Understanding how a web browser works

or tracing your way out of (performance) problems

Alexander Timin (altimin@chromium.org)
FOSDEM 2024
(self-link)

What is this talk actually about?

- Hi, I’m Alex
  - Software engineer in Google’s Web-on-Android Performance team for the last 8+ years
- Problem solving in complex systems with illustrations
  - Chromium (chromium.org)
  - Perfetto (perfetto.dev)
- Not a how-to guide, but hopefully source of inspiration
  - For actually practically useful stuff, see this “intro to Chrome tracing” article
  - Would love to hear and chat more about similar problems

So, you want to improve performance

- Knowing what to improve is often most of the effort
- Performance problems can be anywhere in the code
- Modern web is complex (API surface / browser implementation / various sites)

⇒ ... then you'll be spending considerable effort understanding new code on a recurring basis

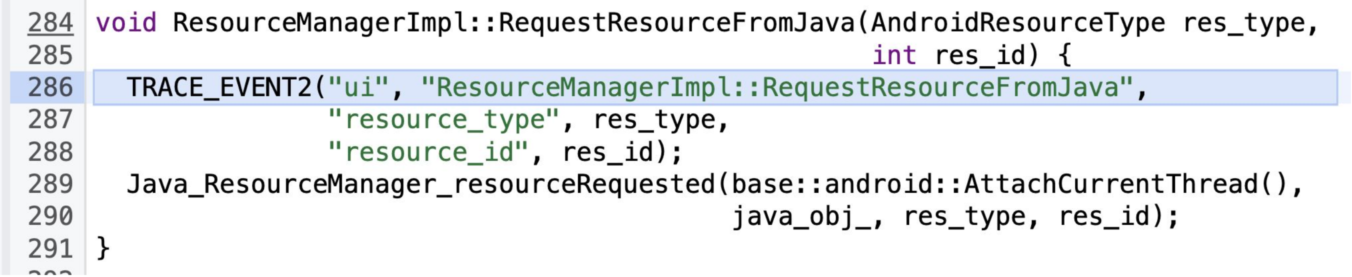
How can do it?

- Read the code
    - Good luck!
- fprintf
    - console.log, (V)LOG, etc.
- debugger
    - gdb, lldb, rr, Chrome DevTools
- These approaches don’t scale effectively to complex environments
    - Especially when multiple threads/processes are involved
    - Indeterminism (flaky tests)
    - Typically focusing on low-level details, not insights into high-level architecture

Enter tracing

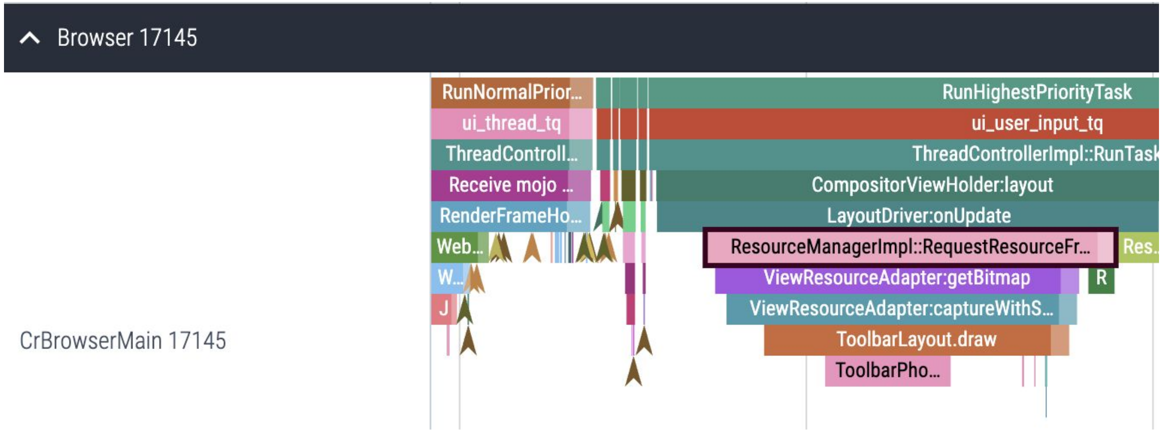
Structured logging with visualisation:

- Turning this:

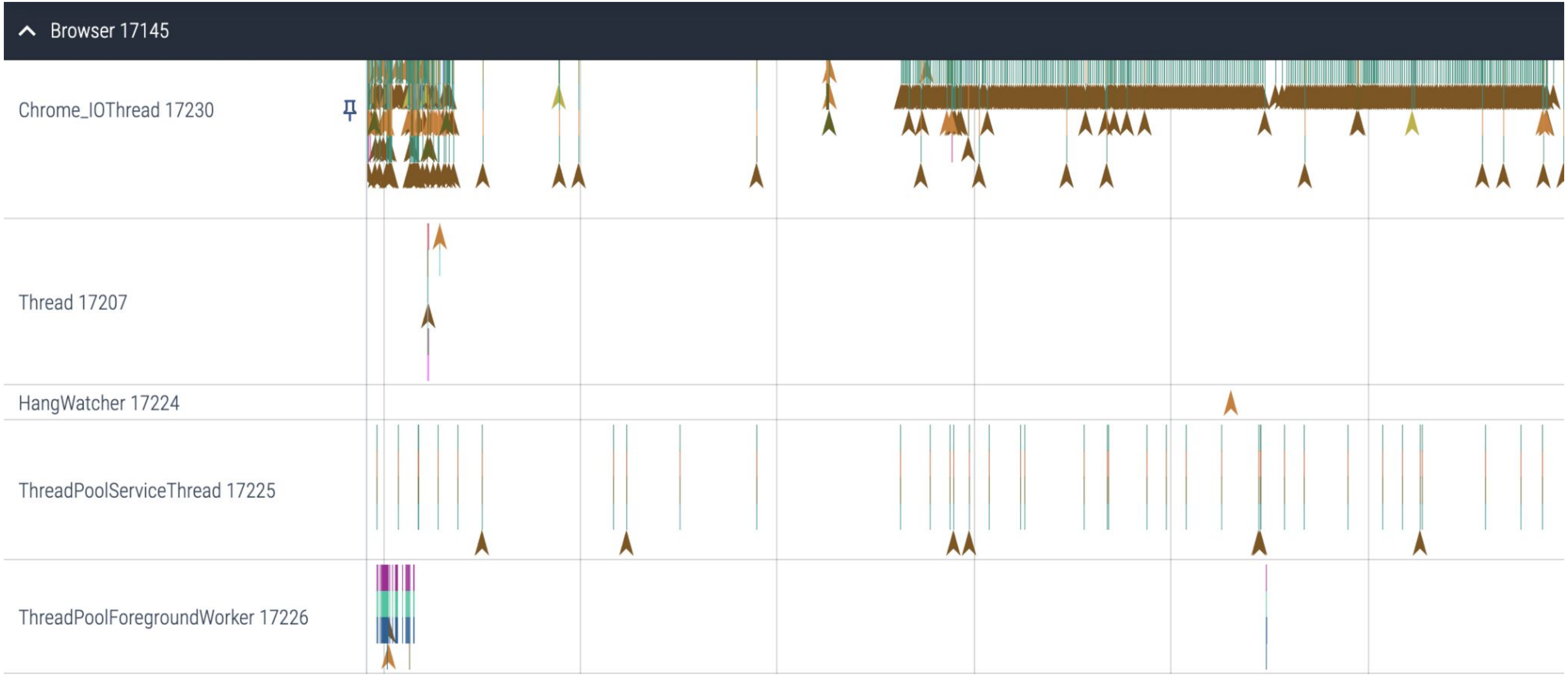


```
void ResourceManagerImpl::RequestResourceFromJava(AndroidResourceType res_type,
                                                  int res_id) {
  TRACE_EVENT2("ui", "ResourceManagerImpl::RequestResourceFromJava",
               "resource_type", res_type,
               "resource_id", res_id);
  Java_ResourceManager_resourceRequested(base::android::AttachCurrentThread(),
                                         java_obj_, res_type, res_id);
}
```

- Into this:



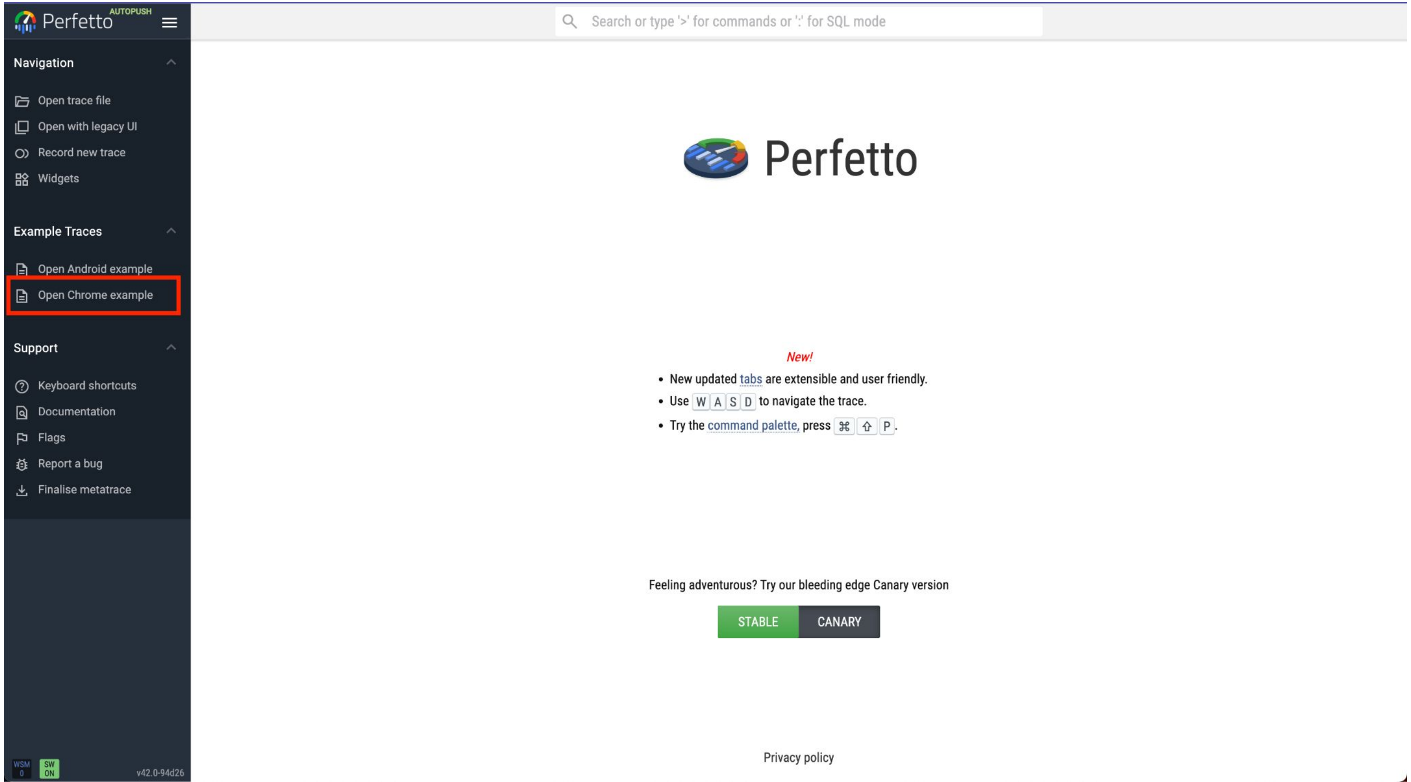
Enter tracing



Visualisation of what multiple threads / processes do in parallel

If you want to try it yourself

- ui.perfetto.dev + “Open Chrome example”



New!

- New updated tabs are extensible and user friendly.
- Use W A S D to navigate the trace.
- Try the command palette, press ⌘ ⇧ P.

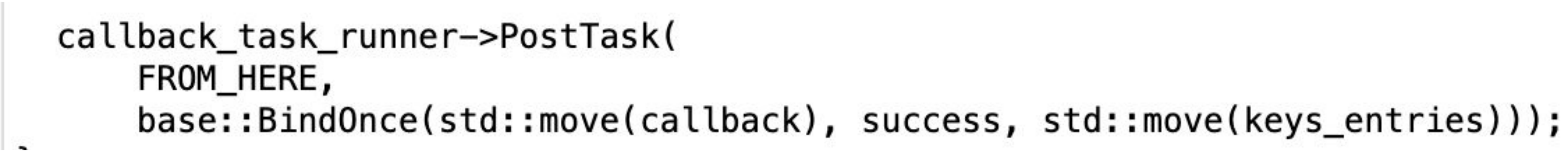
Feeling adventurous? Try our bleeding edge Canary version

But how to make it useful?

- Starting point: instrumenting the code you are working on
  - Flexible and powerful, but not most convenient
  - Folks want to solve the problem, not add instrumentation
    - a single fprintf is more convenient
    - debuggers are guaranteed to have all information
- Unrealistic to have all functions instrumented
  - Too much data and overhead: slow to record and analyse
- Finding opportunities for scaling the usefulness
  - Few instrumentation points which give multiple insights
  - Usually infra / foundational pieces

Chromium task scheduler

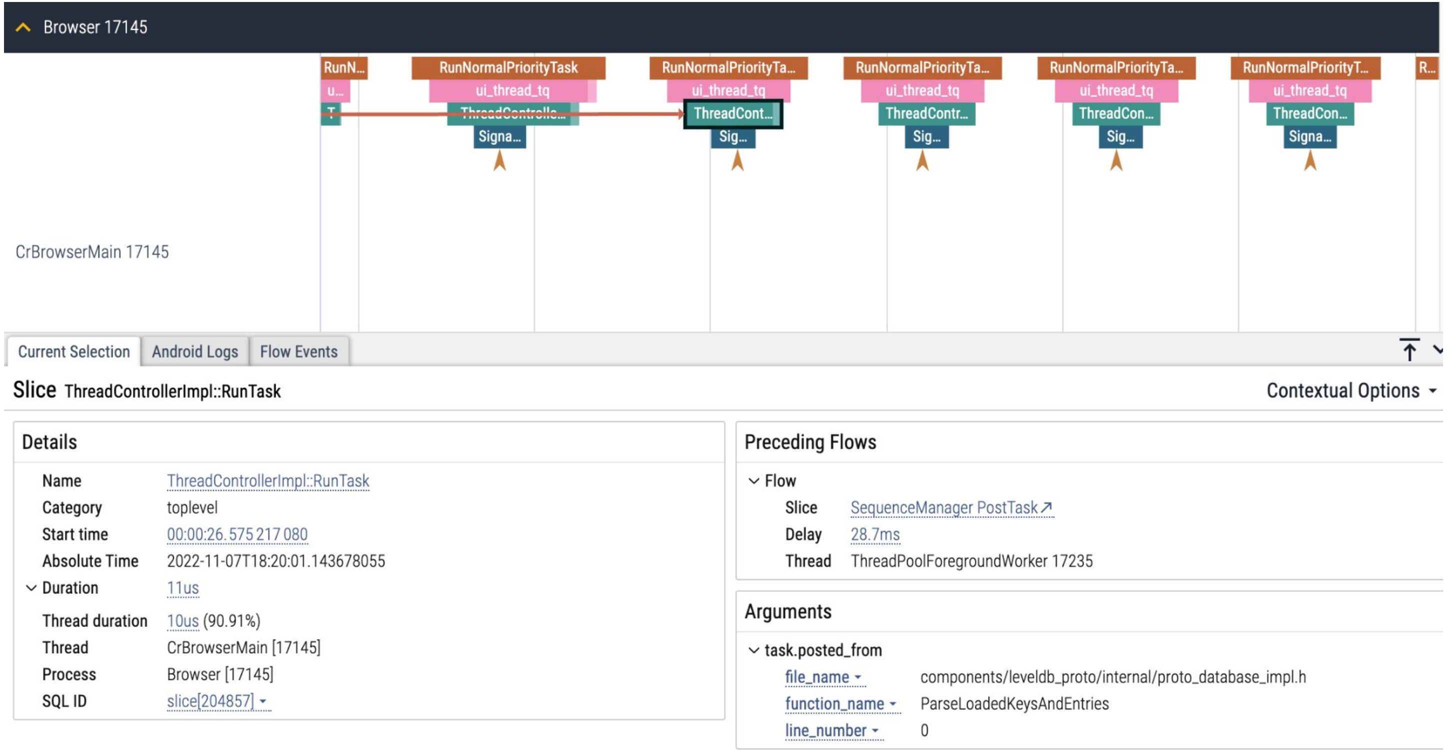
- Event loop model:
  - Thread schedulers for “named” threads
  - Thread pool for “background” work
- Various places in the codebase post tasks:



```
callback_task_runner->PostTask(
    FROM_HERE,
    base::BindOnce(std::move(callback), success, std::move(keys_entries)));
```

- Great chokepoint for tracing instrumentation
  - A couple of paths ~all work in Chromium is going through
  - Can get basic info which part of the codebase a given task is coming from

Chromium task scheduler: a single task



Slice ThreadControllerImpl::RunTask

Details

Name	ThreadControllerImpl::RunTask
Category	toplevel
Start time	00:00:26.575 217 080
Absolute Time	2022-11-07T18:20:01.143678055
Duration	11us
Thread duration	10us (90.91%)
Thread	CrBrowserMain [17145]
Process	Browser [17145]
SQL ID	slice[204857]

Preceding Flows

Flow

Slice	SequenceManager PostTask ↗
Delay	28.7ms
Thread	ThreadPoolForegroundWorker 17235

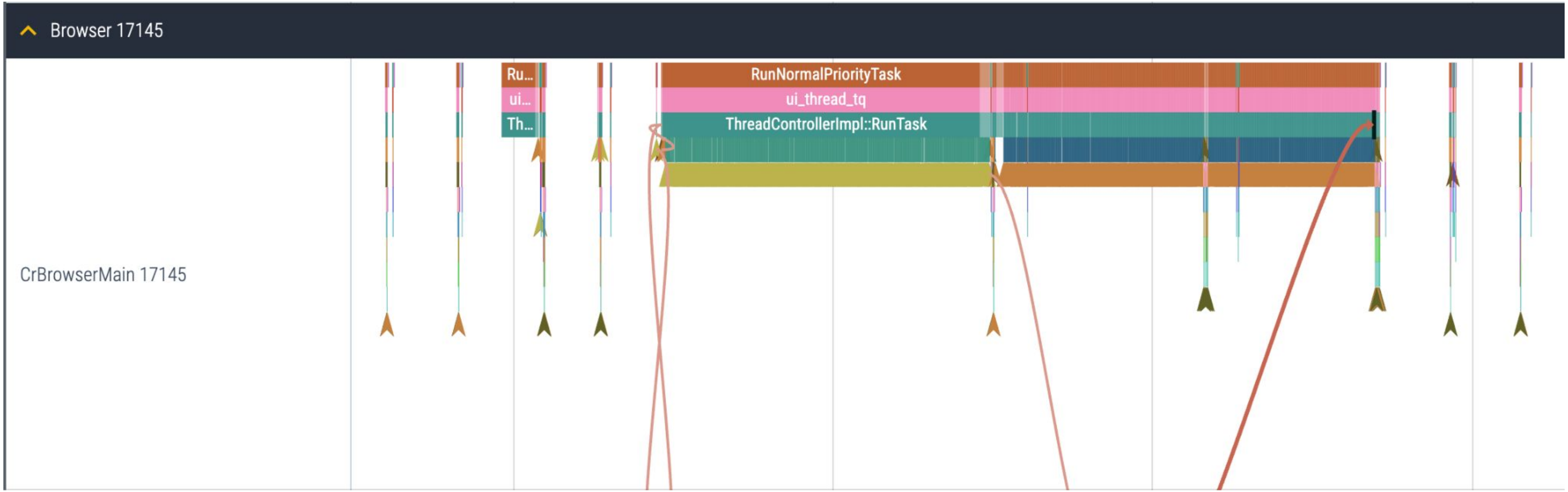
Arguments

task.posted_from

file_name	components/leveldb_proto/internal/proto_database_impl.h
function_name	ParseLoadedKeysAndEntries
line_number	0

A single task (RunTask): FROM_HERE provides basic info about the task

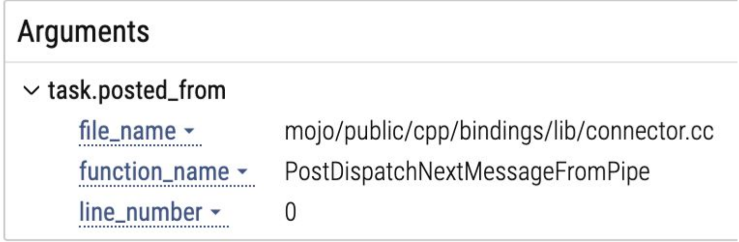
Chromium task scheduler: macro-level



Overview of all thread activity
RunTask trace events: cross-task dependencies are very powerful

Beyond task scheduler

- FROM_HERE might be useful
  - And might be not
- Other “chokepoints”
  - IPC system (mojo): cross-process communication
  - console.log & (D)(V)LOG
  - blink bindings (JS => C++ boundary): which JS calls are being made
  - JNI: Java => C++ boundary
  - GPU scheduler
  - Blink dispatched events
  - locks and other //base primitives



Arguments

task.posted_from

file_name	mojo/public/cpp/bindings/lib/connector.cc
function_name	PostDispatchNextMessageFromPipe
line_number	0

What’s next?

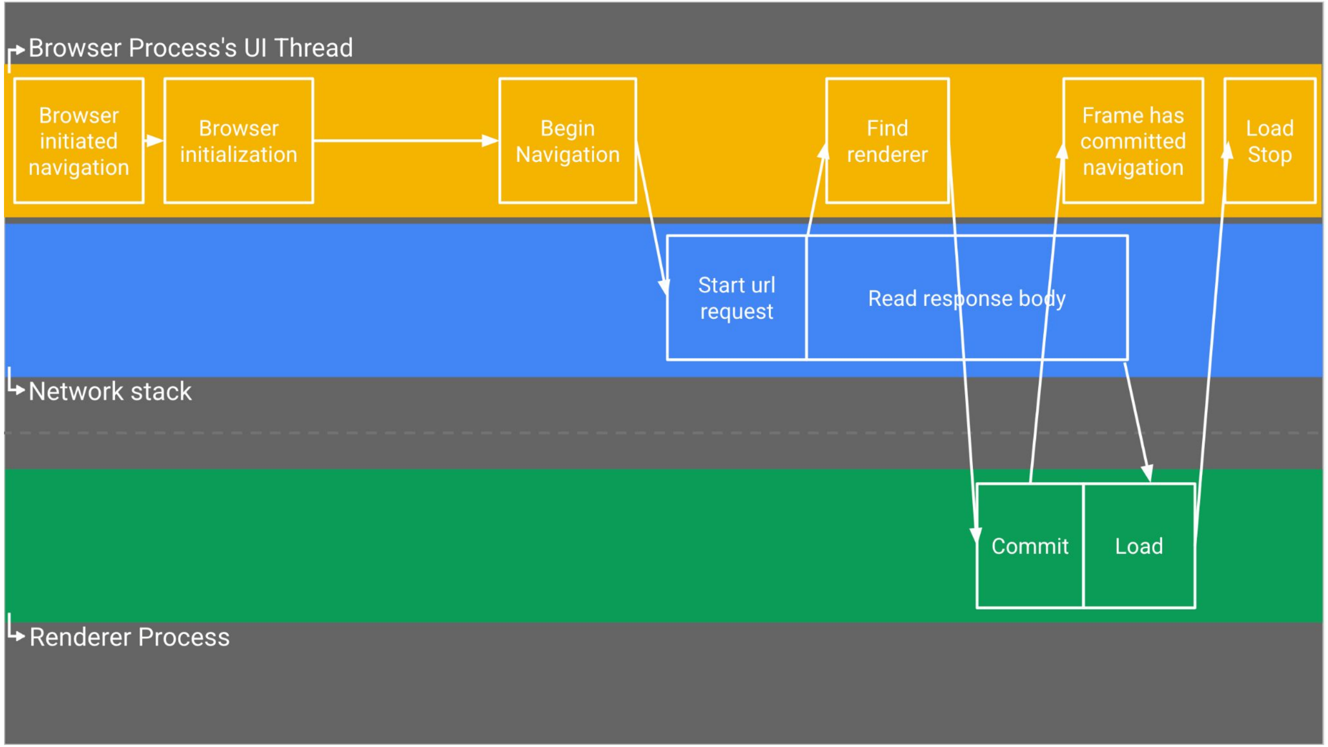
Status quo:

- **Good:** we have visibility into ~everything Chromium is doing
- **Bad:** it’s mostly low-level details and slow to work with
- **Ugly:** expertise-intensive

Aspiration:

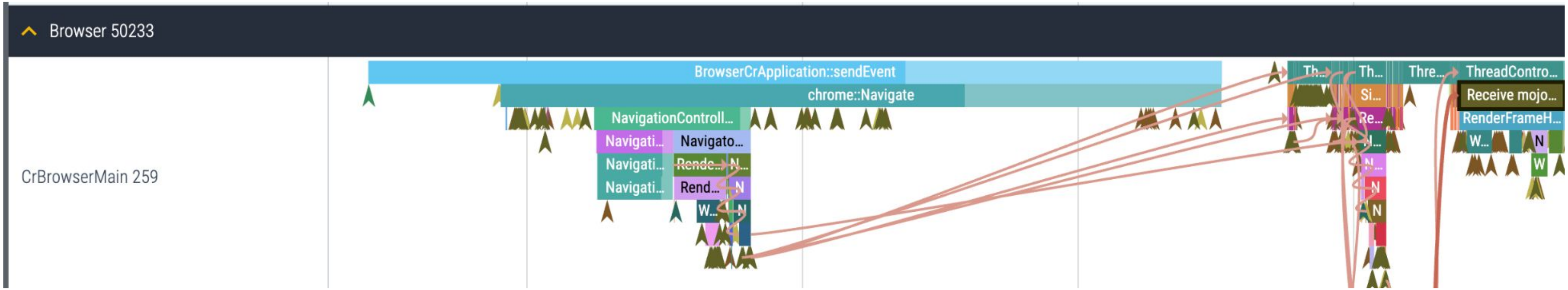
- One can open a trace and learn something about how Chromium works
- (instead of requiring MS in tracing and PhD in Chromium architecture)

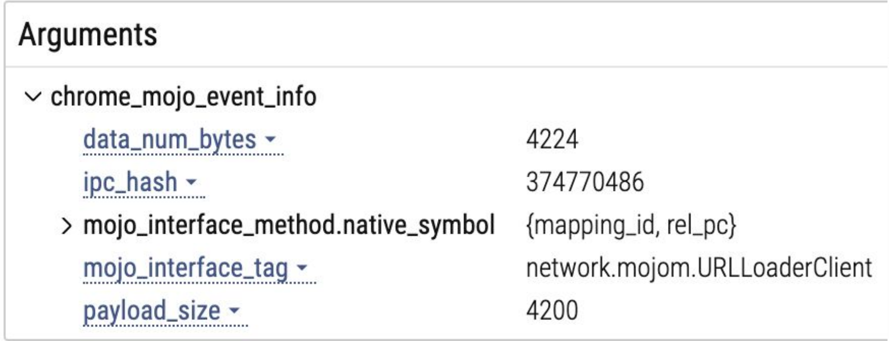
Inspiration



Architecture diagram from a Life of a Navigation talk from Chromium University

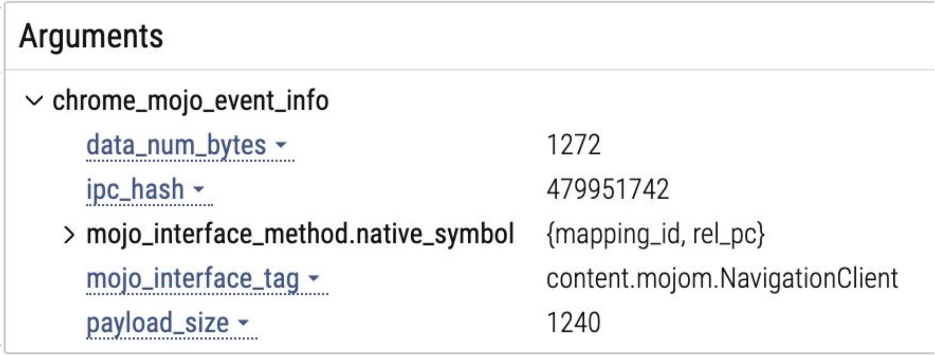
… and the status quo





Arguments

⌄ chrome_mojo_event_info	
data_num_bytes ▾	4224
ipc_hash ▾	374770486
› mojo_interface_method.native_symbol	{mapping_id, rel_pc}
mojo_interface_tag ▾	network.mojom.URLLoaderClient
payload_size ▾	4200



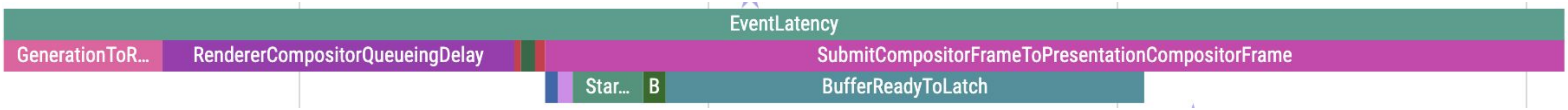
Arguments

⌄ chrome_mojo_event_info	
data_num_bytes ▾	1272
ipc_hash ▾	479951742
› mojo_interface_method.native_symbol	{mapping_id, rel_pc}
mojo_interface_tag ▾	content.mojom.NavigationClient
payload_size ▾	1240

The information is there, but the same insights will take a bit longer to get (trace)

Existing examples

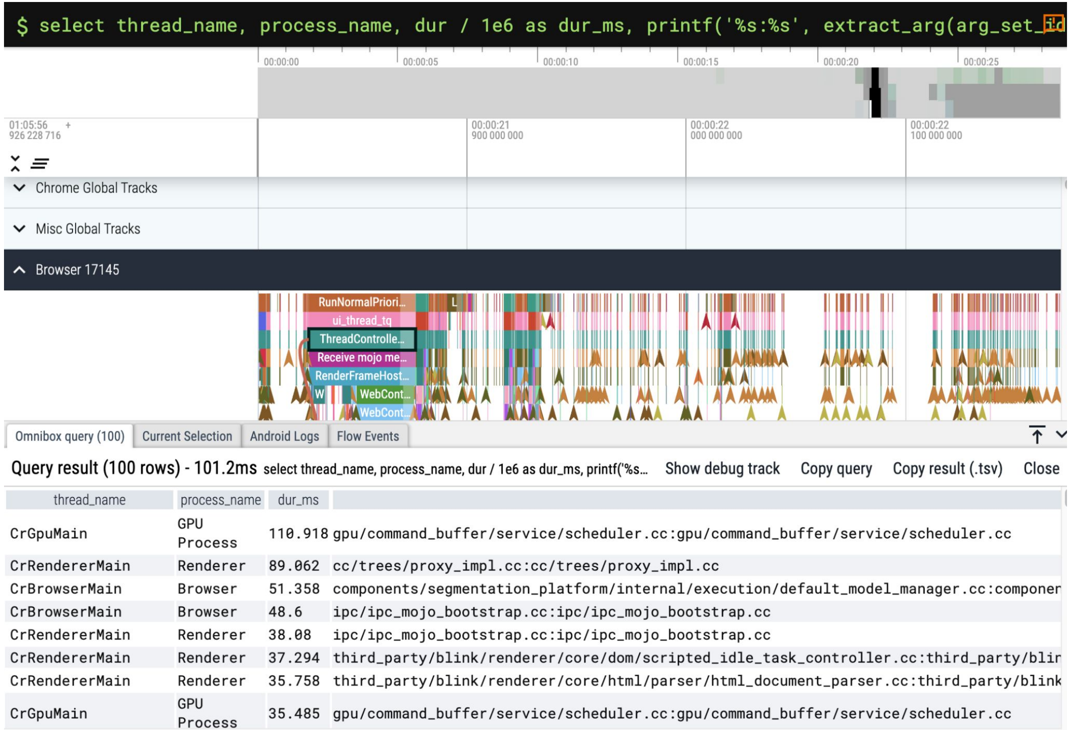
- EventLatency: breakdown of processing an input event and generating a frame
- Currently requires plumbing all of the data to a single location
    - Plumbing is very expensive in a large project (e.g. layering concerns, serialisation cost)
    - Difficult to scale


EventLatency
GenerationToR...
RendererCompositorQueueingDelay
SubmitCompositorFrameToPresentationCompositorFrame
Star...
B
BufferReadyToLatch

Enter Perfetto

- From chrome://tracing to perfetto.dev
- New UI, new more efficient format
- SQL data mode and query engine
  - Running custom queries from the UI
  - Running trace processor + SQLite in the browser via WASM
- Allows separation of “recording” and “analysis"

Perfetto powers



```
$ select thread_name, process_name, dur / 1e6 as dur_ms, printf('%s:%s', extract_arg(arg_set_id
```

Query result (100 rows) - 101.2ms select thread_name, process_name, dur / 1e6 as dur_ms, printf('%s...

thread_name	process_name	dur_ms	
CrGpuMain	GPU Process	110.918	gpu/command_buffer/service/scheduler.cc:gpu/command_buffer/service/scheduler.cc
CrRendererMain	Renderer	89.062	cc/trees/proxy_impl.cc:cc/trees/proxy_impl.cc
CrBrowserMain	Browser	51.358	components/segmentation_platform/internal/execution/default_model_manager.cc:componer
CrBrowserMain	Browser	48.6	ipc/ipc_mojo_bootstrap.cc:ipc/ipc_mojo_bootstrap.cc
CrRendererMain	Renderer	38.08	ipc/ipc_mojo_bootstrap.cc:ipc/ipc_mojo_bootstrap.cc
CrRendererMain	Renderer	37.294	third_party/blink/renderer/core/dom/scripted_idle_task_controller.cc:third_party/blir
CrRendererMain	Renderer	35.758	third_party/blink/renderer/core/html/parser/html_document_parser.cc:third_party/blink
CrGpuMain	GPU Process	35.485	gpu/command_buffer/service/scheduler.cc:gpu/command_buffer/service/scheduler.cc

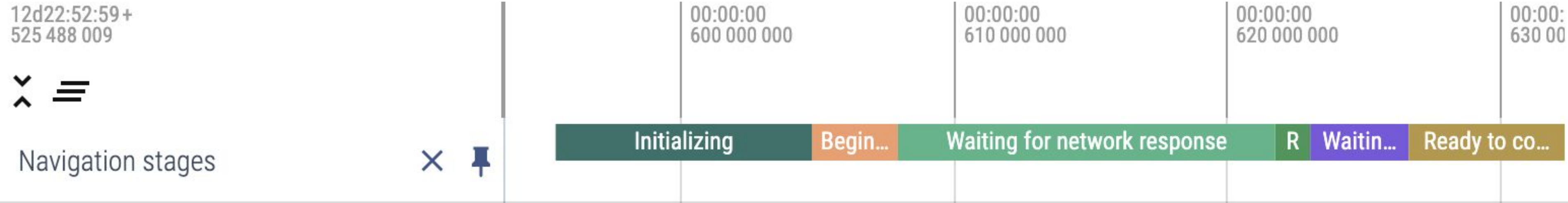
Enter “:” into the search box to enter the SQL mode

Query: select thread_name, process_name, dur / 1e6 as dur_ms, printf('%s:%s', extract_arg(arg_set_id, 'task.posted_from.file_name'), extract_arg(arg_set_id, 'task.posted_from.file_name')) as posted_from from thread_slice where name = 'ThreadControllerImpl::RunTask' ORDER BY dur desc limit 100

Next steps

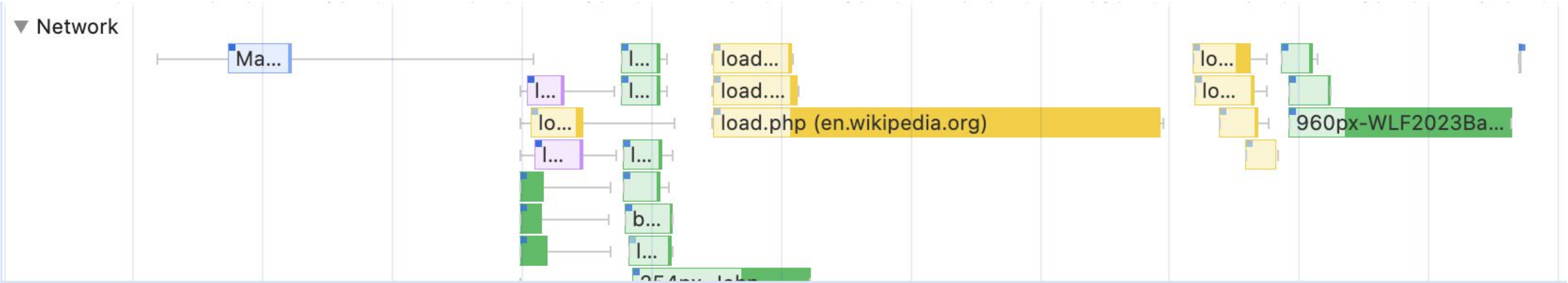
- Trying to build navigation instrumentation in Chromium as PoC
  - Focusing on the higher-level concepts
  - Links to the lower-level implementation details (e.g. specific functions being called)
  - Inline documentation in the UI and explaining the concepts
- Challenge: complexity and # of corner cases
  - ~50+ of various cases which affect the breakdown
  - Automatic testing is a prerequisite

Current status of the prototype:



Bonus: Chrome DevTools

and the importance of presenting the right information

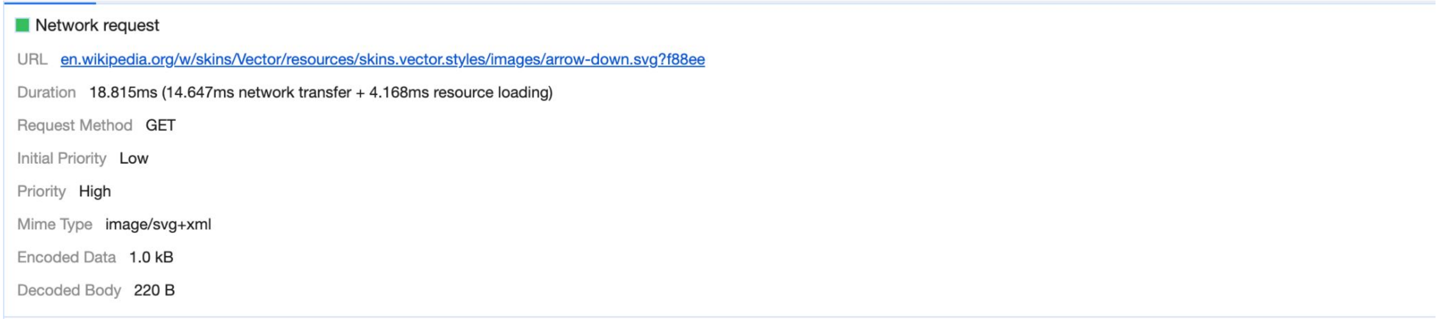


Screenshot of a network section of a performance trace from Chrome DevTools (trace)

Bonus: Chrome DevTools

and the importance of presenting the right information

It’s just Chrome traces with post-processing in DevTools frontend



🟩 Network request

URL en.wikipedia.org/w/skins/Vector/resources/skins.vector.styles/images/arrow-down.svg?f88ee

Duration 18.815ms (14.647ms network transfer + 4.168ms resource loading)

Request Method GET

Initial Priority Low

Priority High

Mime Type image/svg+xml

Encoded Data 1.0 kB

Decoded Body 220 B

You can open the same trace in chrome://tracing / Perfetto, but it will be less useful

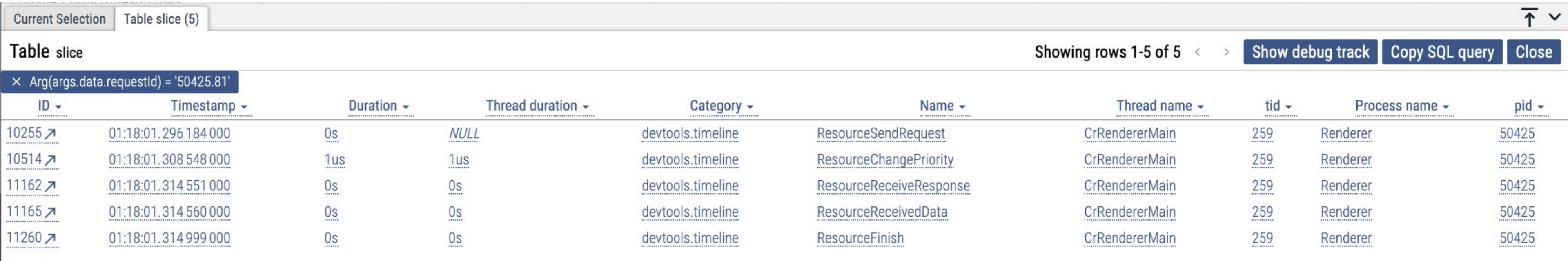


Table slice

Showing rows 1-5 of 5

× Arg(args.data.requestId) = '50425.81'

ID	Timestamp	Duration	Thread duration	Category	Name	Thread name	tid	Process name	pid
10255 ↗	01:18:01.296 184 000	0s	*NULL*	devtools.timeline	ResourceSendRequest	CrRendererMain	259	Renderer	50425
10514 ↗	01:18:01.308 548 000	1us	1us	devtools.timeline	ResourceChangePriority	CrRendererMain	259	Renderer	50425
11162 ↗	01:18:01.314 551 000	0s	0s	devtools.timeline	ResourceReceiveResponse	CrRendererMain	259	Renderer	50425
11165 ↗	01:18:01.314 560 000	0s	0s	devtools.timeline	ResourceReceivedData	CrRendererMain	259	Renderer	50425
11260 ↗	01:18:01.314 999 000	0s	0s	devtools.timeline	ResourceFinish	CrRendererMain	259	Renderer	50425