

A

# Practical Perfetto Introduction for AOSP and Linux Developers

INNOVATE. INTEGRATE. EXCEED.



inovex



# Hello,

I'm Stefan Lengfeld

Embedded Software Developer, Germany

- Embedded Systems (Linux and Android)
- Linux Kernel
- Build systems
- Linux and Android Graphics Stack
- Performance and Tracing

 stefan.lengfeld@inovex.de

 stefan@lengfeld.xyz

 @stefan-lengfeld

 @lengfeld

 stefan.lengfeld.xyz

This is my third presentation  
about tracing with [Perfetto](#) (and [systrace](#)).

I'm a  
**Perfetto Enthusiast by choice.**

# Perfetto is so cool. Try it out and get help!

Support by Perfetto devs:

- [Mailinglist/Google Group](#)
- [Discord Server](#)

Support by me:

- See my contact details
- Or join the discord on [aosp-devs.org](https://aosp-devs.org) for AOSP/Android system tracing



# Perfetto





## AGENDA

- Introduction
  - a. Android
  - b. Embedded Linux
- Advanced usage  
(Tips & Tricks)
- Outlook



# Introduction

## Why Tracing?

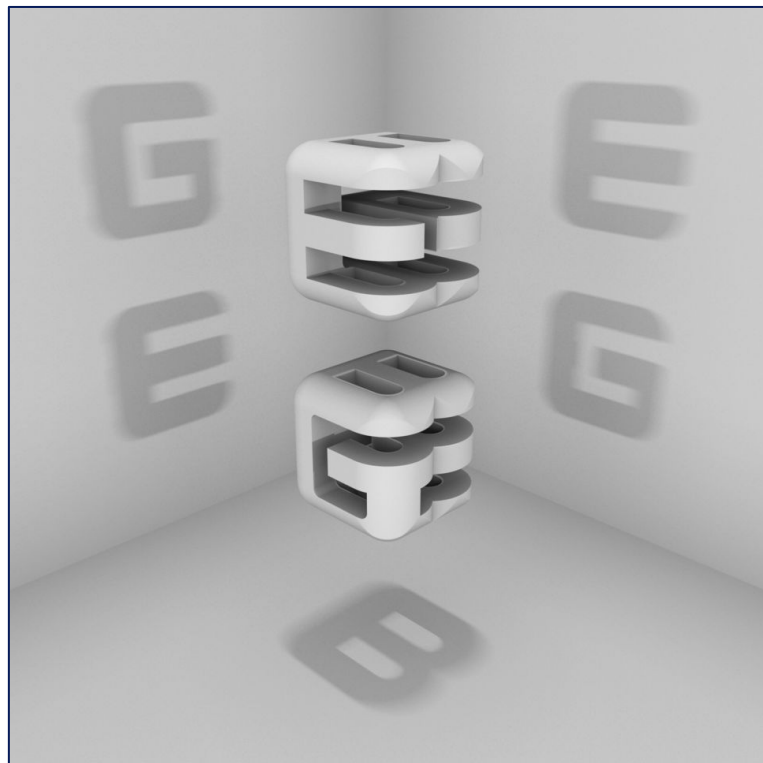
When you only look at the source code, you just see **G**.

But when you trace, you also see

- **E** (the runtime behavior of your code – in-app tracing) and
- **B** (the runtime behavior of the other parts of the system – system-wide tracing)

It this avoids:

*[...] premature optimization is the root of all evil.* [Donald Ervin Knuth](#)

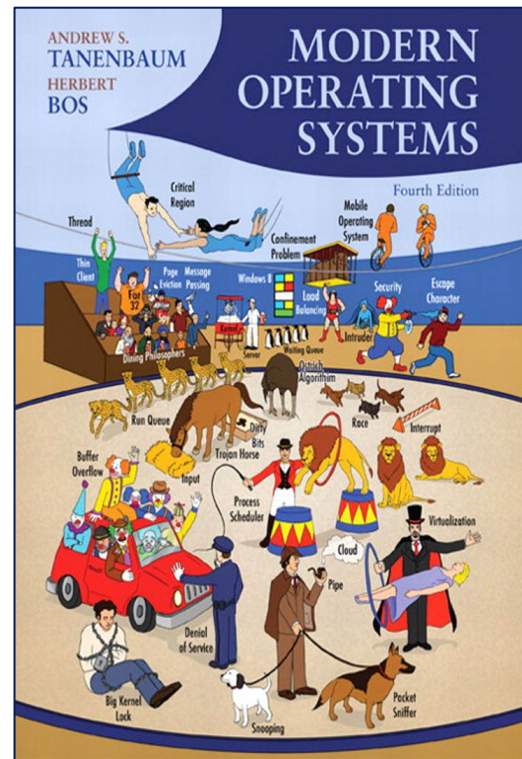
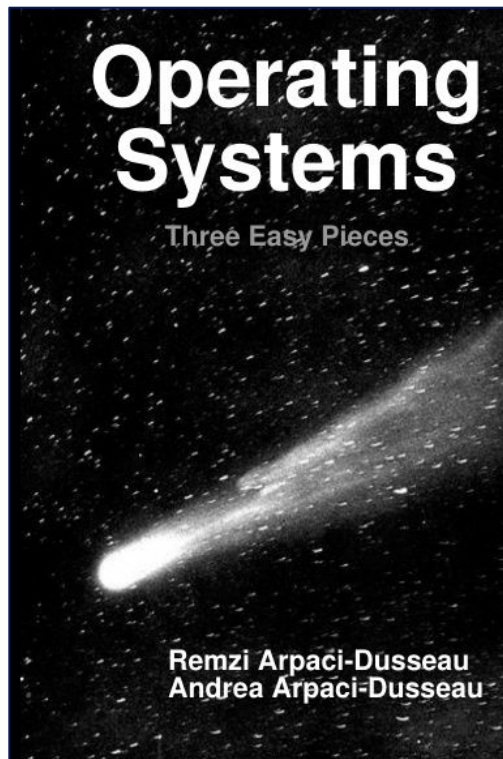


## Prerequisites

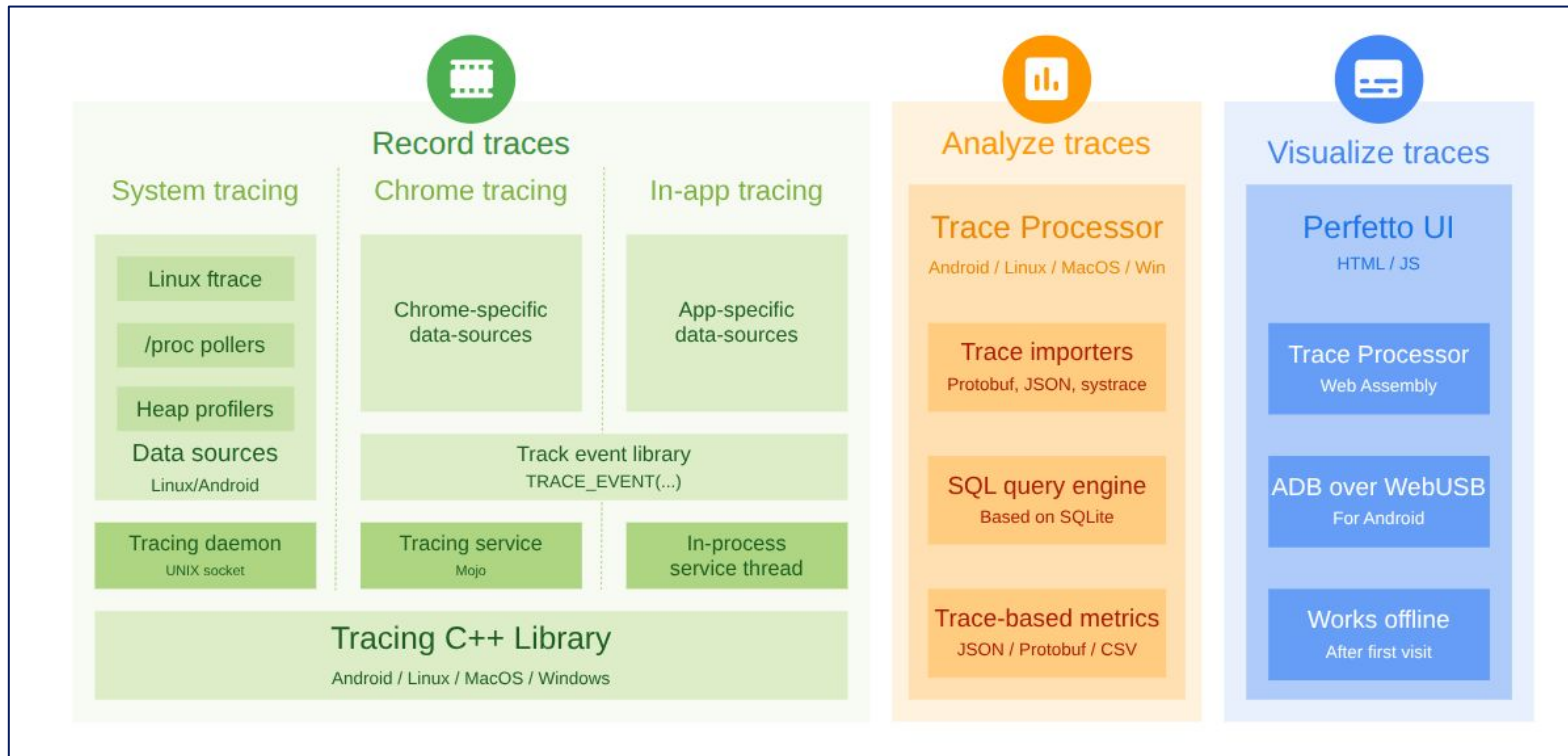
You need to know some OS fundamentals:

- processes, threads
- thread states
- system calls
- scheduling
- (virtual) memory
- (call) stacks

At least a bit!



# Perfetto Overview



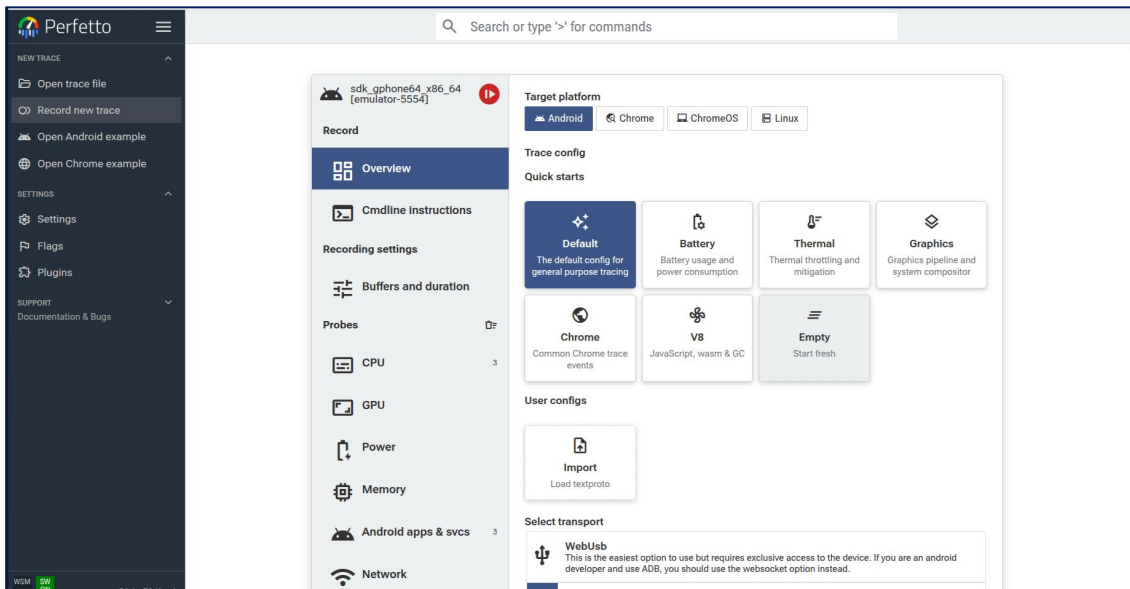
# Tracing on Android

There is **plenty** of documentation and it's **well** integrated.

- 1) Start your Android device and connect via adb
- 2) Go to [ui.perfetto.dev](https://ui.perfetto.dev) and select “Record a new Trace”
- 3) Use WebUSB or start “./tracebox websocket\_bridge”

See also

- [Performance: Perfetto Traceviewer - MAD Skills](#)
- [Debug Android Performance with Tracing & Perfetto](#)



## Tracing on Embedded Linux (e.g. Yocto)

There is **not plenty** of docu and it's **not well** integrated.

Therefore I have created an example and tutorial:




<https://codeberg.org/lengfeld/perfetto-yocto-tutorial>

# Advanced usage (Tips & Tricks)

# Basic Tips

- Gamer keys
- Search for
  - tracks
  - slices
  - logs

🔍 Search or type '>' for commands



# Perfetto

QUICK START

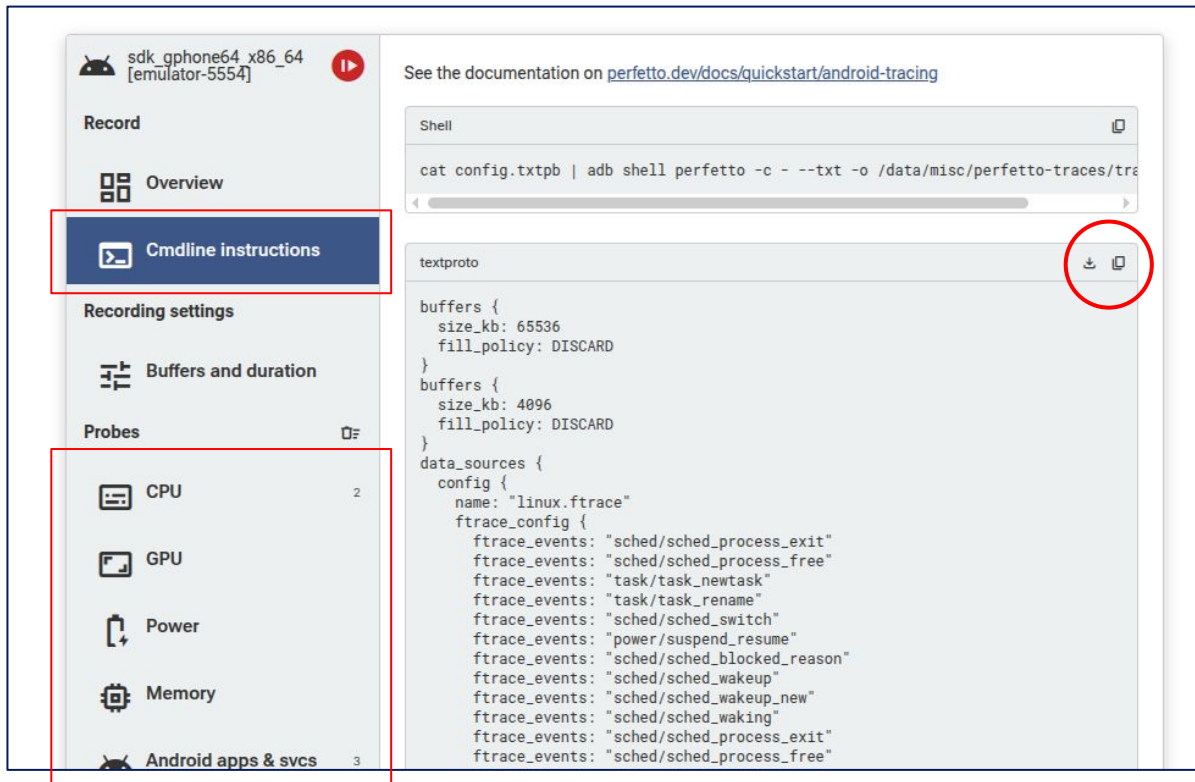
- 📁 Open trace
- 🎙 Record new trace

SHORTCUTS

Find tracks	<div style="display: inline-block; border: 1px solid #ccc; padding: 2px 5px; margin-right: 2px;">Ctrl</div> <div style="display: inline-block; border: 1px solid #ccc; padding: 2px 5px;">P</div>
Navigate timeline	<div style="display: inline-block; border: 1px solid #ccc; padding: 2px 5px; margin-right: 2px;">W</div> <div style="display: inline-block; border: 1px solid #ccc; padding: 2px 5px; margin-right: 2px;">A</div> <div style="display: inline-block; border: 1px solid #ccc; padding: 2px 5px; margin-right: 2px;">S</div> <div style="display: inline-block; border: 1px solid #ccc; padding: 2px 5px;">D</div>
Commands	<div style="display: inline-block; border: 1px solid #ccc; padding: 2px 5px; margin-right: 2px;">Ctrl</div> <div style="display: inline-block; border: 1px solid #ccc; padding: 2px 5px; margin-right: 2px;">↑</div> <div style="display: inline-block; border: 1px solid #ccc; padding: 2px 5px;">P</div>

# Use the TraceViewer as your config config manager

- 1) Explore the data sources in the Trace Viewer
- 2) Copy into a file for manual tracing



The screenshot displays the Android Trace Viewer interface. On the left, a sidebar contains several sections: 'Record' with a play button, 'Overview' with a grid icon, 'Cmdline instructions' with a terminal icon (highlighted with a red box), 'Recording settings' with a list icon, and 'Probes' with a list icon. Under 'Probes', there are five categories: 'CPU' (2), 'GPU', 'Power', 'Memory', and 'Android apps & svcs' (3), all of which are highlighted with a red box. The main area on the right shows a terminal window with the command `cat config.txtpb | adb shell perfetto -c - --txt -o /data/misc/perfetto-traces/tr`. Below the terminal, a 'textproto' section displays a JSON configuration for ftrace events, with a download icon (highlighted with a red circle) in the top right corner.

```
See the documentation on perfetto.dev/docs/quickstart/android-tracing
```

```
Shell
```

```
cat config.txtpb | adb shell perfetto -c - --txt -o /data/misc/perfetto-traces/tr
```

```
textproto
```

```
buffers {
  size_kb: 65536
  fill_policy: DISCARD
}
buffers {
  size_kb: 4096
  fill_policy: DISCARD
}
data_sources {
  config {
    name: "linux.ftrace"
    ftrace_config {
      ftrace_events: "sched/sched_process_exit"
      ftrace_events: "sched/sched_process_free"
      ftrace_events: "task/task_newtask"
      ftrace_events: "task/task_rename"
      ftrace_events: "sched/sched_switch"
      ftrace_events: "power/suspend_resume"
      ftrace_events: "sched/sched_blocked_reason"
      ftrace_events: "sched/sched_wakeup"
      ftrace_events: "sched/sched_wakeup_new"
      ftrace_events: "sched/sched_waking"
      ftrace_events: "sched/sched_process_exit"
      ftrace_events: "sched/sched_process_free"
    }
  }
}
```

# Built-in Histogram and statistics

1. Click on Slice
2. Then click on slice name
3. Select “with same name”

Current Selection

Slice **Compiling optimized**

Details

<b>Name</b>	<u>Compiling optimized</u>
<b>Category</b>	Slices with the same name (this track)
<b>Start</b>	Slices with the same name (across trace) 7
<b>Absolute</b>	
<b>Duration</b>	<u>1μs 783ns</u>
<b>Running</b>	<u>1μs 783ns</u> (100.00%)
<b>Thread</b>	<u>Jit thread pool [15950]</u>

Current Selection Compiling optimized (this track) X

Compiling optimized (this track) slice Add debug track

Instances

id	ts	dur ↓
<a href="#">4100</a>	<a href="#">00:00:03.345 968 069</a>	<a href="#">67μs 77ns</a>
<a href="#">3421</a>	<a href="#">00:00:03.028 094 618</a>	<a href="#">59μs 803ns</a>
<a href="#">7171</a>	<a href="#">00:00:05.313 964 067</a>	<a href="#">49μs 564ns</a>
<a href="#">3454</a>	<a href="#">00:00:03.067 000 430</a>	<a href="#">49μs 303ns</a>
<a href="#">12127</a>	<a href="#">00:00:07.703 228 682</a>	<a href="#">44μs 524ns</a>
<a href="#">6999</a>	<a href="#">00:00:05.207 433 778</a>	<a href="#">38μs 914ns</a>
<a href="#">7414</a>	<a href="#">00:00:05.413 073 735</a>	<a href="#">37μs 320ns</a>
<a href="#">3415</a>	<a href="#">00:00:03.022 080 163</a>	<a href="#">36μs 58ns</a>
<a href="#">14264</a>	<a href="#">00:00:09.364 109 145</a>	<a href="#">33μs 633ns</a>
<a href="#">3395</a>	<a href="#">00:00:03.003 754 838</a>	<a href="#">28μs 554ns</a>
<a href="#">9093</a>	<a href="#">00:00:05.757 079 093</a>	<a href="#">26μs 329ns</a>
<a href="#">3399</a>	<a href="#">00:00:03.009 863 019</a>	<a href="#">24μs 426ns</a>
<a href="#">3426</a>	<a href="#">00:00:03.036 809 384</a>	<a href="#">20μs 68ns</a>
<a href="#">3403</a>	<a href="#">00:00:03.015 989 655</a>	<a href="#">19μs 708ns</a>
<a href="#">7834</a>	<a href="#">00:00:05.436 078 021</a>	<a href="#">18μs 465ns</a>
<a href="#">3449</a>	<a href="#">00:00:03.052 410 242</a>	<a href="#">14μs 407ns</a>
<a href="#">12009</a>	<a href="#">00:00:07.632 815 426</a>	<a href="#">13μs 726ns</a>

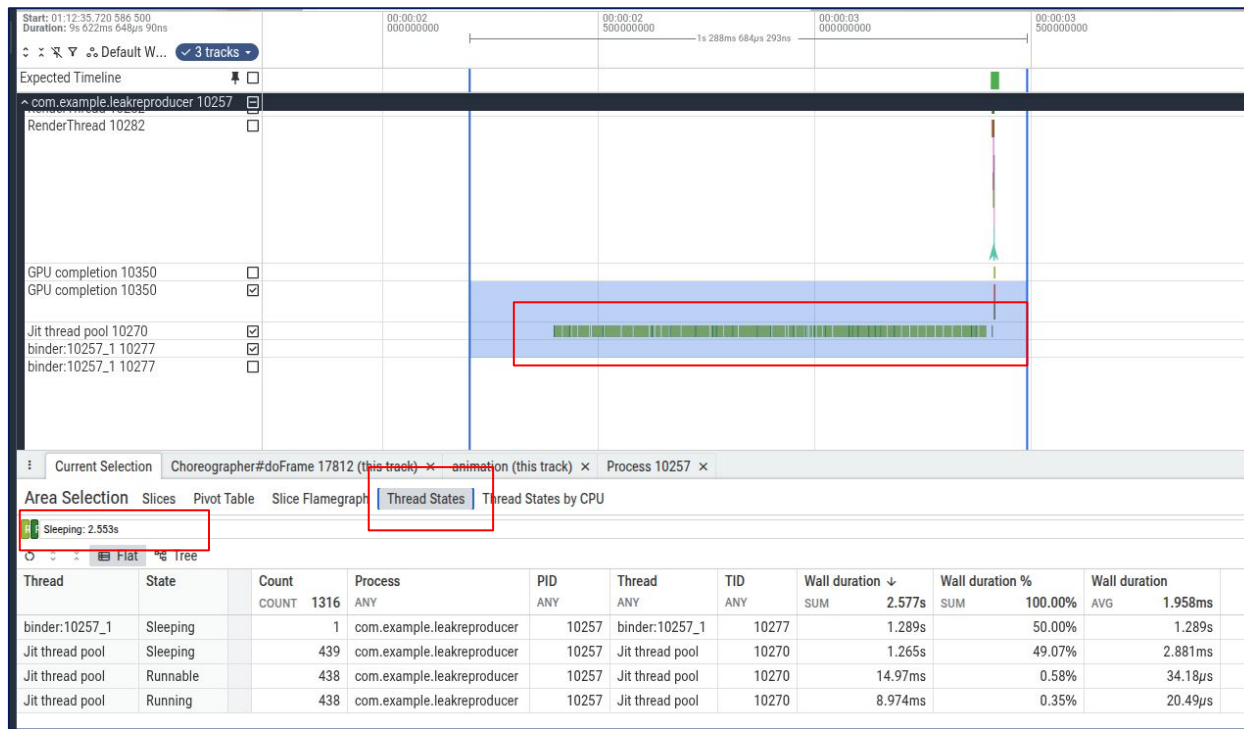
Histogram

<b>Count</b>	1,540	<b>p50</b>	<a href="#">4μs 448ns</a>
<b>Total</b>	<a href="#">7ms 184μs 103ns</a>	<b>p75</b>	<a href="#">5μs 781ns</a>
<b>Min</b>	<a href="#">1μs 503ns</a>	<b>p95</b>	<a href="#">7μs 838ns</a>
<b>Mean</b>	<a href="#">4μs 665ns</a>	<b>p99</b>	<a href="#">14μs 141ns</a>
<b>Max</b>	<a href="#">67μs 77ns</a>	<b>p99.9</b>	<a href="#">54μs 284ns</a>

# Thread states

- 1) Use rectangle select
- 2) Click on “Thread States”
- 3) Shows percentage for states (sleeping, runnable, running, ...)

General advance:  
*Rectangle select* does a lot of useful things!



# CPU tracks

And rectangle select on CPU tracks

- 1) Select a region, e.g. all CPU tracks for a given time
- 2) Select “CPU by process” and sort



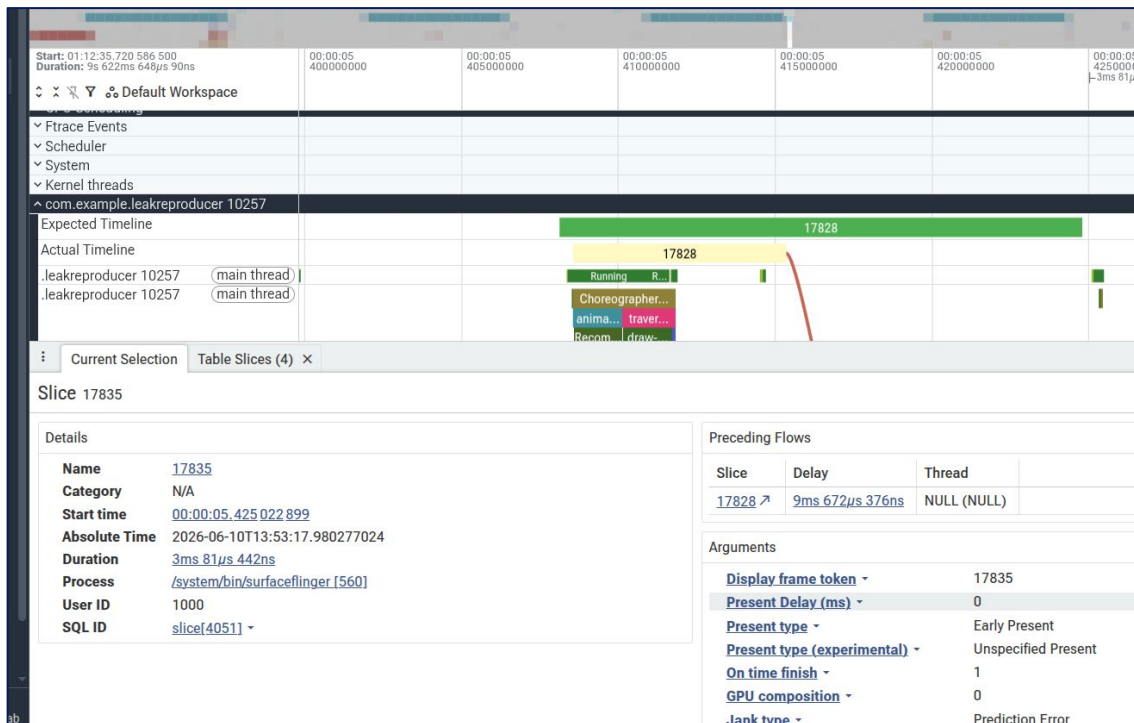
# Android's Frame Timeline

There is the

- Expected Timeline
- Actual Timeline

Every frame has additional information:

- jank type
- ...



# Android's logcat

- Works only with “userdebug builds
- Logs shown in the tab are based on the current view and adapt when moving and zooming

Search or type '>' for commands, ':' for SQL query

Start: 00:57:04.607 565 230  
Duration: 9s 604ms 478µs 357ns

Default Workspace

^ Android logs

- system\_server 1494
- com.android.systemui 4078
- com.android.camera2 5256
- com.android.launcher3 4481
- /apex/com.google.pixel.camera.hal/bi...
- /vendor/bin/aocd 1165
- com.android.permissioncontroller 40...

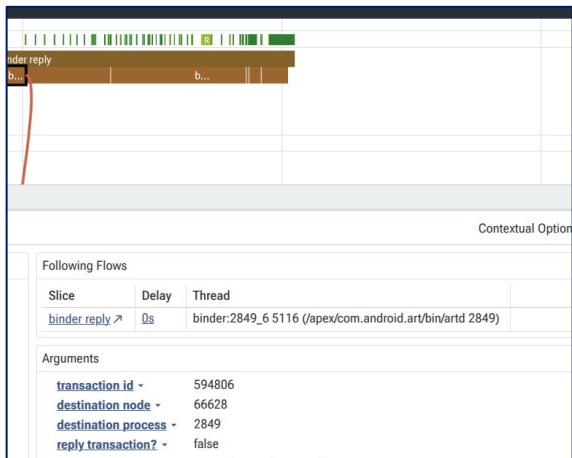
Current Selection: Android Logs x

Android Logs Total messages: 461 Log Level: Verbose Filter by tag... Search logs...

Timestamp	PID	TID	Level	Process	Tag
00:00:00.491881551	5256	6179	D	com.android.camera2	ReflectedParamUpdater
00:00:00.492013468	1165	1165	D	/vendor/bin/aocd	AOC
00:00:00.492189330	1165	1165	D	/vendor/bin/aocd	AOC
00:00:00.492329508	5256	6179	D	com.android.camera2	ReflectedParamUpdater
00:00:00.493465454	946	6181	I	/apex/com.google.pixel.camera.hal/bin/hw/android.hardware...	Lyric

# Binder (and Flows)

- Click on slice for Flow information

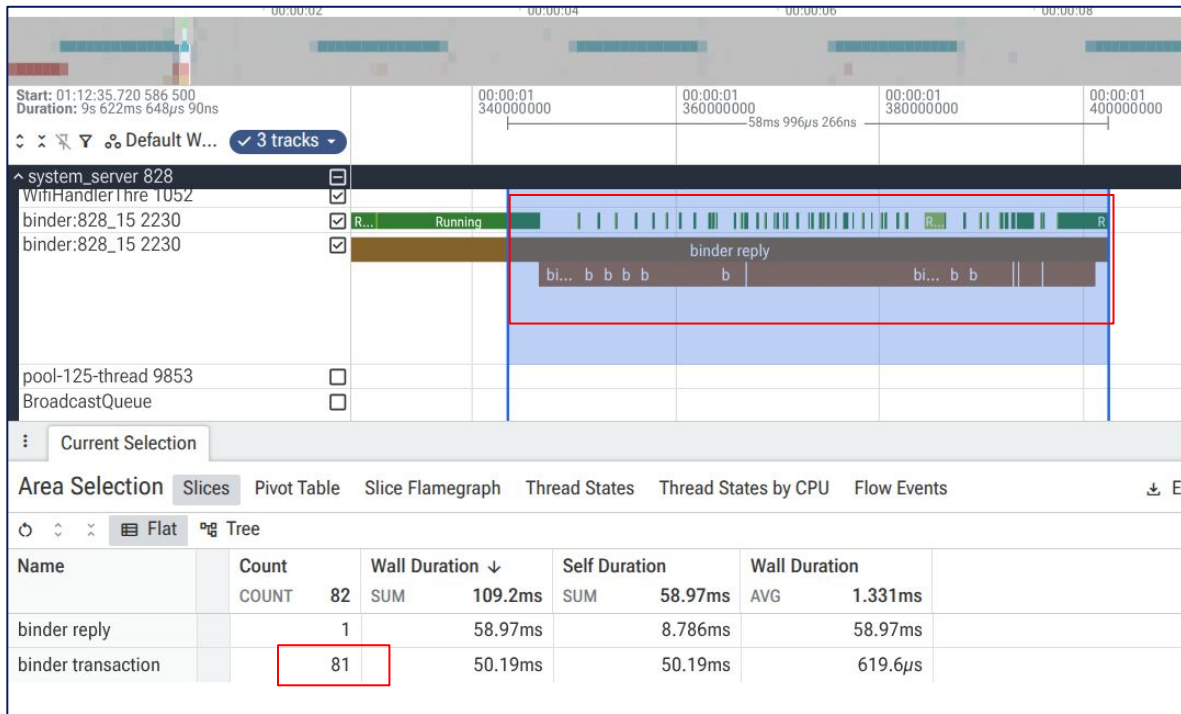


Following Flows

Slice	Delay	Thread
<a href="#">binder_reply</a>	0s	binder:2849_6_5116 (/apex/com.android.art/bin/artd 2849)

Arguments

- transaction\_id - 594806
- destination\_node - 66628
- destination\_process - 2849
- reply\_transaction? - false



Name	Count	Wall Duration ↓	Self Duration	Wall Duration
	COUNT	SUM	SUM	AVG
binder reply	1	58.97ms	8.786ms	58.97ms
binder transaction	81	50.19ms	50.19ms	619.6μs

- Rectangle select to see statistics

# Perfetto SQL

- Look out for “Copy SQL Query” in the context menus
- Use Tables tab as Quick Reference
- Use Format button

The screenshot shows the Perfetto SQL interface. At the top, there is a search bar with the text "Search or type '>' for commands, ':' for SQL query". Below this, the "Query 1" tab is active, showing a SQL query. A red box highlights the "Format" button in the top right of the query editor. Below the query, the results are displayed as a table with 3 rows and 3 columns: "name", "cmdline", and "min\_dur\_us".

Returned 3 rows in 4 ms ⊞ Add debug track ⌵ Export

name	cmdline	min_dur_us
/usr/bin/python3	/usr/bin/python3 /usr/bin/example-python 13	
example-cpp	example-cpp /dev/i2c-13	
/usr/bin/python3	/usr/bin/python3 /usr/bin/example-python 13	

On the right side, the "Tables" tab is active, showing a list of tables. A red box highlights the "Tables" tab. The list includes:

- chrome\_loadline2\_stages
- chrome\_jetstream\_3\_measure
- chrome\_jetstream\_3\_benchmark\_score
- chrome\_inputs
- chrome\_input\_pipeline\_steps
- chrome\_coalesced\_inputs
- chrome\_touch\_move\_to\_scroll\_update
- chrome\_dispatch\_android\_input\_event\_to\_touch\_mo...
- chrome\_scroll\_frame\_info\_v4
- chrome\_speedometer\_2\_1\_measure
- chrome\_speedometer\_2\_1\_iteration
- chrome\_event\_latencies
- chrome\_gesture\_scroll\_updates
- chrome\_java\_views
- chrome\_scheduler\_tasks
- chrome\_tasks
- chrome\_scroll\_update\_refs

# Stack Sampling

- 1) Enable stack sample in Trace Viewer UI
- 2) Either click on one trace sample or use rectangle select
- 3) See the flame graph at the bottom to pinpoint expensive functions

Callstack sampling

Periodically records the current callstack (chain of function calls) of processes.

Sampling frequency  100 Hz

The screenshot shows the Trace Viewer interface. The top section displays a list of tracks for the process `com.example.leakreproducer`. The `DefaultDispatch 15971 callstacks cp...` track is selected, and a red rectangle highlights a sample in the trace. Below the trace, the `Perf sample flamegraph` tab is active, showing a flame graph of the sampled callstack. The flame graph is filtered to show `main HF: alloc.*`. The most expensive function in the callstack is `android::uptimeMillis()`, which is highlighted with a red rectangle. Other functions in the callstack include `art_quick_generic_jni_trampoline`, `systemTime`, `clock_gettime`, and `vdso_clock_gettime`.

# Release Notes (on Github and Google groups)

last week

perfetto-automation

v56.0

62048c1

Compare

## Perfetto v56.0

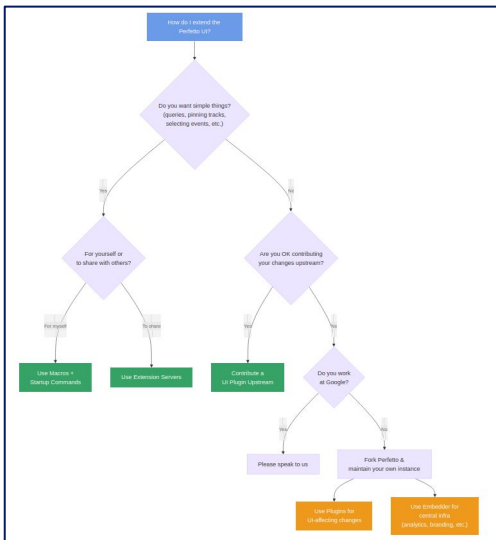
Perfetto v56.0 is out! This release adds more to the Heap Dump Explorer, a round of UI improvements, easier-to-read multi-machine traces, kernel-assisted stack sampling in `traced_perf`, and a few new SDK knobs.

### Heap Dump Explorer: flamegraphs & bitmaps

The Heap Dump Explorer now has a built-in **flamegraph viewer**, so you can jump between the class/object/dominator views and a flamegraph without leaving the page. There's also a new **bitmap metadata page** for bitmaps captured in Android heap dumps, and a feature flag to control whether the explorer **auto-opens** on traces with heap-graph data.

# Extending Perfetto

## Extending the UI



## Custom data sources

(when track events are not enough)

```

traceconv text example_custom_data_source.perfetto-trace
...
packet {
  trusted_uid: 0
  timestamp: 42
  trusted_packet_sequence_id: 2
  previous_packet_dropped: true
  for_testing {
    str: "Hello world!"
  }
}
...
    
```

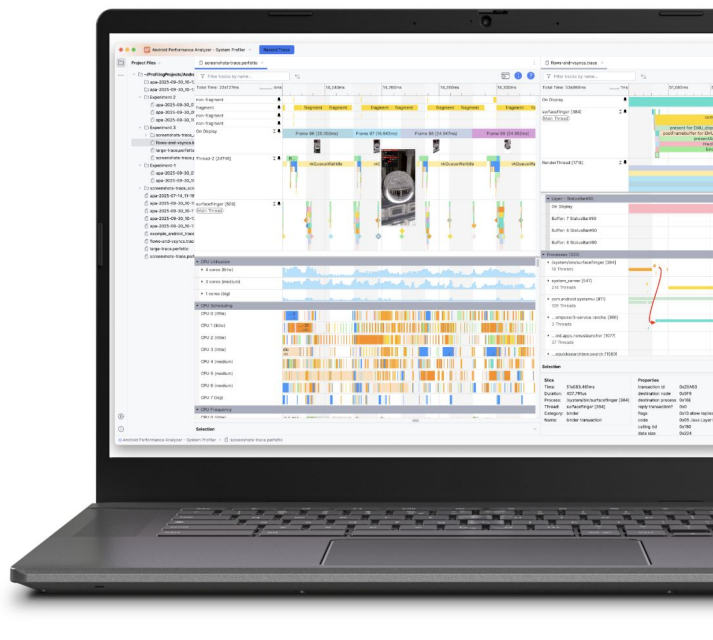
# Outlook

# New: Android Performance Analyzer (APA)

## Android Performance Analyzer

### System wide performance

- GPU, CPU, memory, battery
- Works for all major GPU's
- Built for Vulkan profiling
- Project, folders & tabs based workflow
- Native desktop app for all major OS
- Available as standalone app & in Android Studio



Nearly the end of my presentation

# Thank You! Additional Reading.

And thanks to

- the Perfetto-Team at Google for building Perfetto,
- and all the people working on the Linux kernel performance framework!

[Is Parallel Programming Hard, And, If So, What Can You Do About It?](#)

(The perfbook,  
~700 pages)

by Paul E. McKenney

Is Parallel Programming Hard, And, If So,  
What Can You Do About It?

Edited by:

Paul E. McKenney  
Meta Platforms, Inc.  
paulmck@kernel.org

June 12, 2026  
Commit: v2025.12.18a-106-g5d674b92

# Perfetto is so cool. Try it out and get help!

Support by Perfetto devs:

- [Mailinglist/Google Group](#)
- [Discord Server](#)

Support by me:

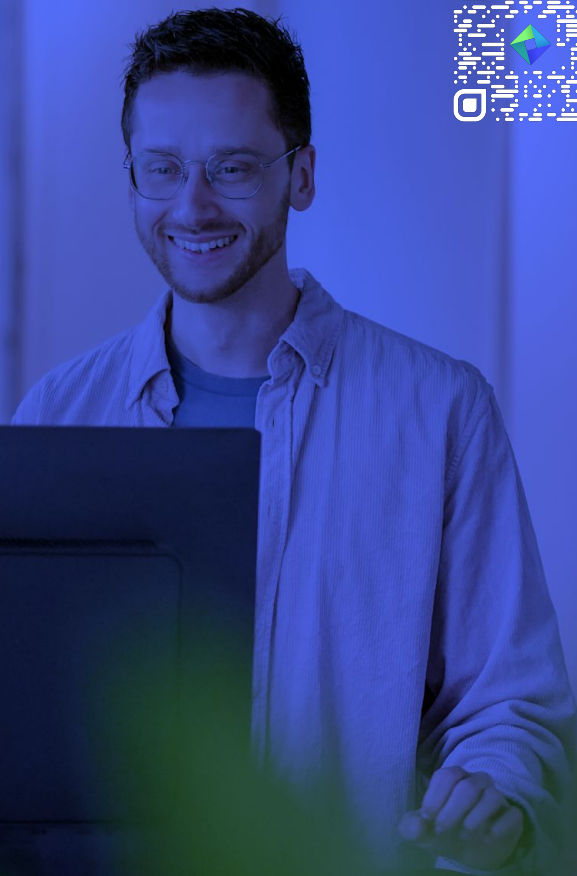
- See my contact details
- Or join the discord on [aosp-devs.org](https://aosp-devs.org) for AOSP/Android system tracing



# Perfetto



# Thank you!



**Stefan Lengfeld**

Perfetto Enthusiast by choice



 [stefan.lengfeld@inovex.de](mailto:stefan.lengfeld@inovex.de)