Chromium on Android
Web Performance

How we **doubled Chromium's Speedometer** scores &
developed the **LoadLine** page load benchmark

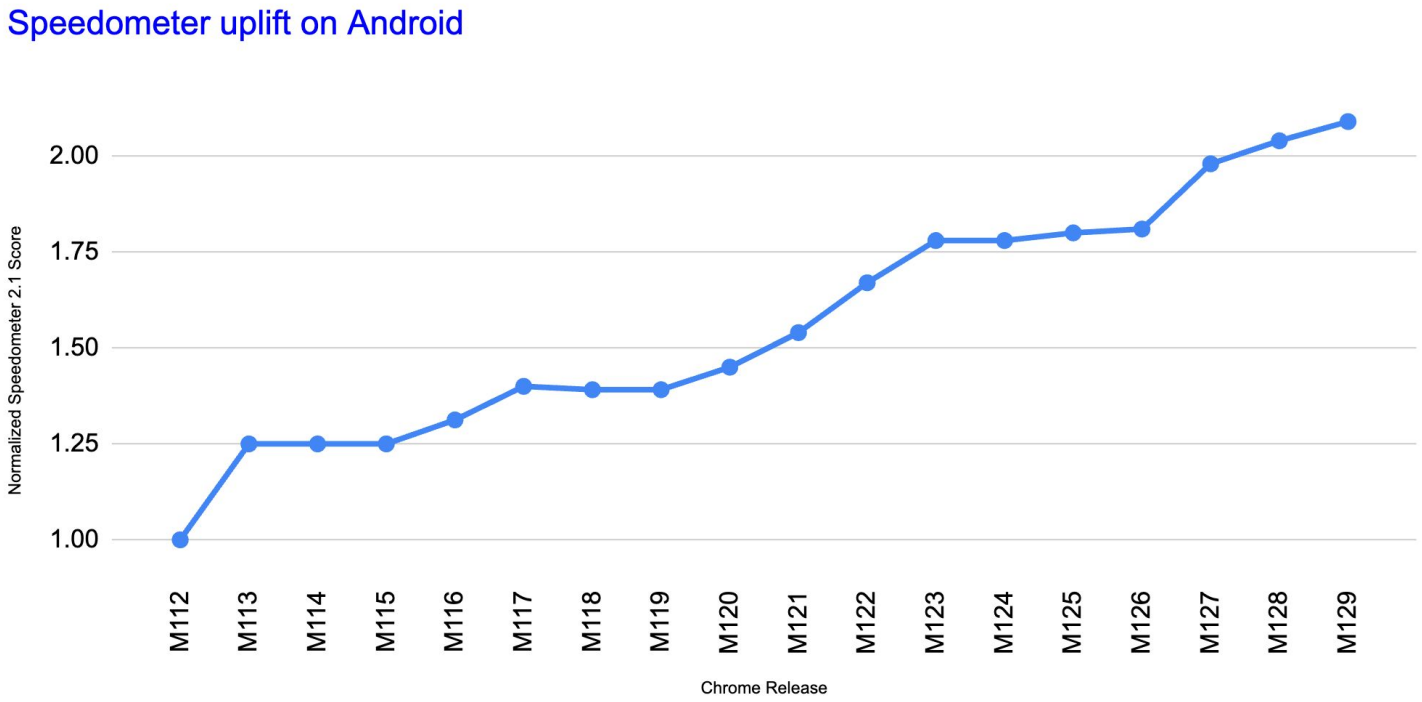
eseckler@chromium.org | Eric Seckler | Google UK | Android Web Perf

February, 2025

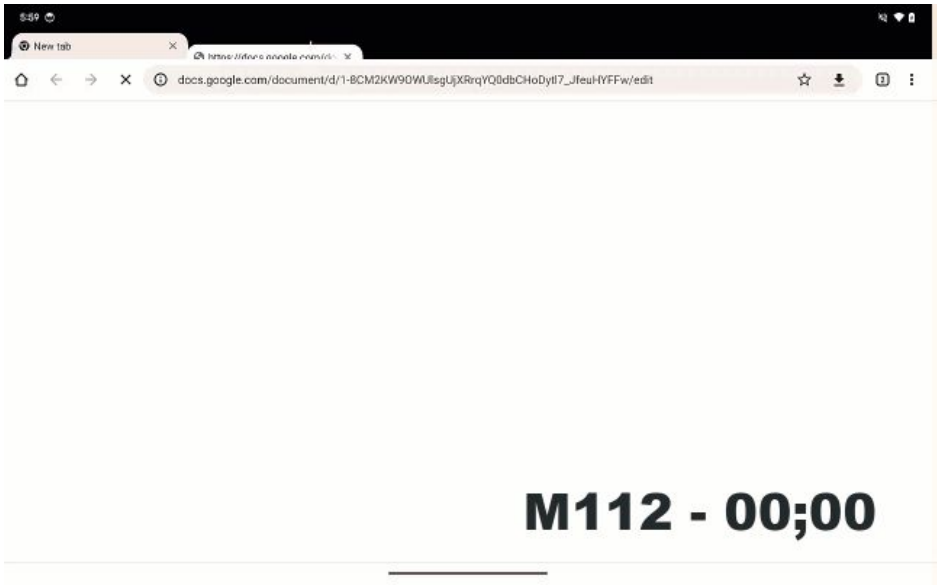
chromium

Speedometer on Android Chrome 🎉

Speedometer 2.1 scores on Android Chrome **increased 109%** (e.g. Pixel Tablet: 97 in 2023 to 203 today).



Faster Speedometer => Faster page load



Thank you for your contributions!

- Chrome/V8
- Android
- Pixel
- ARM
- Qualcomm

Table of contents

01 Breaking down the timeline

02 Deep-dive: Build optimizations

03 Tooling to enable further analysis

04 Beyond Speedometer: LoadLine

05 Q&A

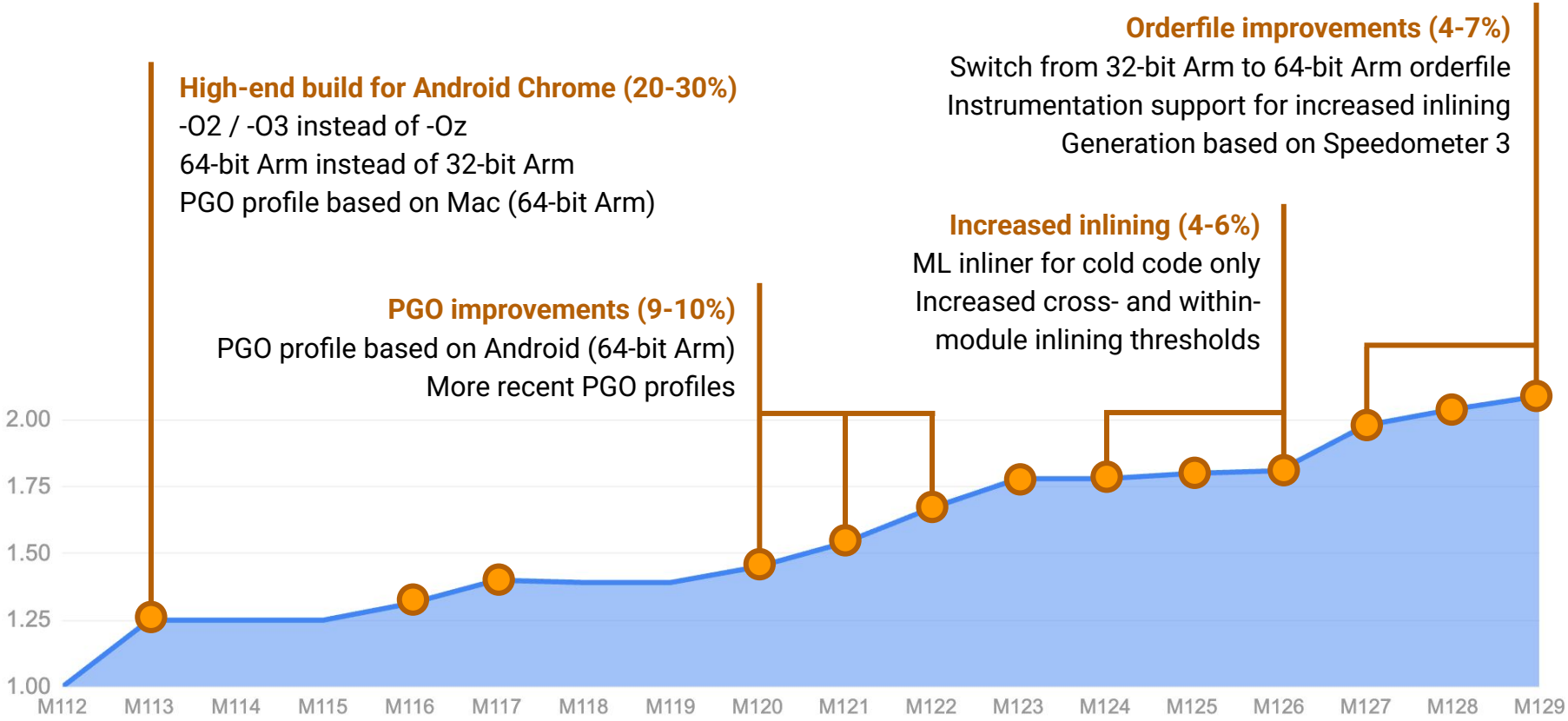
Breaking down the timeline!

01.1 Build improvements

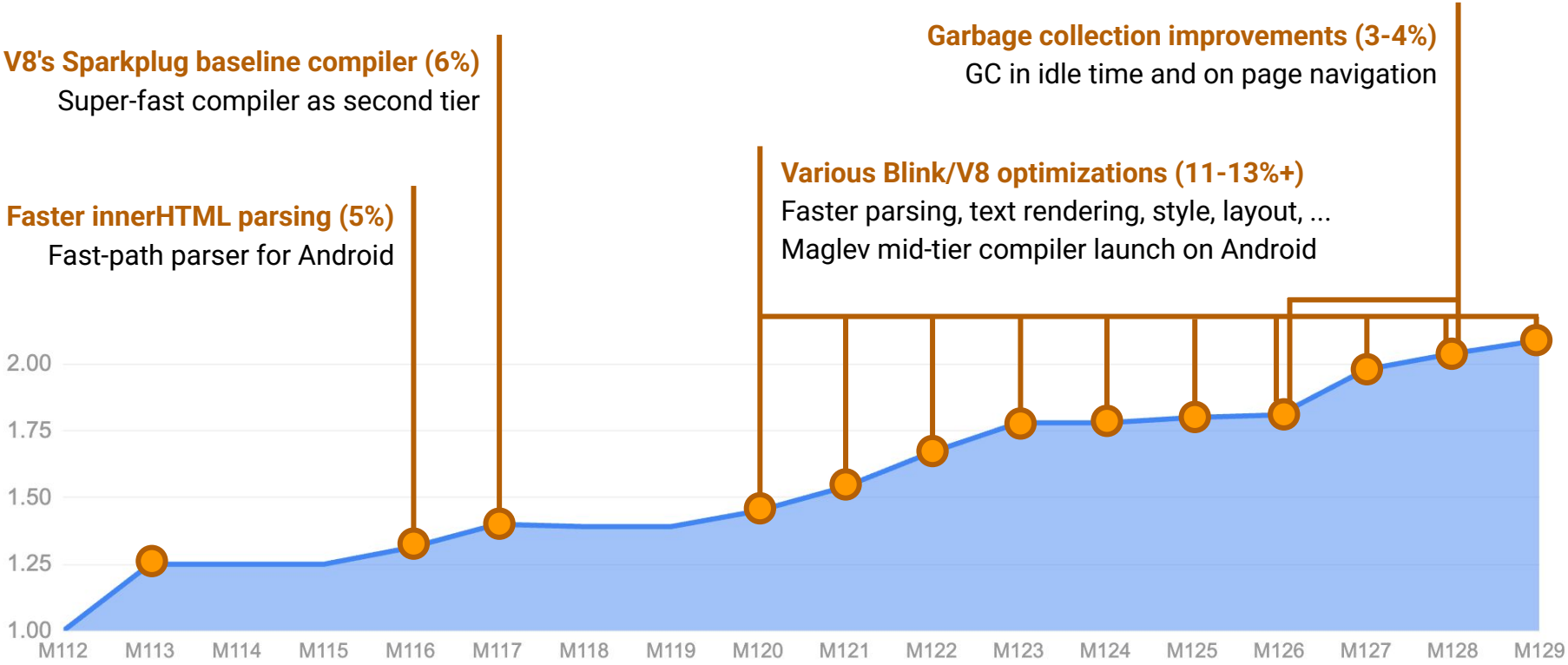
01.2 V8 and Blink improvements

01.3 Scheduling & OS

Build improvements

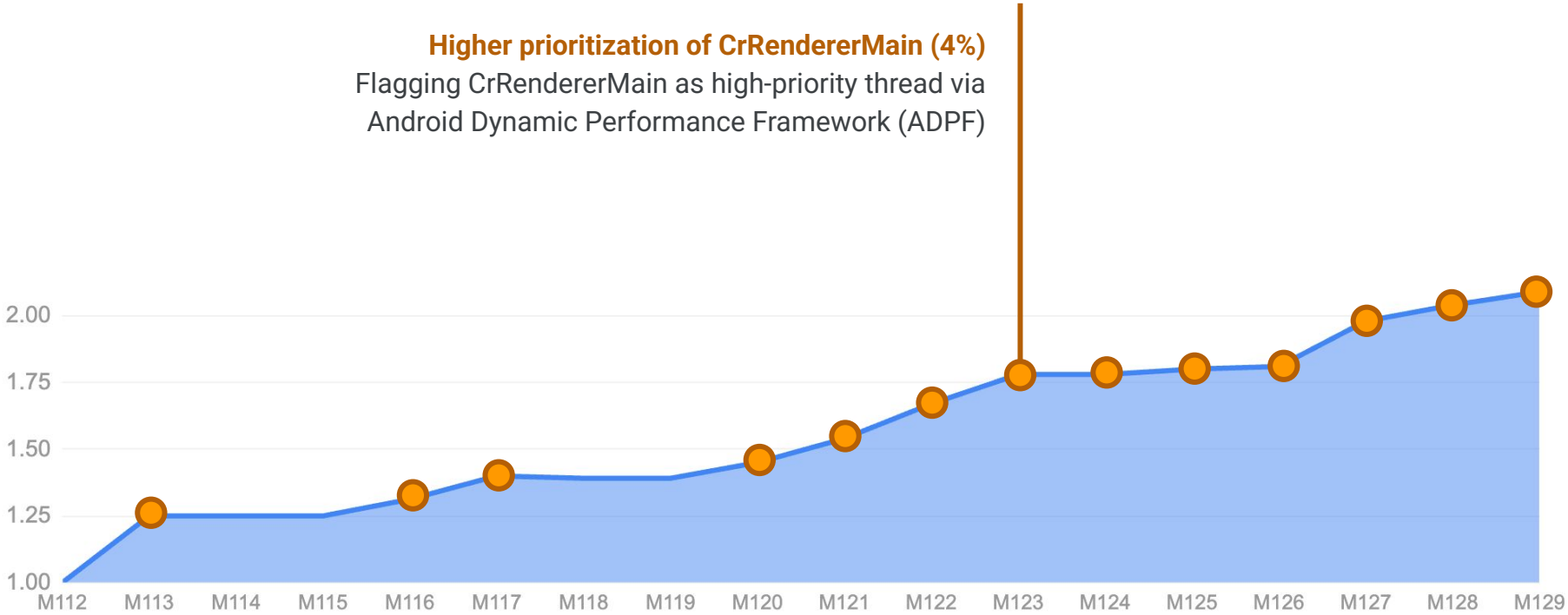


V8 and Blink improvements



Scheduling & OS

Higher prioritization of CrRendererMain (4%)
Flagging CrRendererMain as high-priority thread via Android Dynamic Performance Framework (ADPF)



Insights: Build improvements

Profiling to understand Chrome's CPU bottlenecks

PMU data: high stalls & frontend-bound workload

Suite	Frontend stalls	Backend stalls
Angular2-TypeScript-TodoMVC	45.92%	18.32%
AngularJS-TodoMVC	43.12%	17.47%
BackboneJS-TodoMVC	55.49%	14.97%
Elm-TodoMVC	28.91%	19.35%
EmberJS-Debug-TodoMVC	28.72%	24.77%
EmberJS-TodoMVC	40.41%	20.01%
Flight-TodoMVC	52.32%	17.76%
Inferno-TodoMVC	44.32%	14.69%
Preact-TodoMVC	44.24%	19.10%
React-Redux-TodoMVC	28.03%	16.74%
React-TodoMVC	40.68%	16.36%
Vanilla-ES2015-Babel-Webpack-TodoMVC	38.50%	17.24%
Vanilla-ES2015-TodoMVC	38.42%	17.99%
VanillaJS-TodoMVC	40.66%	17.52%
VueJS-TodoMVC	44.08%	17.19%
jQuery-TodoMVC	28.41%	20.79%
Overall	36.22%	19.27%

Pixel 8 Pro (**Cortex X3**), Chrome **M120**, Speedometer 2.1, CrRendererMain only

Speedometer has **many branches** (~20% of instructions)

Branch mispredicts are costly in many ARM CPUs (instruction cache prefetch)

Optimizing code/branch layout for branch target buffer, caches, and CPU frontend parallelism is critical

CPUs with larger branch predictors and instruction caches are beneficial

Optimized PGO (& CPU) -- bottleneck moves to backend

Subtest	IPC	Frontend stall rate	Backend stall rate
Charts-chartjs	3.19	16.70%	36.16%
Charts-observable-plot	3.01	15.28%	40.54%
Editor-CodeMirror	2.86	23.91%	28.93%
Editor-TipTap	3.36	11.77%	35.44%
NewsSite-Next	2.15	28.81%	35.70%
NewsSite-Nuxt	2.18	28.54%	35.41%
Perf-Dashboard	2.23	29.63%	35.55%
React-Stockcharts-SVG	2.68	20.55%	38.79%
TodoMVC-Angular-Complex-DOM	2.38	24.72%	37.73%
TodoMVC-Backbone	2.10	31.98%	33.02%
TodoMVC-JavaScript-ES5	3.11	15.26%	42.27%
TodoMVC-JavaScript-ES6-[..]	2.96	13.74%	47.57%
TodoMVC-jQuery	3.09	14.26%	42.62%
TodoMVC-Lit-Complex-DOM	2.16	22.96%	43.60%
TodoMVC-Preact-Complex-DOM	1.88	22.73%	47.52%
TodoMVC-React-Complex-DOM	2.62	22.74%	37.56%
TodoMVC-React-Redux	2.91	24.52%	32.13%
TodoMVC-Svelte-Complex-DOM	1.91	24.24%	46.19%
TodoMVC-Vue	2.30	23.91%	39.81%
TodoMVC-WebComponents	2.14	23.08%	43.89%
Overall	**2.69**	**20.78%**	**38.50%**

Pixel 9 Pro (**Cortex X4**), Chrome **M128**, Speedometer 3.0, CrRendererMain only

PGO increases portion of **not-taken** branches by placing hot blocks into fall-through paths

Not-taken branches **consume no BTB space** and enable more efficient utilization of caches and frontend width

Orderfile (function ordering) improves on top of this by **reducing iTLB misses**

Backend bottlenecks are now focus of our investigations

Stalls remain high overall -- high mem-boundedness

Subtest	Frontend stalls	FE L3+ stalls	Backend stalls	BE L3+ stalls
Charts-chartjs	16.70%	7.91%	36.16%	17.30%
Charts-observable-plot	15.28%	6.83%	40.54%	18.94%
Editor-CodeMirror	23.91%	10.78%	28.93%	15.93%
Editor-TipTap	11.77%	4.21%	35.44%	9.29%
NewsSite-Next	28.81%	12.35%	35.70%	16.24%
NewsSite-Nuxt	28.54%	11.57%	35.41%	17.00%
Perf-Dashboard	29.63%	14.39%	35.55%	18.30%
React-Stockcharts-SVG	20.55%	8.84%	38.79%	19.20%
TodoMVC-Angular-Complex-DOM	24.72%	9.13%	37.73%	19.51%
TodoMVC-Backbone	31.98%	11.93%	33.02%	16.90%
TodoMVC-JavaScript-ES5	15.26%	6.82%	42.27%	14.52%
TodoMVC-JavaScript-ES6-[..]	13.74%	7.17%	47.57%	18.90%
TodoMVC-jQuery	14.26%	6.49%	42.62%	14.59%
TodoMVC-Lit-Complex-DOM	22.96%	10.56%	43.60%	21.67%
TodoMVC-Preact-Complex-DOM	22.73%	11.84%	47.52%	28.29%
TodoMVC-React-Complex-DOM	22.74%	8.54%	37.56%	20.13%
TodoMVC-React-Redux	24.52%	9.04%	32.13%	15.17%
TodoMVC-Svelte-Complex-DOM	24.24%	13.58%	46.19%	26.11%
TodoMVC-Vue	23.91%	9.46%	39.81%	21.19%
TodoMVC-WebComponents	23.08%	10.44%	43.89%	19.78%
Overall	20.78%	8.93%	38.50%	16.96%

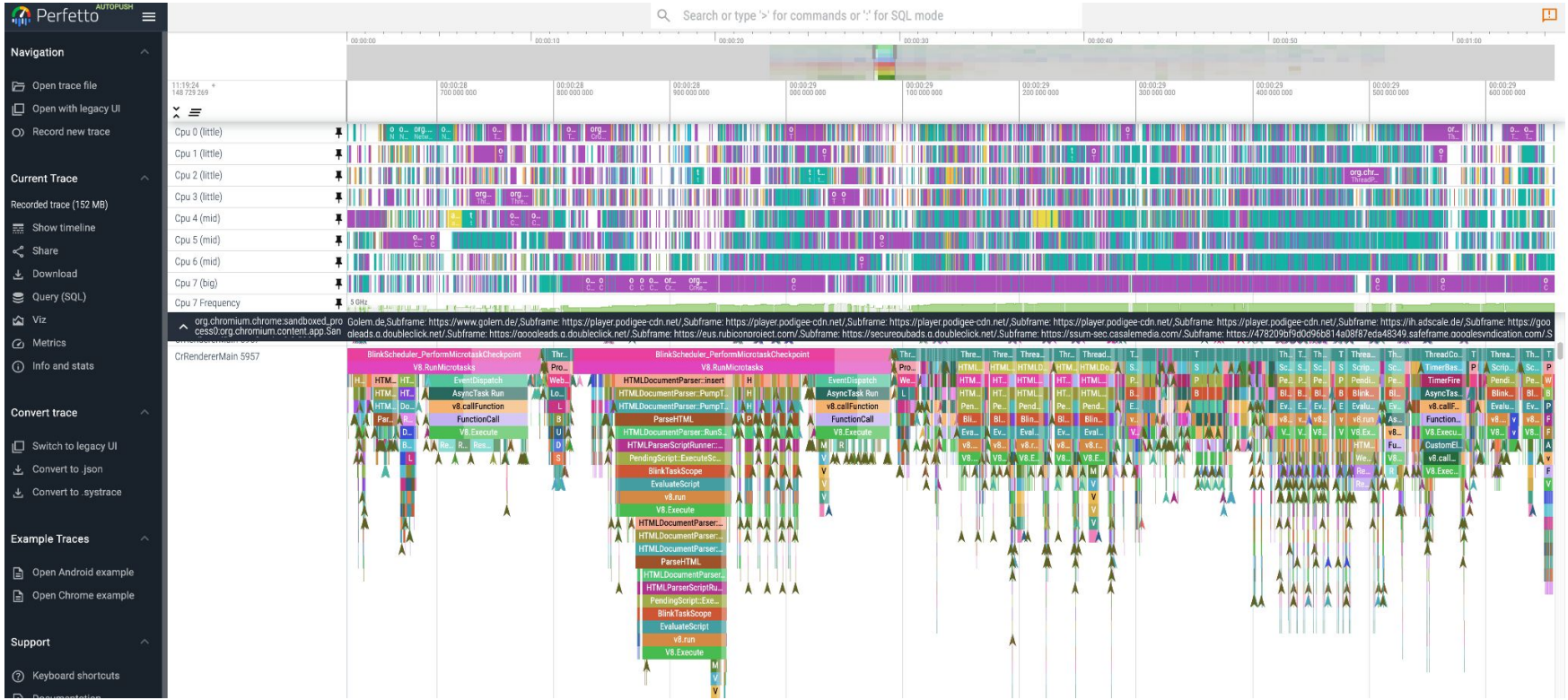
Pixel 9 Pro (**Cortex X4**), Chrome **M128**, Speedometer 3.0, CrRendererMain only

Just under half of the stalls on Cortex X4 stem from **L2 misses**

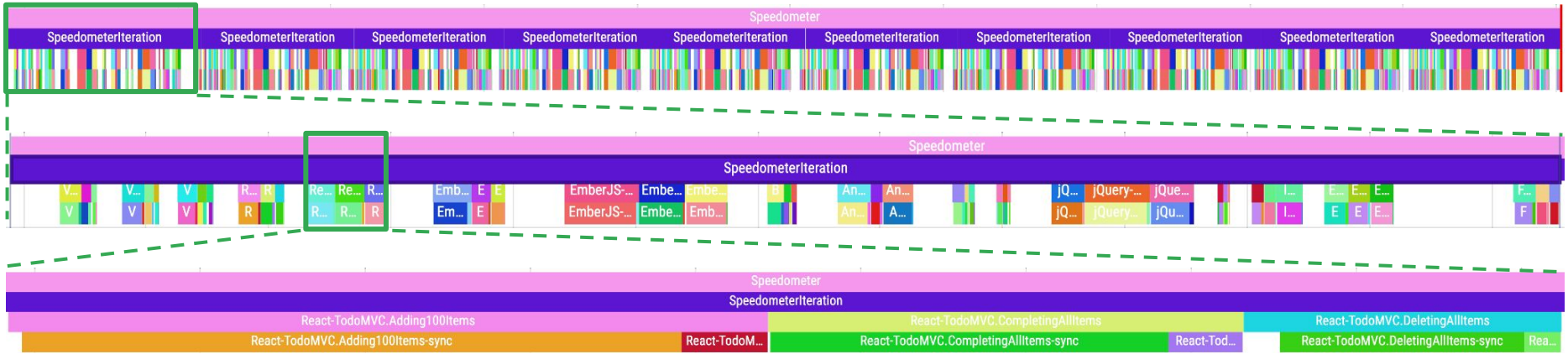
Plan to attribute L2 misses to code and data via profiling

Tooling to enable further analysis

Perfetto with Chrome and system data sources

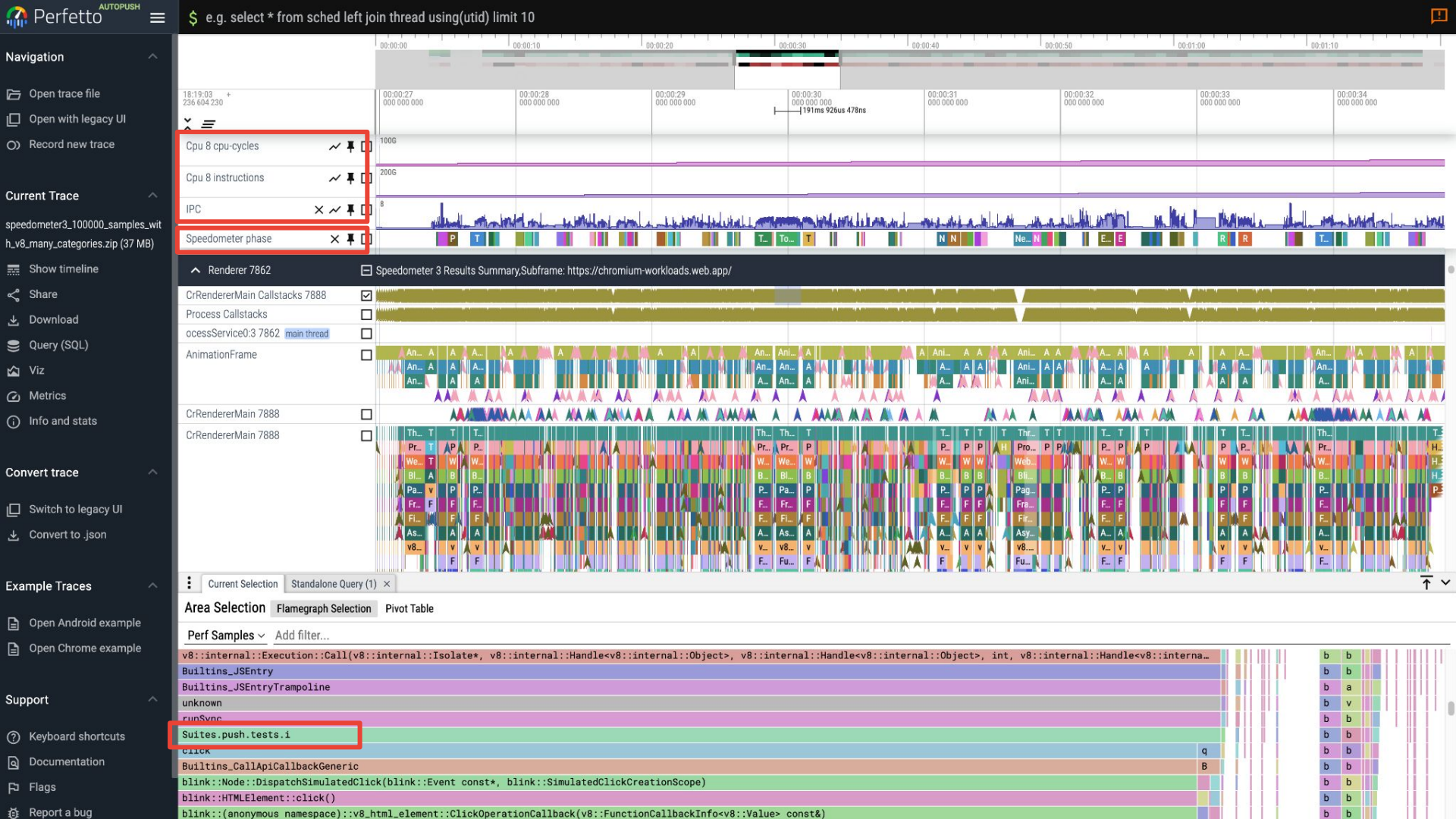


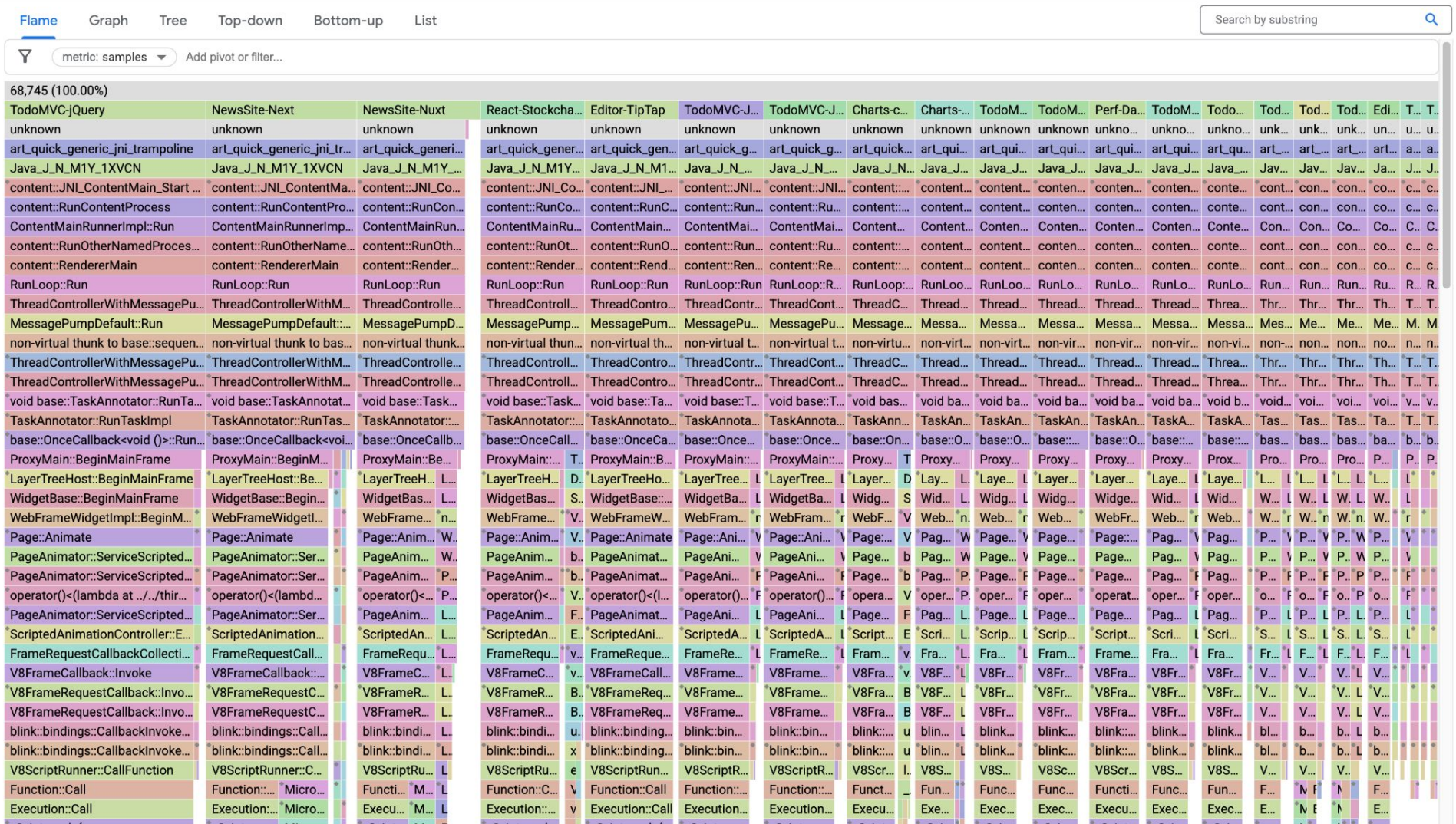
Breaking down Speedometer execution in traces



Annotate Perfetto traces with Speedometer phases + perf/simpleperf integration + C++/V8 symbolization

Allows breakdowns of callstacks, PMU counters, etc. by subtest


Perfetto
AUTOPUSH
e.g. select * from sched left join thread using(utid) limit 10
Navigation
Open trace file
Open with legacy UI
Record new trace
Current Trace
speedometer3_100000_samples_with_v8_many_categories.zip (37 MB)
Show timeline
Share
Download
Query (SQL)
Viz
Metrics
Info and stats
Convert trace
Switch to legacy UI
Convert to .json
Example Traces
Open Android example
Open Chrome example
Support
Keyboard shortcuts
Documentation
Flags
Report a bug
Cpu 8 cpu-cycles
Cpu 8 instructions
IPC
Speedometer phase
191ms 926us 478ns
Renderer 7862
Speedometer 3 Results Summary,Subframe: https://chromium-workloads.web.app/
CrRendererMain Callstacks 7888
Process Callstacks
ocessService0:3 7862
main thread
AnimationFrame
CrRendererMain 7888
CrRendererMain 7888
Current Selection
Standalone Query (1)
Area Selection
Flamegraph Selection
Pivot Table
Perf Samples
Add filter...
v8::internal::Execution::Call(v8::internal::Isolate*, v8::internal::Handle<v8::internal::Object>, v8::internal::Handle<v8::internal::Object>, int, v8::internal::Handle<v8::interna…
Builtins_JSEntry
Builtins_JSEntryTrampoline
unknown
runSync
Suites.push.tests.i
Builtins_CallApiCallbackGeneric
blink::Node::DispatchSimulatedClick(blink::Event const*, blink::SimulatedClickCreationScope)
blink::HTMLElement::click()
blink::(anonymous namespace)::v8_html_element::ClickOperationCallback(v8::FunctionCallbackInfo<v8::Value> const&)


Flame
Graph
Tree
Top-down
Bottom-up
List
Search by substring
metric: samples
Add pivot or filter...
68,745 (100.00%)
TodoMVC-jQuery
NewsSite-Next
NewsSite-Nuxt
React-Stockcha...
Editor-TipTap
TodoMVC-J...
TodoMVC-J...
Charts-c...
Charts-...
TodoM...
TodoM...
Perf-Da...
TodoM...
Todo...
Tod...
Tod...
Tod...
Edi...
unknown
art_quick_generic_jni_trampoline
Java_J_N_M1Y_1XVCN
content::JNI_ContentMain_Start ...
content::RunContentProcess
ContentMainRunnerImpl::Run
content::RunOtherNamedProces...
content::RendererMain
RunLoop::Run
ThreadControllerWithMessagePu...
MessagePumpDefault::Run
non-virtual thunk to base::sequen...
base::OnceCallback<void ()>::Run...
ProxyMain::BeginMainFrame
LayerTreeHost::BeginMainFrame
WidgetBase::BeginMainFrame
WebFrameWidgetImpl::BeginM...
Page::Animate
PageAnimator::ServiceScripted...
operator()<(lambda at ../../thir...
ScriptedAnimationController::E...
FrameRequestCallbackCollecti...
V8FrameCallback::Invoke
V8FrameRequestCallback::Invo...
blink::bindings::CallbackInvoke...
V8ScriptRunner::CallFunction
Function::Call
Execution::Call

Additional low-level data sources: ETM and SPE

Query result (10 rows) - 85.7ms SELECT h.*, s.name FROM PERF_INSTRUCTION_RANGE h, frame f USING (frame_id), symbol s ON (f.symbol_set_id = s.id) WHERE start = 499699399144 LIMIT 10

id	type	cpu	utid	cycle_set_id	start	length	instruction_count	avg_cycles	frame_id	
278	perf_instruction_range	7	4	276	499699399144	16	4	8.363636363636363	1212	std::__Cr::__tree_node_base<void*>*& std::__Cr::__t std::__Cr::__map_value_compare<char const*, std::__ const*>, true>, std::__Cr::allocator<std::__Cr::__v const*>(std::__Cr::__tree_end_node<std::__Cr::__tre
280	perf_instruction_range	7	4	280	499699399144	16	4	13	1212	std::__Cr::__tree_node_base<void*>*& std::__Cr::__t std::__Cr::__map_value_compare<char const*, std::__ const*>, true>, std::__Cr::allocator<std::__Cr::__v const*>(std::__Cr::__tree_end_node<std::__Cr::__tre
283	perf_instruction_range	7	4	283	499699399144	16	4	299	1212	std::__Cr::__tree_node_base<void*>*& std::__Cr::__t std::__Cr::__map_value_compare<char const*, std::__ const*>, true>, std::__Cr::allocator<std::__Cr::__v const*>(std::__Cr::__tree_end_node<std::__Cr::__tre
466	perf_instruction_range	7	4	466	499699399144	16	4	35	1212	std::__Cr::__tree_node_base<void*>*& std::__Cr::__t std::__Cr::__map_value_compare<char const*, std::__ const*>, true>, std::__Cr::allocator<std::__Cr::__v

Query result (100 rows) - 2.4ms INCLUDE PERFETTO MODULE linux.perf.spe; SELECT * FROM linux_perf_spe_record where ts >= 711265250880000 limit 100;

ts	utid	exception_level	instruction_frame_id	operation	data_virtual_address	data_physical_address	total_latency	issue_latency	translation_latency	data_source
711265250880075	2	EL0	4867	BRANCH	0	0	9	8	0	NULL
711265250882109	2	EL0	75188	LOAD	481039752721	37495930385	14	8	1	L1D
711265250888579	2	EL0	73844	LOAD	481083917848	41576699416	2565	43	1499	DRAM
711265250890898	2	EL0	74102	BRANCH	0	0	4	3	0	NULL
711265250892852	2	EL0	938	BRANCH	0	0	4	3	0	NULL
711265250895415	2	EL0	73967	LOAD	510653862160	45780164880	10	6	1	L1D

Beyond Speedometer

CUJs exercise browser components differently

Process	Thread	Component	Benchmark / CUJ				
			JetStream	Speedometer	Page load	Scrolling	Tap/Typing
Browser	UI				critical path	critical path	TBD
	Network				critical path		
	ThreadPool						
Renderer	Main	V8	impacts score	impacts score	critical path		
		Blink		impacts score	critical path		
	Compositor				critical path	critical path	
	ThreadPool	GC/compile	impacts score	impacts score	critical path		
		Raster			critical path		
	JS workers	WASM	impacts score				
GPU					critical path	critical path	
SurfaceFlinger					critical path	critical path	

Degree of component usage (CPU-time-based)

Google

CUJs exercise browser components differently

Process	Thread	Component	Benchmark / CUJ				
			JetStream	Speedometer	Page load	Scrolling	Tap/Typing
Browser	UI				critical path	critical path	TBD
	Network				critical path		
	ThreadPool						
Renderer	Main	V8	impacts score	impacts score	critical path		
		Blink		impacts score	critical path		
	Compositor				critical path	critical path	
	ThreadPool	GC/compile	impacts score	impacts score	critical path		
		Raster			critical path		
	JS workers	WASM	impacts score				
GPU					critical path	critical path	
SurfaceFlinger					critical path	critical path	

Developing new lab workloads in 2024/25

bit.ly/loadline

Difficulties developing a page load benchmark

Relevance

Select ~5 representative sites based on **product needs** and **performance characteristics**

Analyzed ~50 popular sites in 20+ dimensions via traces; clustering similar ones.

Strive for **maximum coverage**

Metrics

General-purpose loading metrics (LCP, FCP, ..) don't work well for low # sites

Custom instrumentation to enable **site-specific** metrics tracking readiness to interact

Strive for **normal distribution**

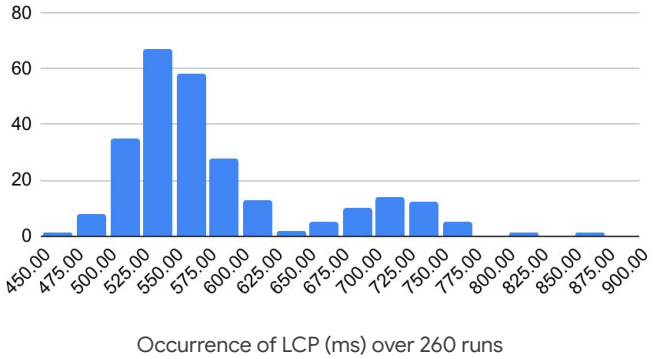
Noise

Sites evolve over time and behave differently in different geographies

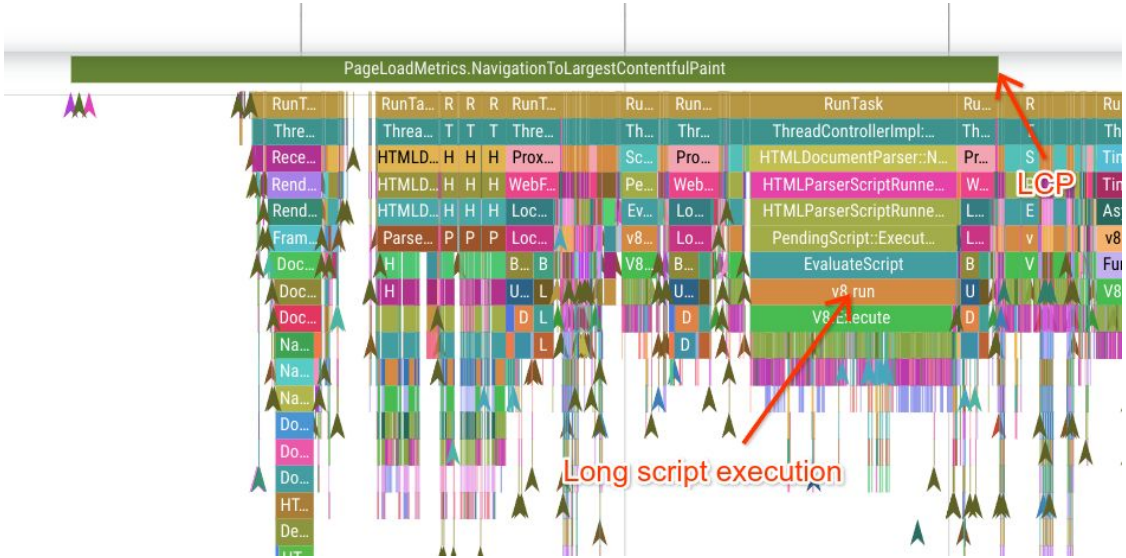
Record and replay resource loads via **WPR on device**; avoid incompatible sites

Detect 1% change in 1hr

Example: Metric bimodality



Occurrence of LCP (ms) over 260 runs

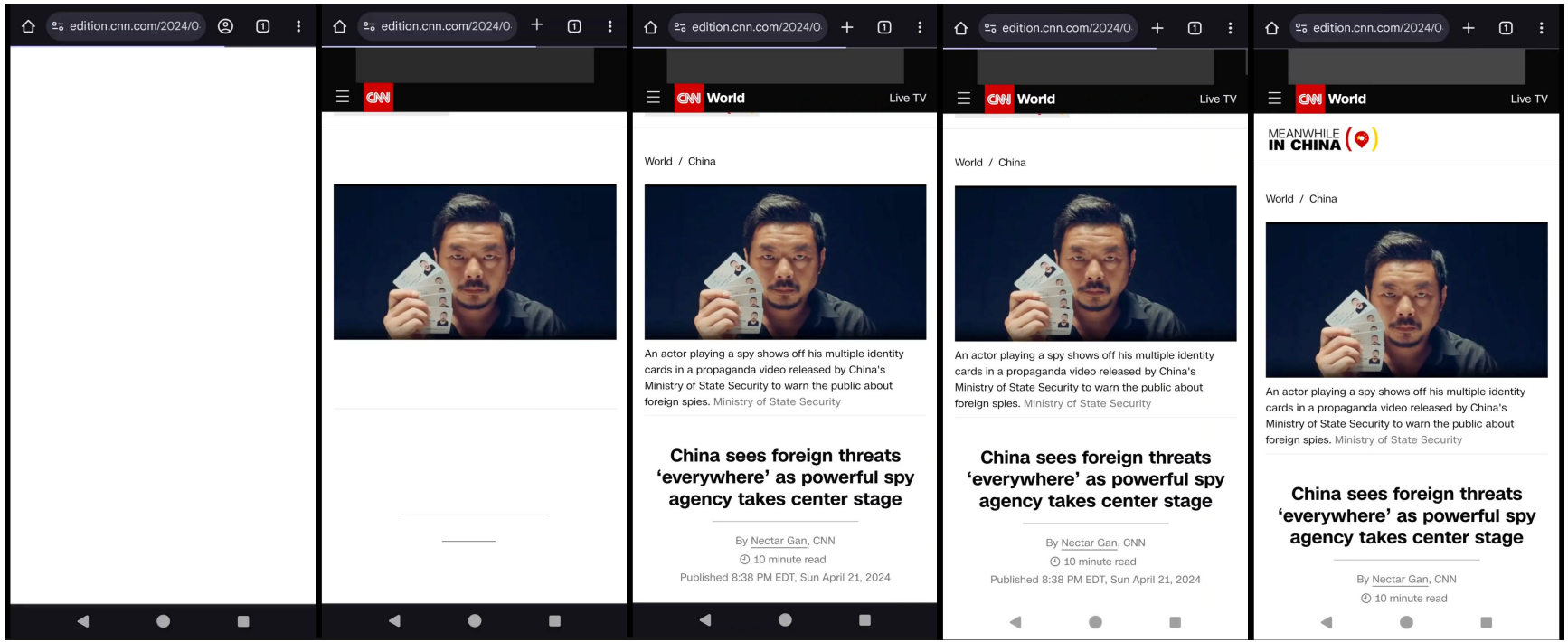


Bimodality caused by LCP-related paint running before or after a long unrelated script execution

Google

Example: Tracking a UX-relevant moment

↓ LCP ↓ Custom metric ↓ onload



Custom metrics aim to track the earliest moment that the main content is loaded and ready for interaction

Google

LoadLine stories: Phone (mobile) configuration

Page (mobile)	CUJ type / Product narrative	Performance characteristics	Metric
amazon.co.uk product page	Shopping: high usage	• average page load, large workload, large DOM/JS (but heavier on DOM) • *high on OOPIFs, input, http(s) resources, frame production*	JS ready
cnn.com article	News: high usage	• slow page load, large workload, large DOM/JS (but heavier on JS) • *high on iframes, main frame, local storage, cookies, http(s) resources*	Main content created
google.com search results	Search: largest single CUJ	• fast page load, average workload, average DOM + JS • *high on main frame, local storage, video*	LCP
globo.com homepage	News / web portal: high usage	• slow page load, large workload, small DOM, large JS • *high on iframes, OOPIFs, http(s) resources, frame production, cookies*	Cookie banner closed
wikipedia.org article	Reference work: simple/fast site	• fast page load, small workload, large DOM, small JS • *low on iframes, http(s) resources, frame production*	Last important event

LoadLine stories: Tablet (desktop) configuration

Page (desktop)	CUJ type / Product narrative	Performance characteristics	Metric
amazon.co.uk product page	Shopping: high usage	• average page load, large workload, large DOM, average JS • *high on OOPIFs, http(s) resources, frame production*	JS ready
cnn.com article	News: high usage	• slow page load, large workload, large DOM/JS (but heavier on JS) • *high on iframes, local storage, video, frame production, cookies*	Main content created
google.com search results	Search: largest single CUJ	• fast page load, low workload, low DOM + JS • *high on main frame, local storage, low on video*	LCP
youtube.com video page	Media: high usage	• slow page load, very high workload, large DOM, small/average JS • *high on video*	Cookie banner closed
docs.google.com document	Productivity: expect increased relevance, challenging workload	• slow page load, large workload, large DOM + JS (heavier on JS) • *high on main frame, font resources*	LCP

Larger focus on productivity & challenging stories for the tablet configuration

Caveats

Today, LoadLine is an internal Chromium optimization target – not (yet) capable of comparing browsers or platforms.

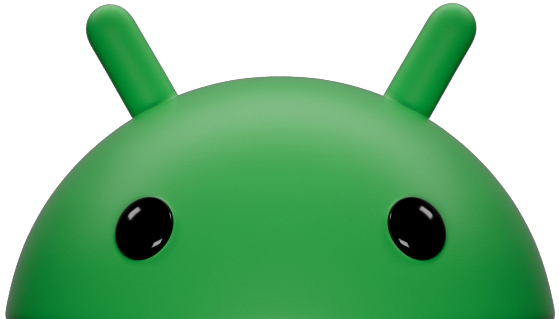
- Built for Android – site selection primarily based on mobile browsing (and Chromium only)
  - We provide phone (mobile) and tablet (desktop) workloads
- Covers fundamental CPU/GPU browser performance, but doesn't cover many networking intrinsics
  - May be extended with e.g. a traffic-shaping proxy to approximate some networking effects
- Not a micro-benchmark: Workload can change depending on device characteristics (e.g. frame rate)
  - Reflects the adaptable nature of the web and end-to-end page load performance
- Limited number of stories, some browser features are not captured
- Custom metrics are rudimentary today, so caution needed when evaluating browser behavior changes

Based on crossbench

- Setup crossben.ch
- ./cb.py loadline-phone

Available as v1.1 now

- Docs: bit.ly/loadline
- Feedback encouraged

Q&A

Thank you