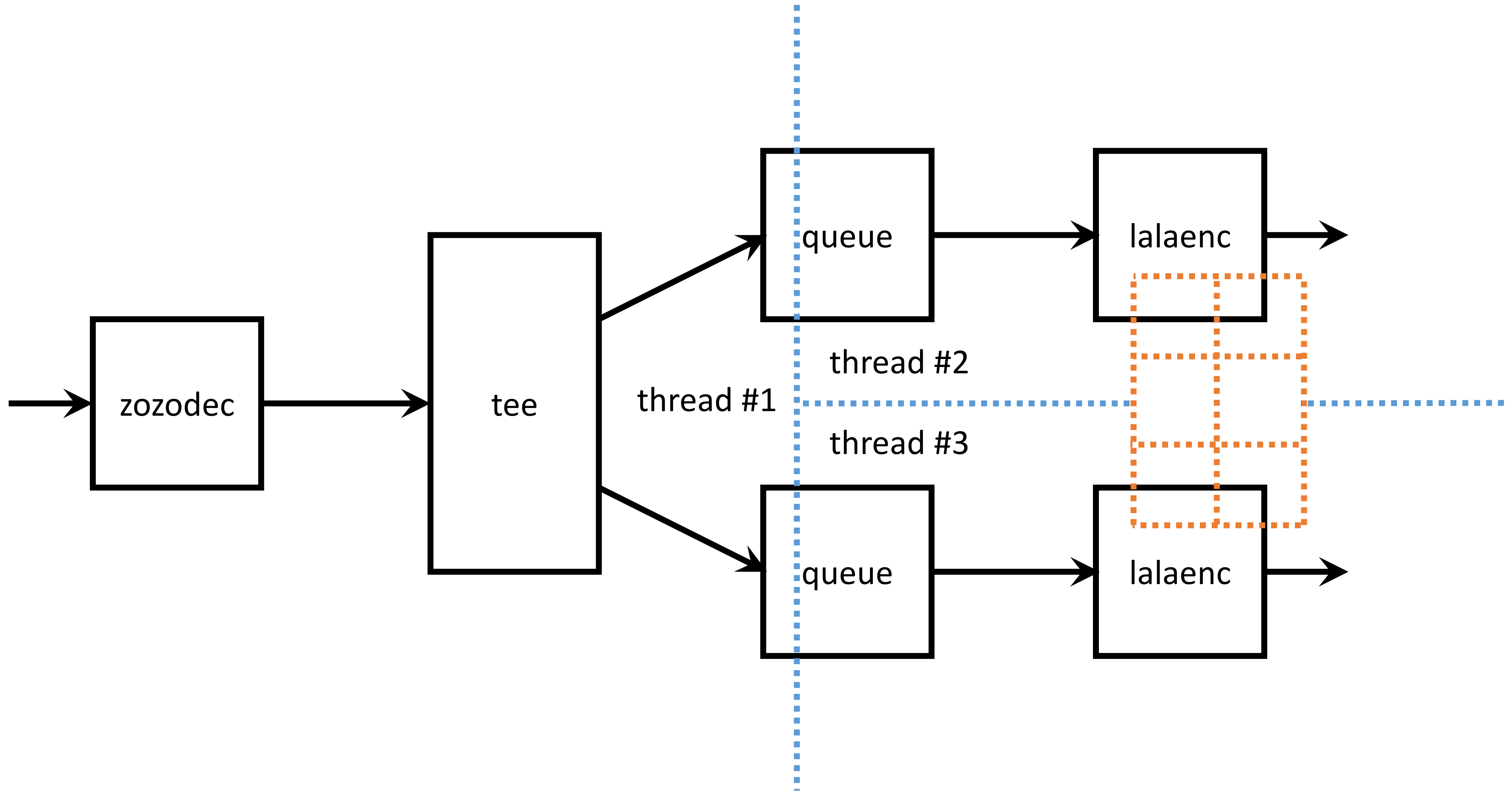
- Find corresponding ENTER
- Log time we were in element and subtract downstack time

Threads outside GThreadPool & GstTask

- Wrap thread creation
- Assign created threads to corresponding elements
- When pushing / pulling, take a look on execution time of each thread assigned to element

Third-party thread pools

Threads' realms



What new since 1.6?

- Tracing subsystem integrated
- .DYLD interpose implemented
- Trace format switched to binary

Todos:

- Measuring CPU time taken by non-GTasked threads

Thank you!

Any questions?