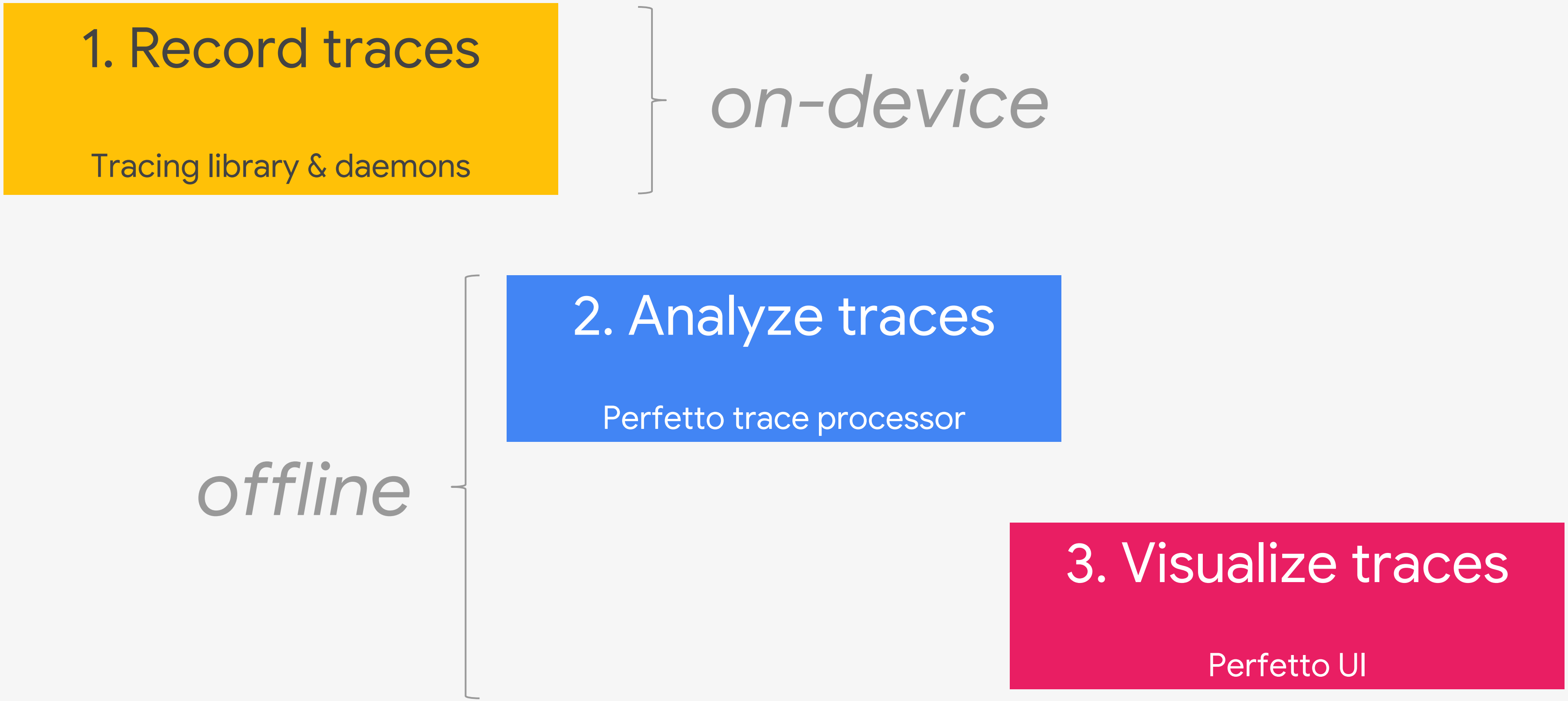
Perfetto

Platform-wide performance instrumentation and tracing for Android and Chrome

Tracing Summit 2018 - Edinburgh

primiano@google.com

What is Perfetto about?



1. Record traces

Tracing library & daemons

on-device

2. Analyze traces

Perfetto trace processor

offline

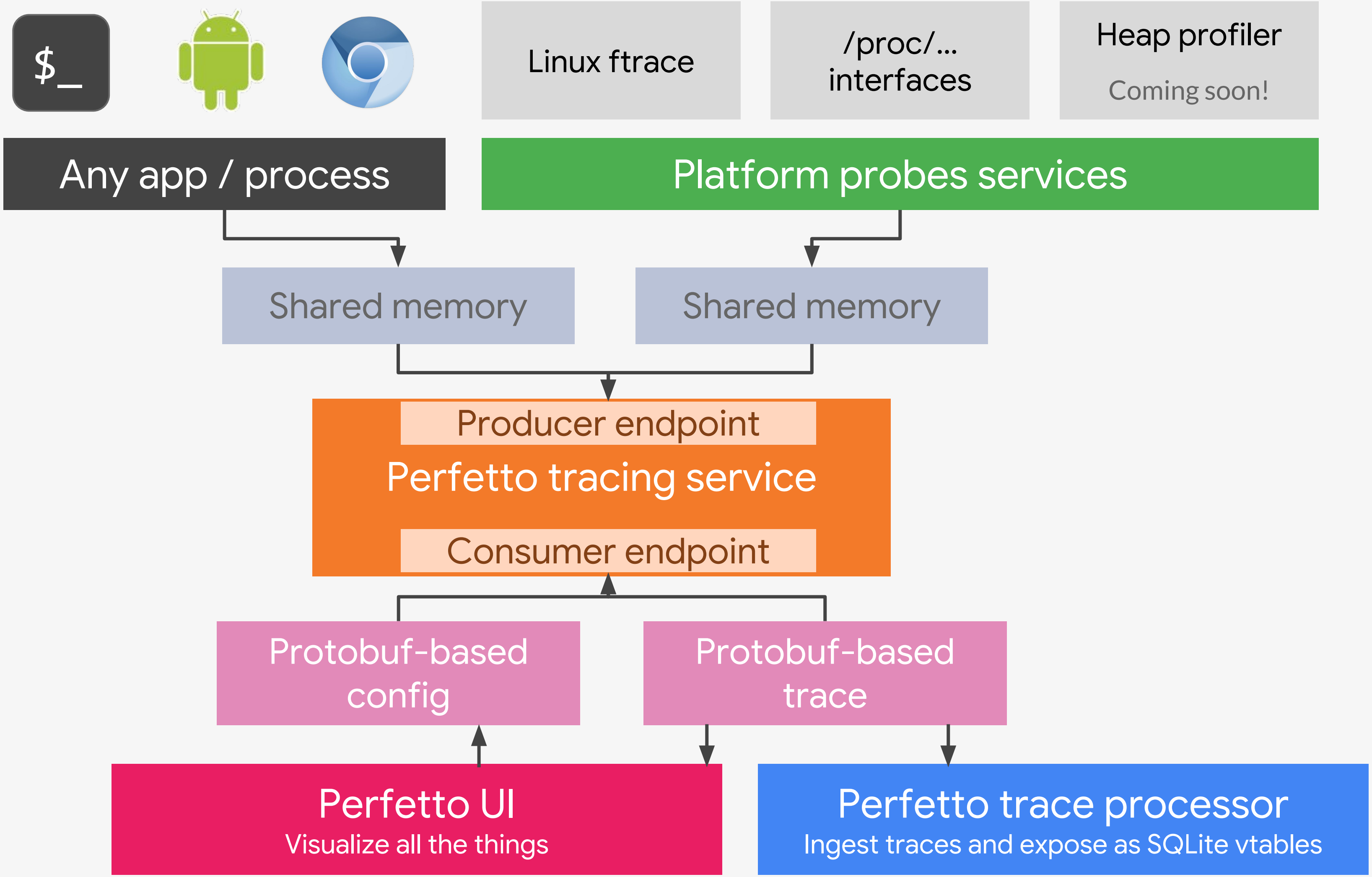
3. Visualize traces

Perfetto UI

What is Perfetto about?

1. An open source (AOSP / Apache2 license) project for recording, processing and visualizing traces.
2. A production C++11 codebase for secure and efficient (zero-copy*, zero-malloc*) userspace-to-userspace tracing.
3. Integration with ftrace, /proc/{stat,vmstat,pid/*} and soon perf_event_open.
4. A SQLite-based codebase for analyzing and processing traces.
5. A UI frontend.

* Some copies / allocations are involved, once every ~4KB.



$_

Linux ftrace

/proc/... interfaces

Heap profiler

Coming soon!

Any app / process

Platform probes services

Shared memory

Shared memory

Producer endpoint

Perfetto tracing service

Consumer endpoint

Protobuf-based config

Protobuf-based trace

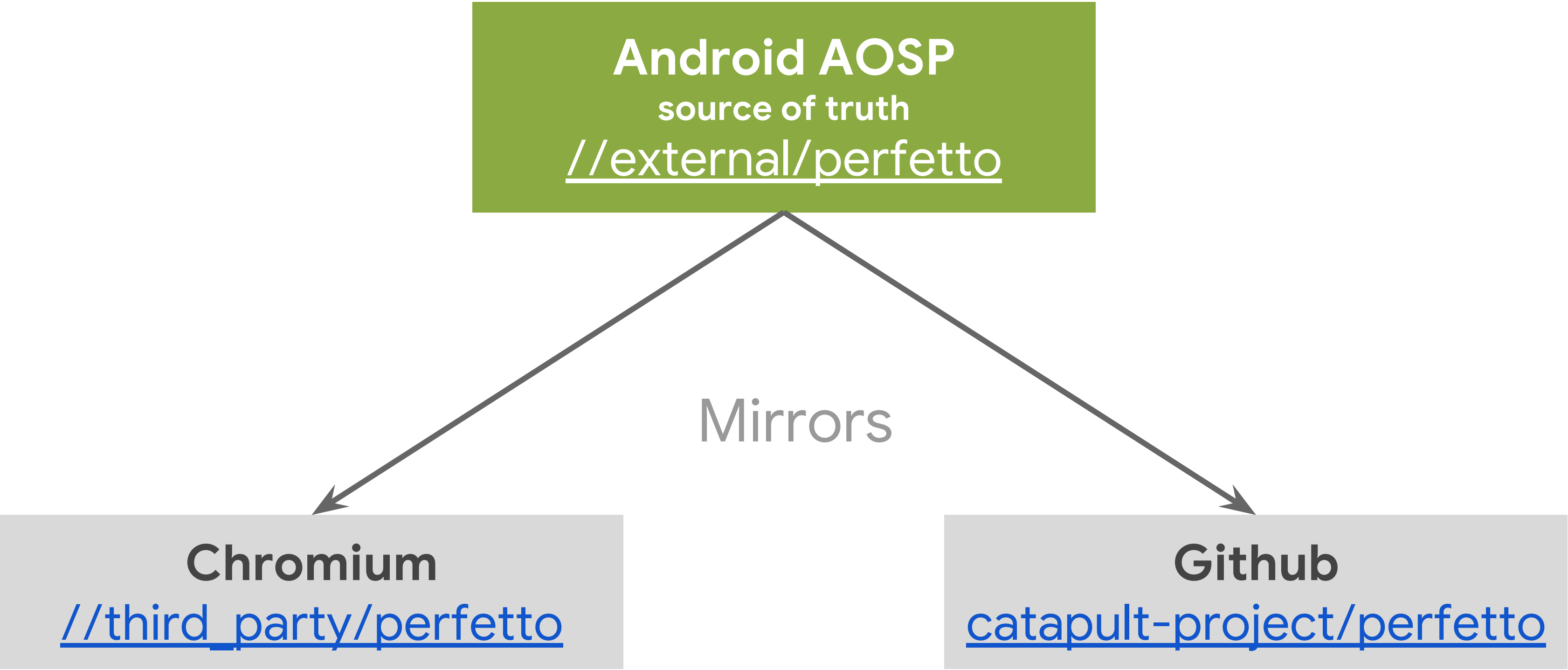
Perfetto UI

Visualize all the things

Perfetto trace processor

Ingest traces and expose as SQLite vtables

Where to find the code?



Android AOSP

source of truth

//external/perfetto

Mirrors

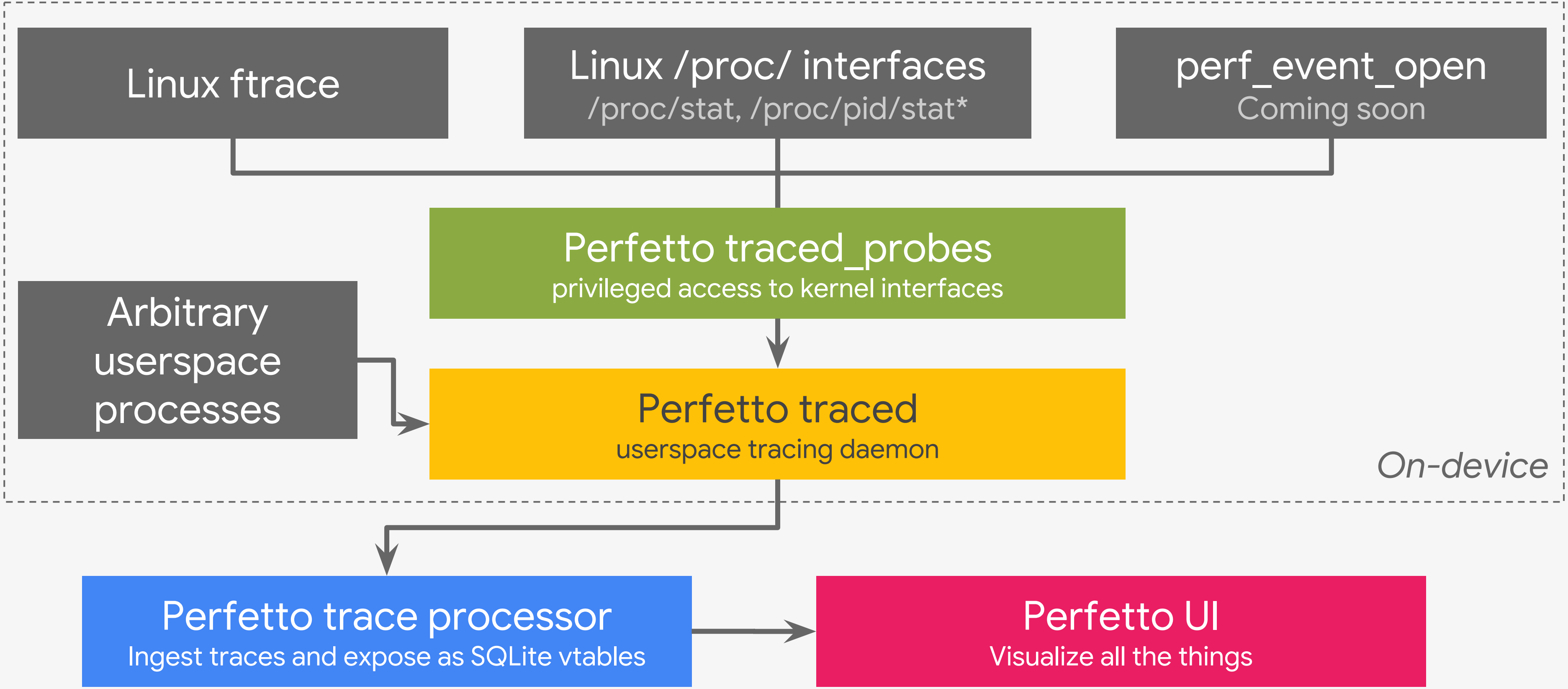
Chromium

//third_party/perfetto

Github

catapult-project/perfetto

What is Perfetto about?



Linux ftrace

Linux /proc/ interfaces
/proc/stat, /proc/pid/stat*

perf_event_open
Coming soon

Perfetto traced_probes
privileged access to kernel interfaces

Arbitrary userspace processes

Perfetto traced
userspace tracing daemon

On-device

Perfetto trace processor
Ingest traces and expose as SQLite vtables

Perfetto UI
Visualize all the things

Userspace tracing library

Key concepts

Producers

- The thing that writes protobufs into the trace buffers
- Untrusted. Potentially malicious. Everything can be a Perfetto producer.
- On startup advertises its capabilities to the tracing service.
- At some point the tracing service asks it to start collecting data

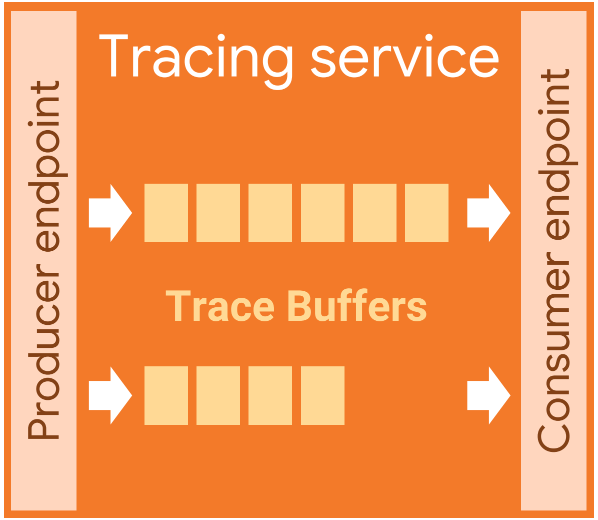
Tracing service

- The thing that owns the log buffers (there is one* buffer for the all system / browser)
- Acts as registry and handles handshakes between producers and consumer(s)
- In chrome: a /services service
- In android: a system service (traced)

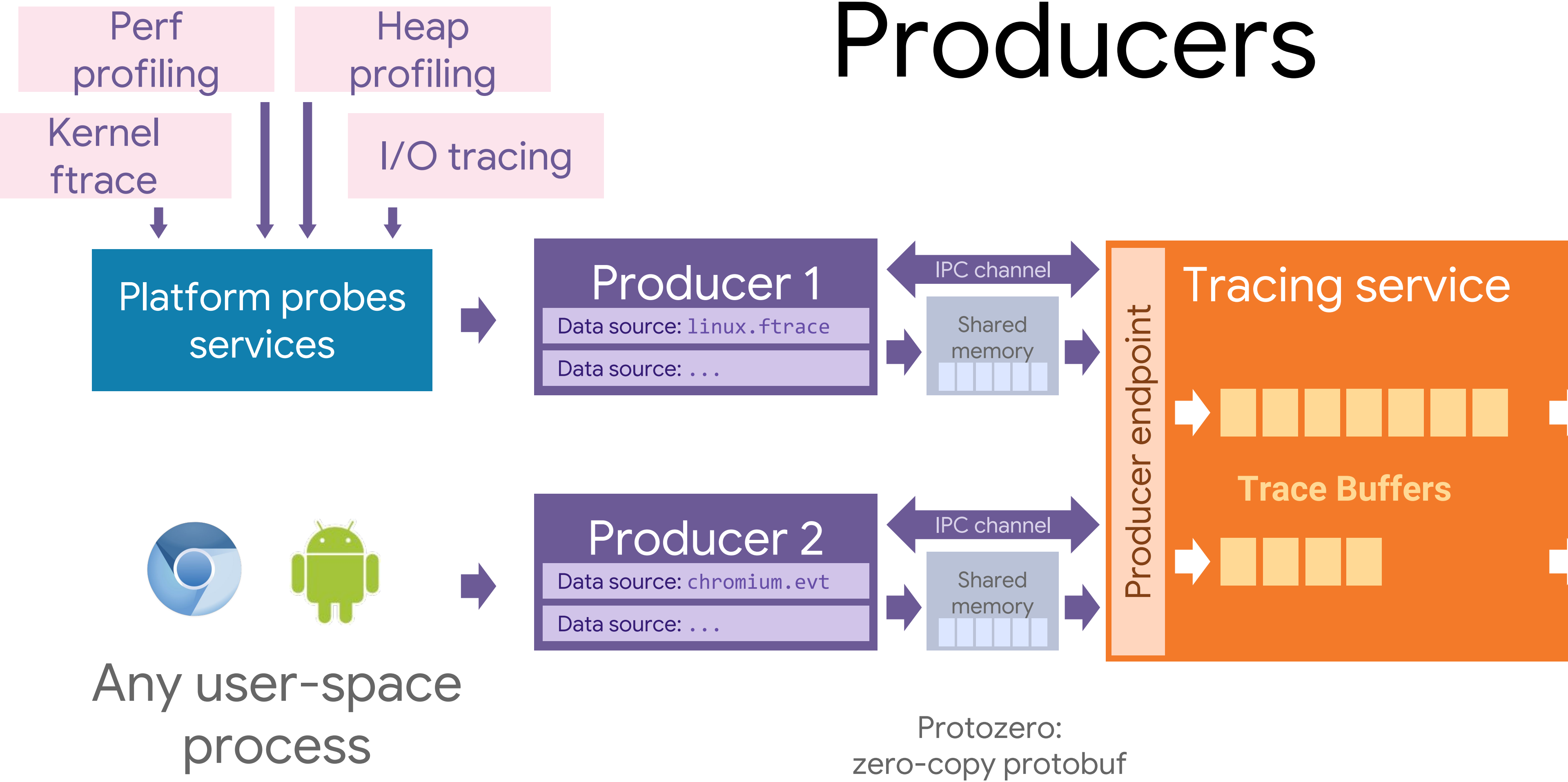
Consumer(s)

- The thing that configures all the tracing session and decides who should trace and what.
- Is allowed to configure the tracing service and read back the trace data
- Trusted / privileged
- In chrome: the thing that exposes data to the UI
- In android: shell (for the UI) and Android Metrics services

Tracing Service



Producers



Perf profiling

Heap profiling

Kernel ftrace

I/O tracing

Platform probes services

Any user-space process

Producer 1

Data source: `linux.ftrace`

Data source: `...`

Producer 2

Data source: `chromium.evt`

Data source: `...`

IPC channel

Shared memory

IPC channel

Shared memory

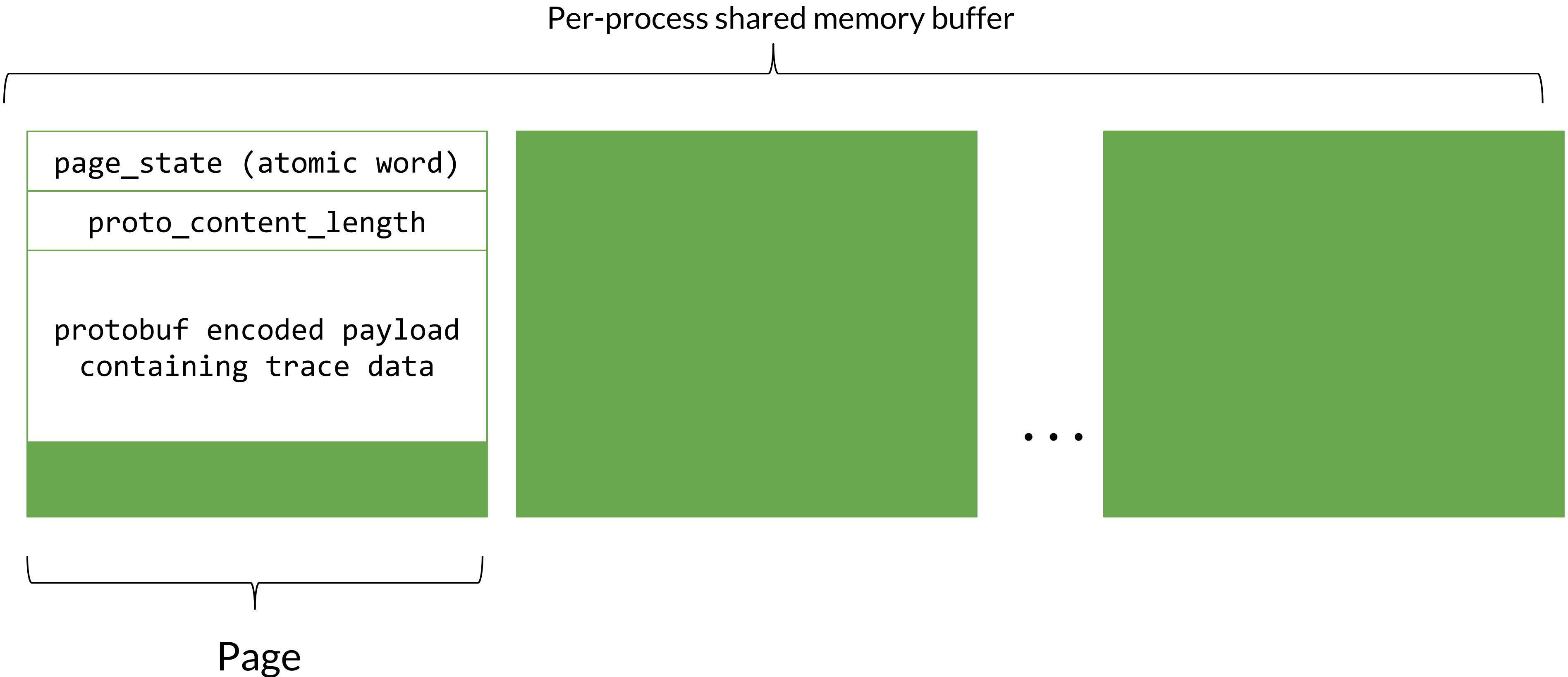
Producer endpoint

Tracing service

Trace Buffers

Protozero:
zero-copy protobuf

Shmem buffer format



Per-process shared memory buffer

page_state (atomic word)

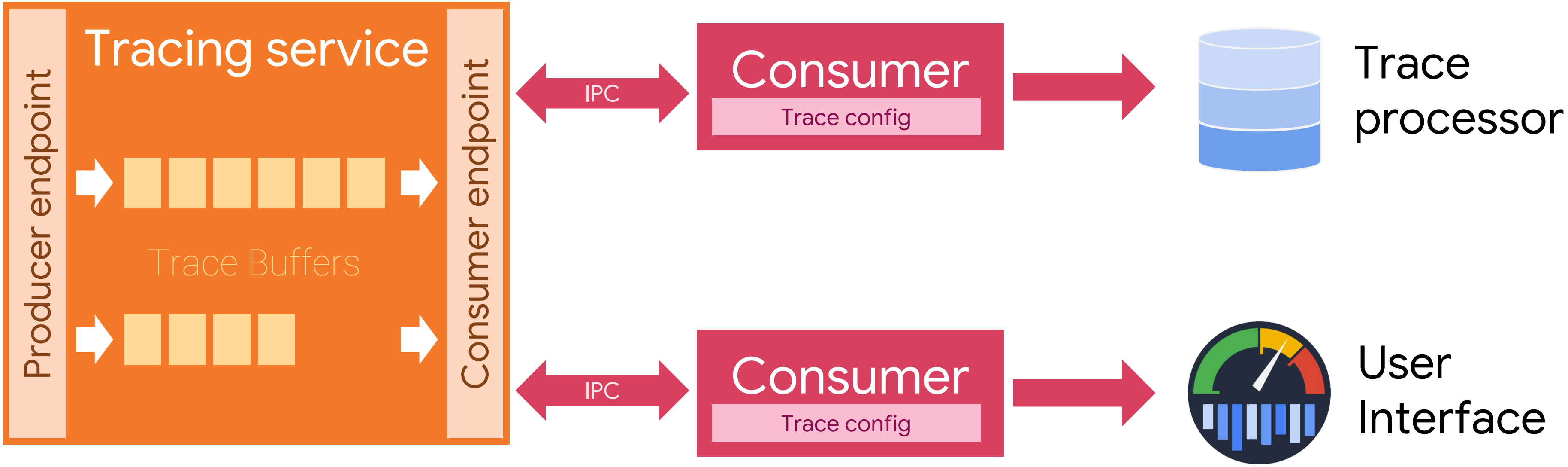
proto_content_length

protobuf encoded payload containing trace data

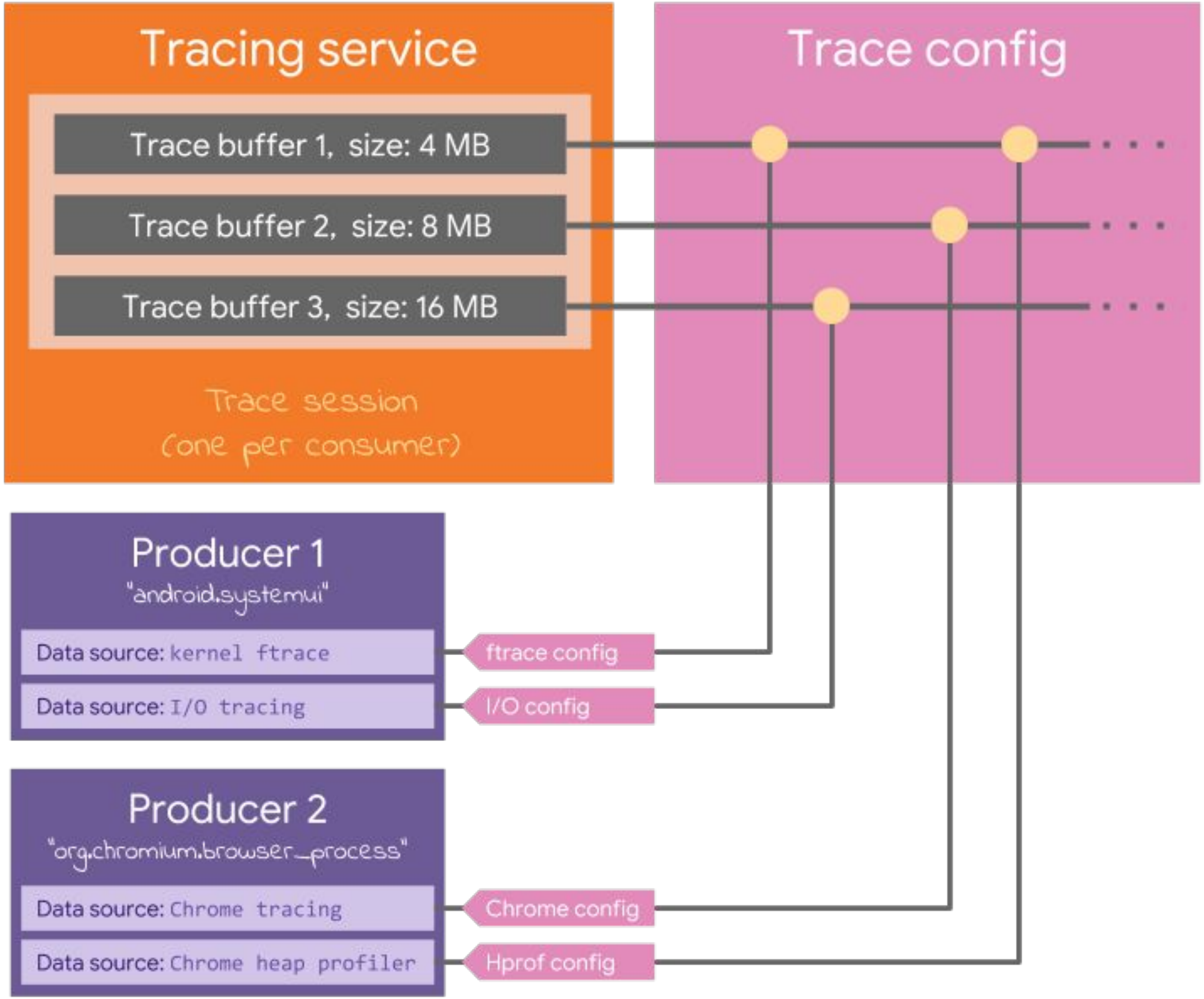
. . .

Page

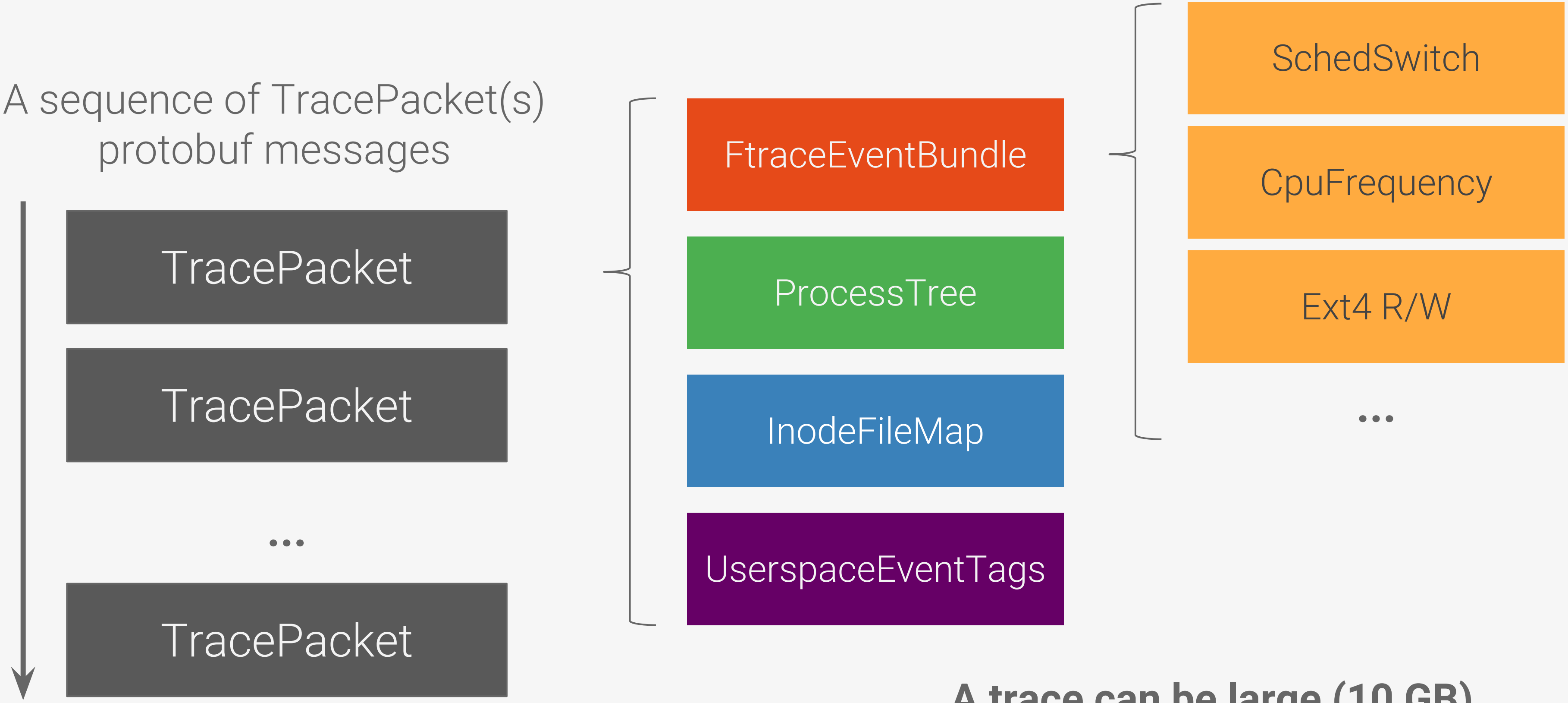
Consumers


Producer endpoint
Tracing service
Trace Buffers
Consumer endpoint
IPC
Consumer
Trace config
Trace
processor
IPC
Consumer
Trace config
User
Interface

Trace config


Tracing service
Trace buffer 1, size: 4 MB
Trace buffer 2, size: 8 MB
Trace buffer 3, size: 16 MB
Trace session
(one per consumer)
Trace config
Producer 1
"android.systemui"
Data source: kernel ftrace
Data source: I/O tracing
ftrace config
I/O config
Producer 2
"org.chromium.browser_process"
Data source: Chrome tracing
Data source: Chrome heap profiler
Chrome config
Hprof config

What is a trace?



A trace can be large (10 GB)

Trace Processor

Trace processor

C++11 + SQLite codebase

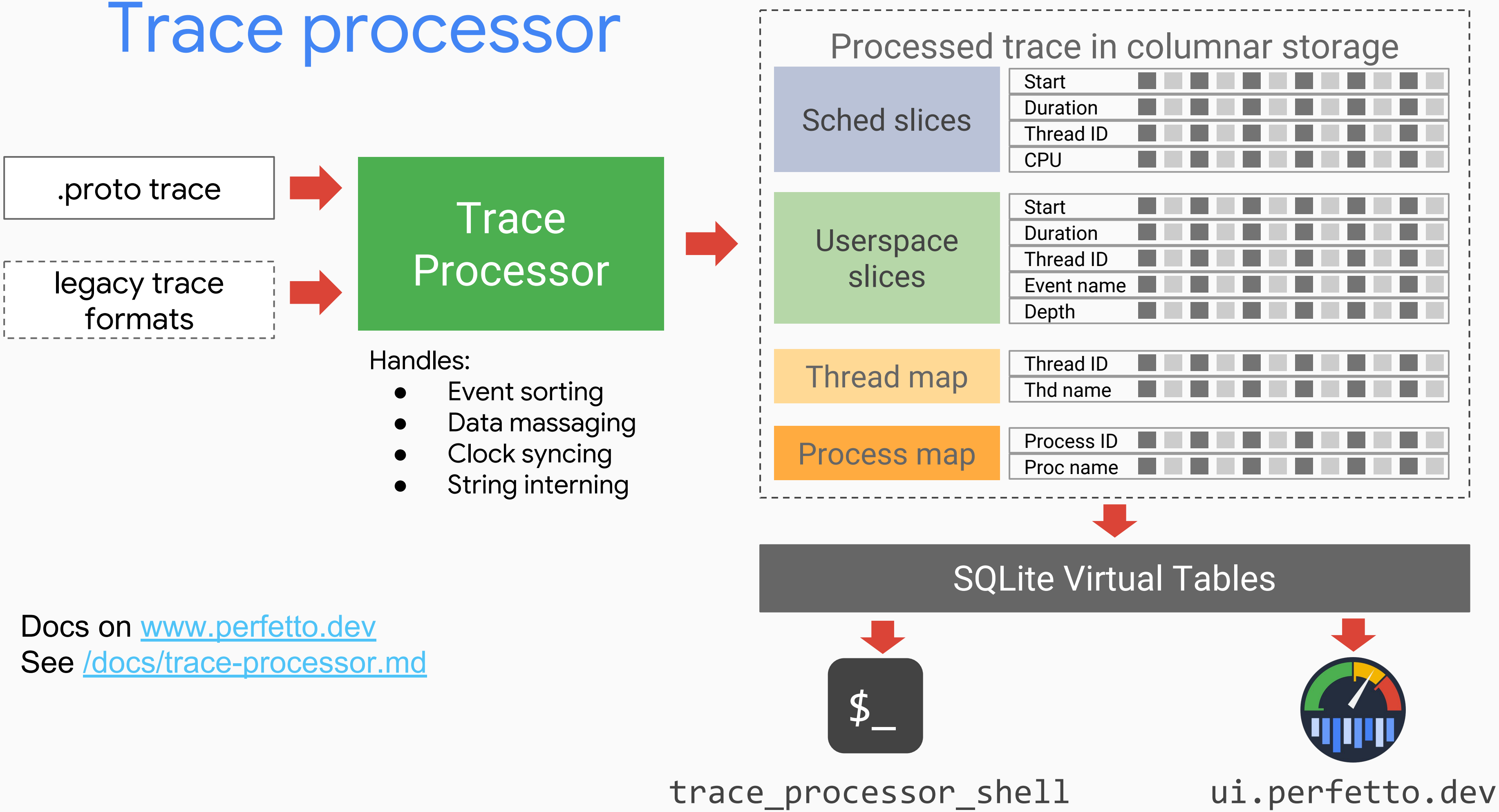
Ingests traces of various formats (for now our .proto and Chrome's JSON, in future also ftrace text)

Builds an in-memory columnar database from trace contents.

Exposes the storage to SQLite through vtable hooks

Adds some trace-specific constructs on top of conventional SQLite ones.

Trace processor



.proto trace

legacy trace formats

Trace Processor

Handles:

- Event sorting
- Data massaging
- Clock syncing
- String interning

Processed trace in columnar storage

Sched slices
- Start
- Duration
- Thread ID
- CPU

Userspace slices
- Start
- Duration
- Thread ID
- Event name
- Depth

Thread map
- Thread ID
- Thd name

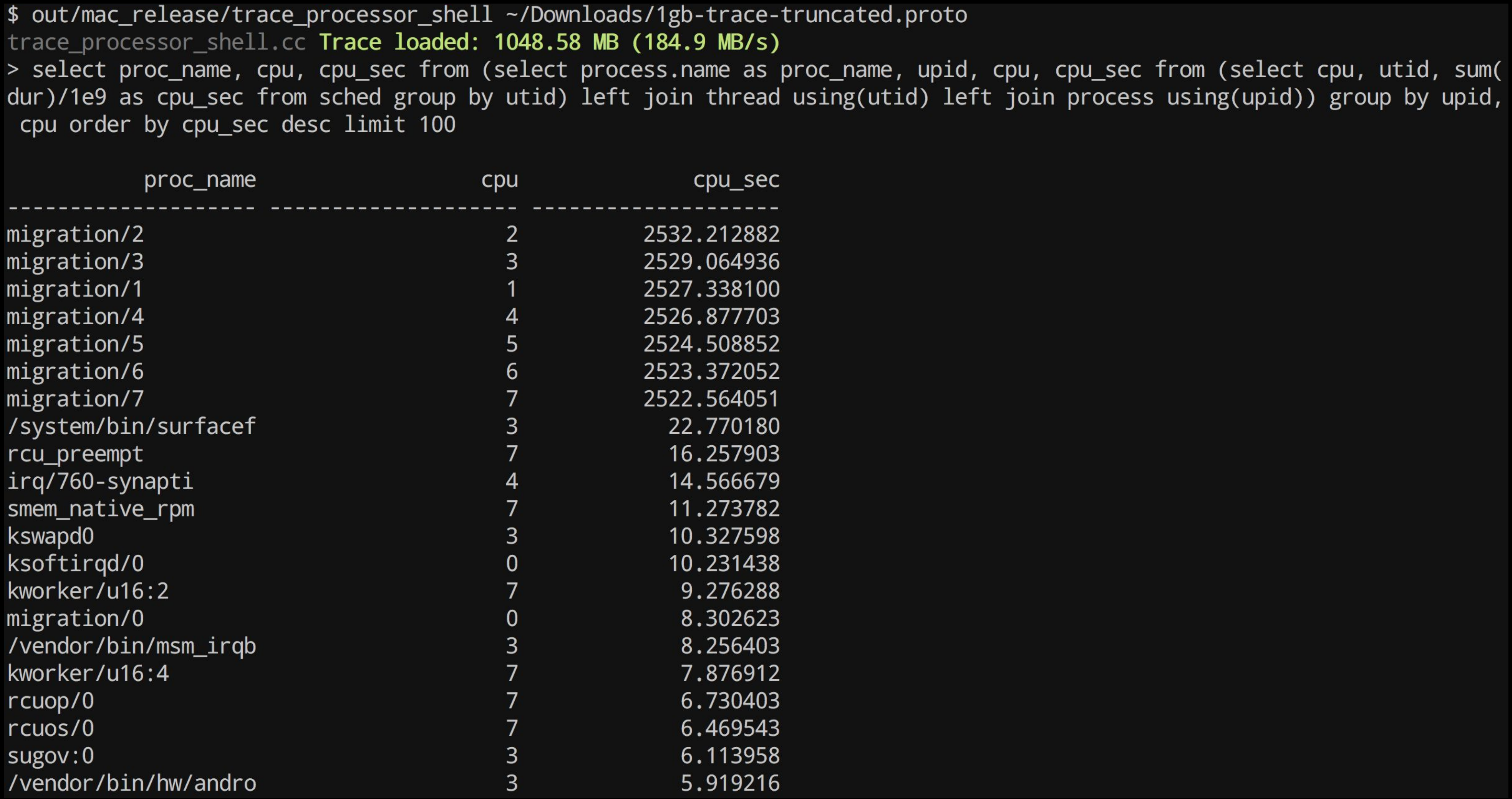
Process map
- Process ID
- Proc name

SQLite Virtual Tables

`trace_processor_shell`

`ui.perfetto.dev`

Docs on www.perfetto.dev
See /docs/trace-processor.md

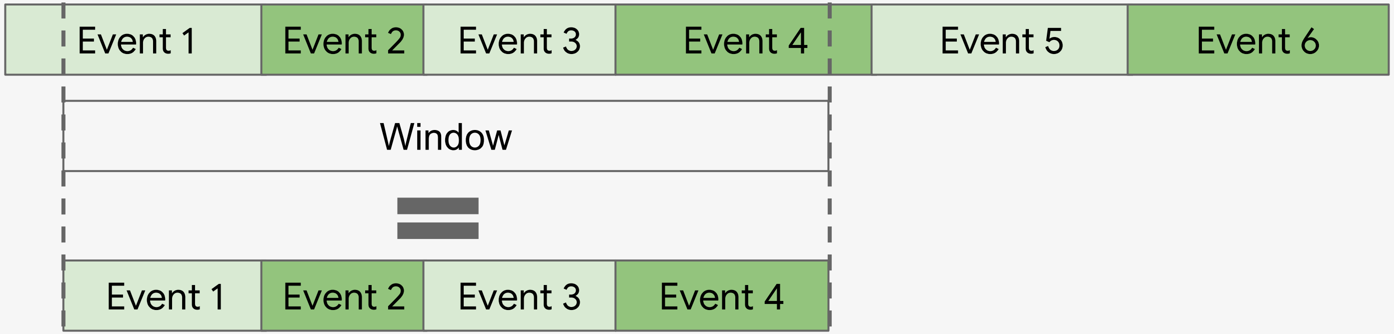


```
$ out/mac_release/trace_processor_shell ~/Downloads/1gb-trace-truncated.proto
trace_processor_shell.cc Trace loaded: 1048.58 MB (184.9 MB/s)
> select proc_name, cpu, cpu_sec from (select process.name as proc_name, upid, cpu, cpu_sec from (select cpu, utid, sum(
dur)/1e9 as cpu_sec from sched group by utid) left join thread using(utid) left join process using(upid)) group by upid,
 cpu order by cpu_sec desc limit 100

           proc_name                  cpu              cpu_sec
-------------------- -------------------- --------------------
migration/2                             2          2532.212882
migration/3                             3          2529.064936
migration/1                             1          2527.338100
migration/4                             4          2526.877703
migration/5                             5          2524.508852
migration/6                             6          2523.372052
migration/7                             7          2522.564051
/system/bin/surfacef                    3            22.770180
rcu_preempt                             7            16.257903
irq/760-synapti                         4            14.566679
smem_native_rpm                         7            11.273782
kswapd0                                 3            10.327598
ksoftirqd/0                             0            10.231438
kworker/u16:2                           7             9.276288
migration/0                             0             8.302623
/vendor/bin/msm_irqb                    3             8.256403
kworker/u16:4                           7             7.876912
rcuop/0                                 7             6.730403
rcuos/0                                 7             6.469543
sugov:0                                 3             6.113958
/vendor/bin/hw/andro                    3             5.919216
```

New constructs

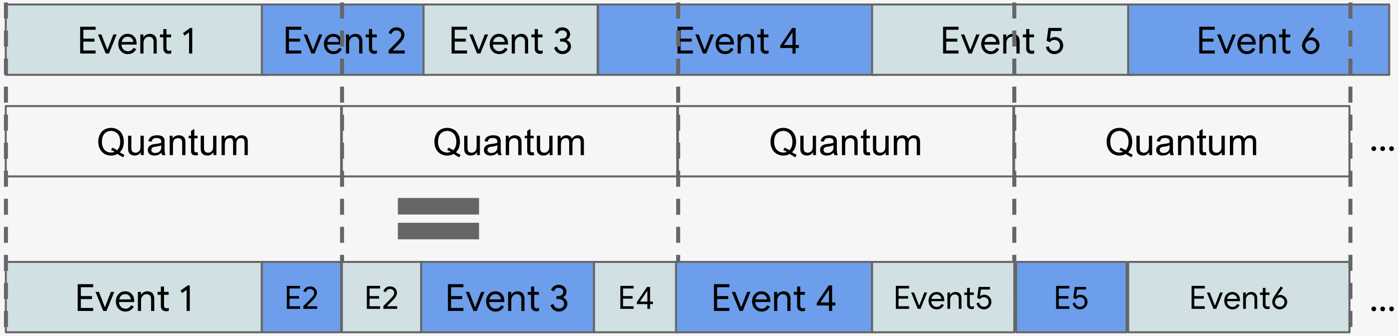
Windowing



```
CREATE VIRTUAL TABLE bounds USING window;
UPDATE bounds SET window_start=X, window_dur=Y where 1
CREATE VIRTUAL TABLE clipped USING span(sched, bounds)
```

New constructs

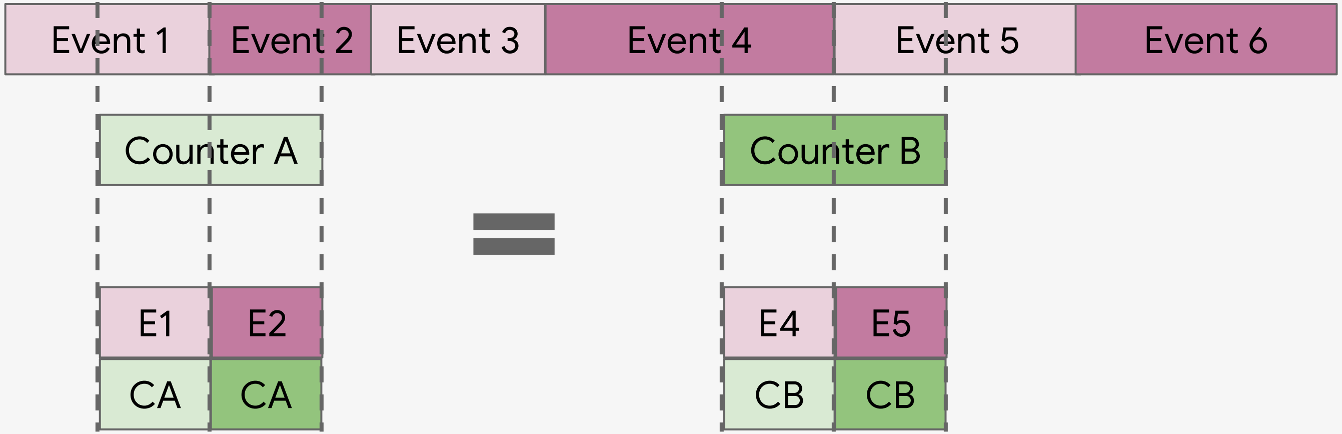
Quantization



```
CREATE VIRTUAL TABLE bounds USING window;
UPDATE bounds SET quantum=Z where 1
CREATE VIRTUAL TABLE quantized USING span(sched, bounds)
```

New constructs

Harmonization



```
CREATE VIRTUAL TABLE quantized USING span(sched, counters, cpu)
```

Join key

UI

Perfetto UI

Re-written from scratch from the ashes of chrome://tracing

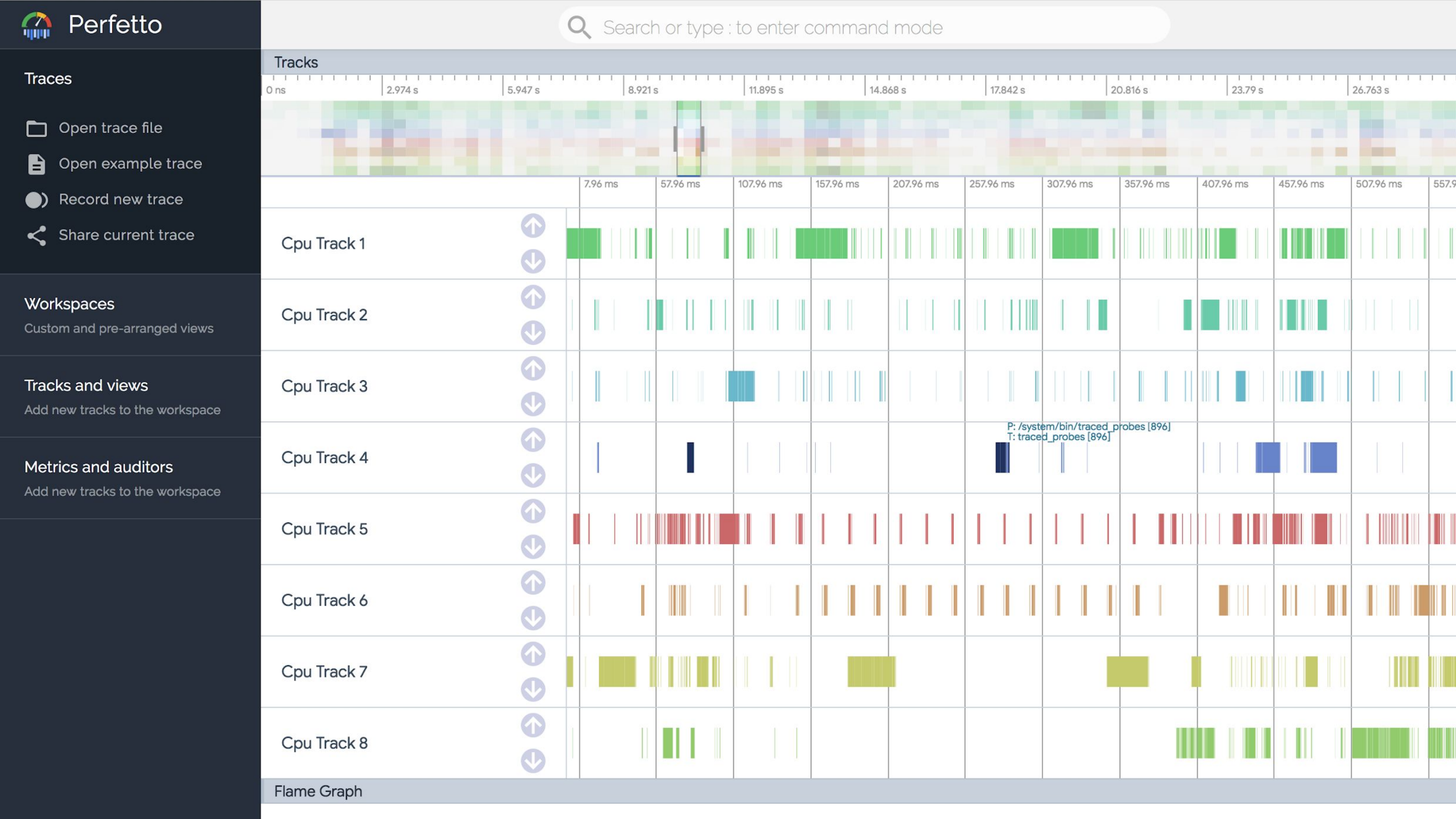
Web-based: TypeScript + WebAssembly running in a worker

All the processing / analysis engine is based on the Trace Processor

Supports ~5 GB traces (limited by browser renderer limit)

URL: https://ui.perfetto.dev

Or just build it from sources and run locally.


Perfetto
Search or type : to enter command mode
Traces
Open trace file
Open example trace
Record new trace
Share current trace
Workspaces
Custom and pre-arranged views
Tracks and views
Add new tracks to the workspace
Metrics and auditors
Add new tracks to the workspace
Tracks
0 ns
2.974 s
5.947 s
8.921 s
11.895 s
14.868 s
17.842 s
20.816 s
23.79 s
26.763 s
7.96 ms
57.96 ms
107.96 ms
157.96 ms
207.96 ms
257.96 ms
307.96 ms
357.96 ms
407.96 ms
457.96 ms
507.96 ms
557.9
Cpu Track 1
Cpu Track 2
Cpu Track 3
Cpu Track 4
P: /system/bin/traced_probes [896]
T: traced_probes [896]
Cpu Track 5
Cpu Track 6
Cpu Track 7
Cpu Track 8
Flame Graph

Thanks for your attention

For docs / links:

www.perfetto.dev

primiano@google.com