Mobile at 60fps

May 13, 2016

nduca@chromium

Our goal

- 60fps on mobile

- New focus:
  mobile apps, not just web pages

Hitting 60fps

Possible only with huge life support systems

Long list of "don'ts"

Doesn't scale to application development

We need 60fps to be an **expected** and **verifiable** outcome

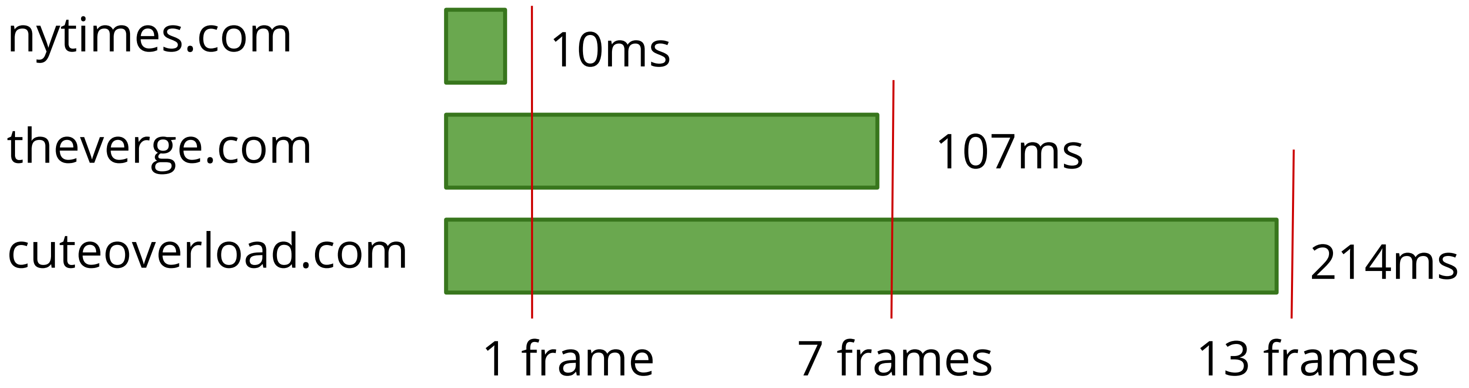
How hard could it be?

Ye Old Web Rendering

- Following lead of iOS
- Assume the web page is incurably slow to render.
- This definitely was true at the time: need great web browsers on mobile RIGHT NOW

Rasterization is SLOW

From-scratch software rasterize time is agonizingly slow:

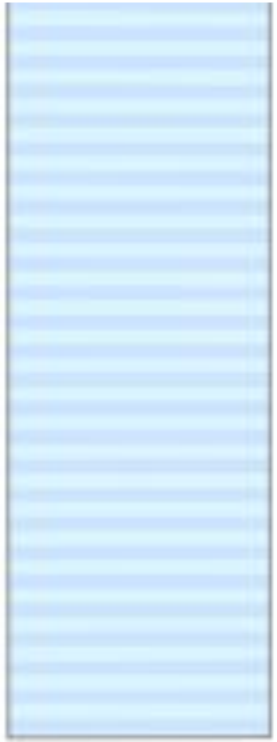


(Nexus 10)

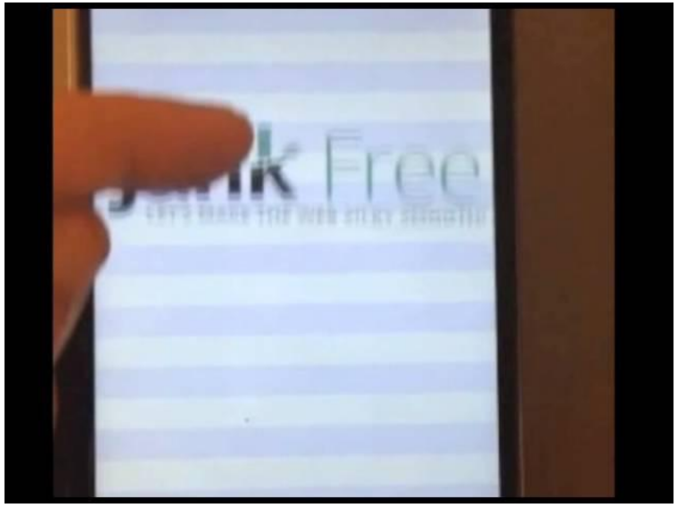
Layers + Compositing FTW

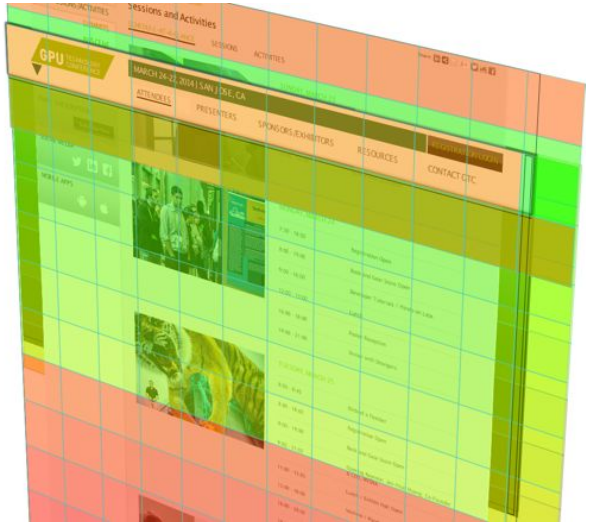
Jank Free
LET'S MAKE THE WEB SILKY SMOOTH!

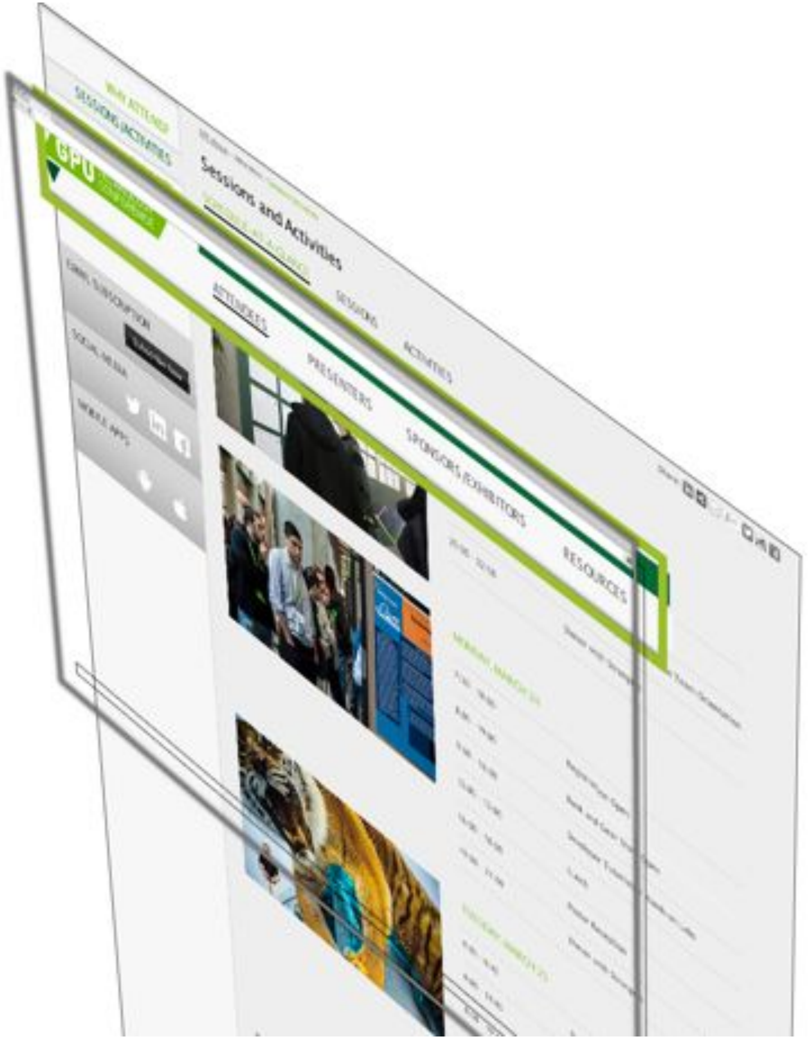
+



=







Layers are key to performance
- Fast scrolling
- Protection from raster

But, layers and scrolling are magic

Great web browser

!=

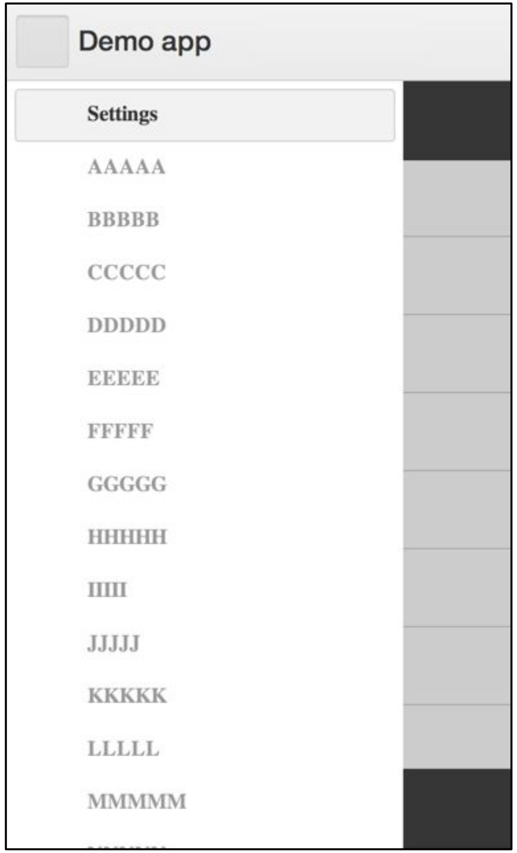
Great platform

Then we looked at mobile apps...

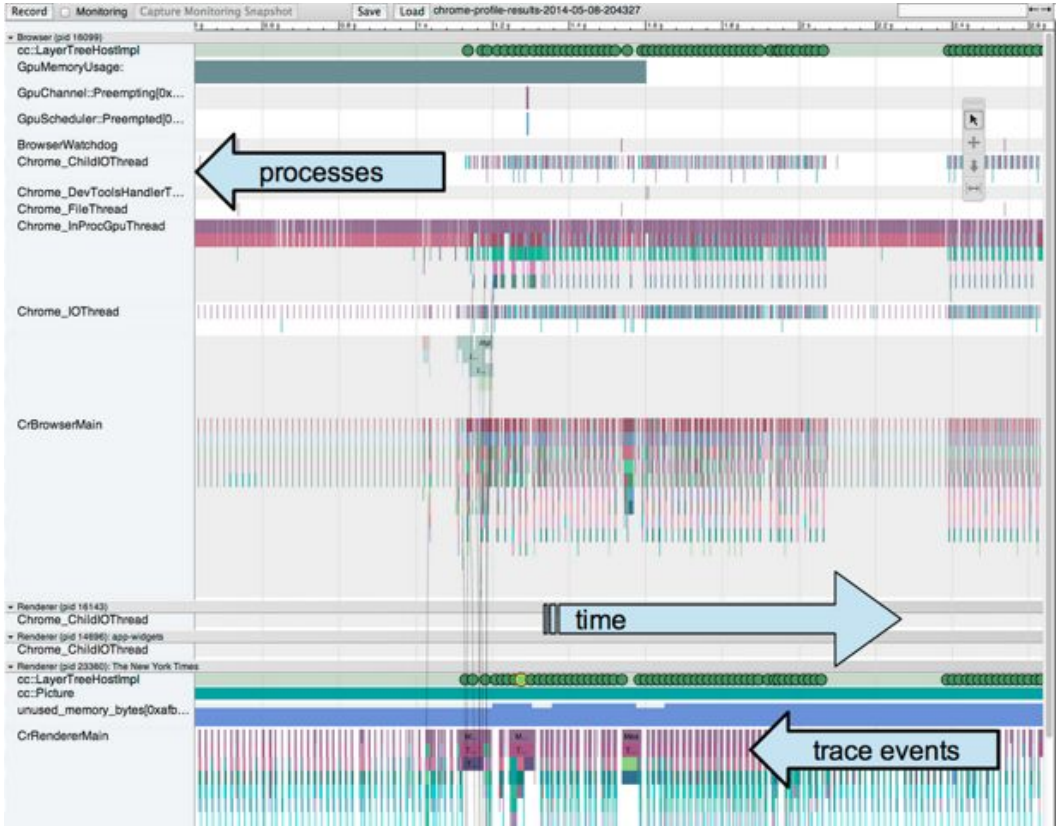
key_silk_cases

various polymer apps

some stuff that was internal and mobile-looking



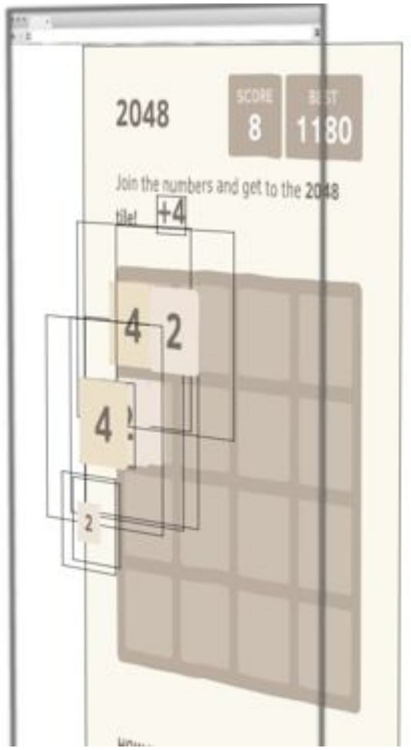
Digging in: Tracing



Digging in: Advanced tracing

"How much work, and why"?

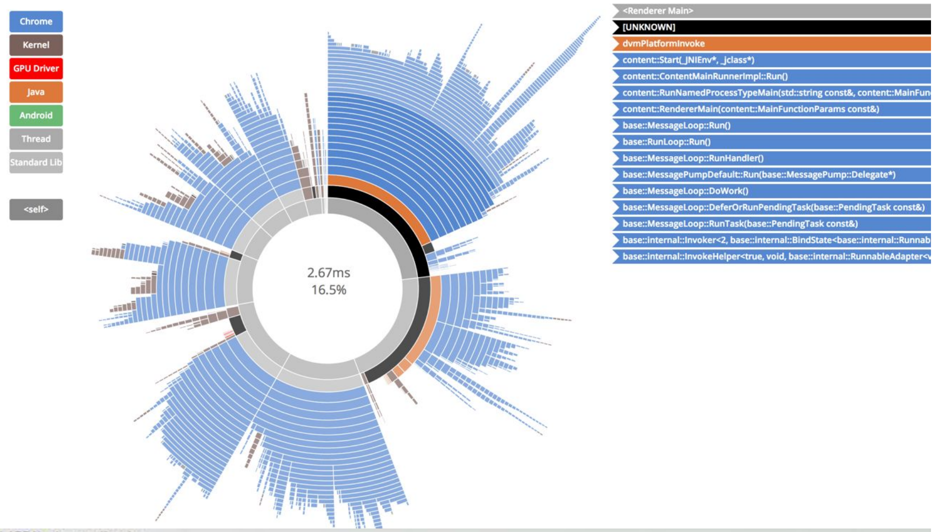
- Input latency
- Actual skia commands
- Style invalidate reasons
- Repaint reasons
- Full compositor frames



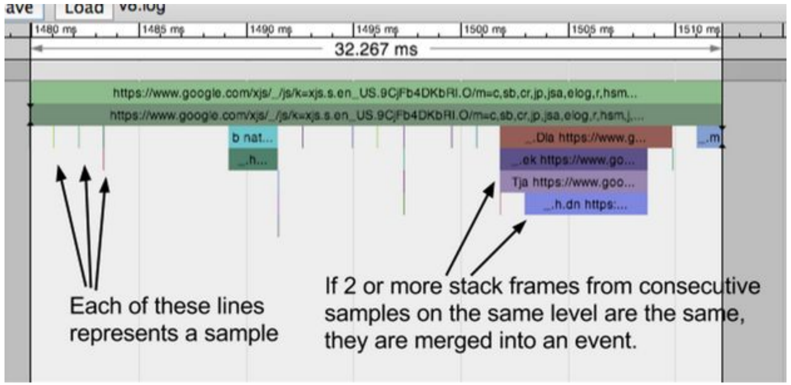
Digging in: Sampling profilers

Hard to use correctly

Better tools WIP

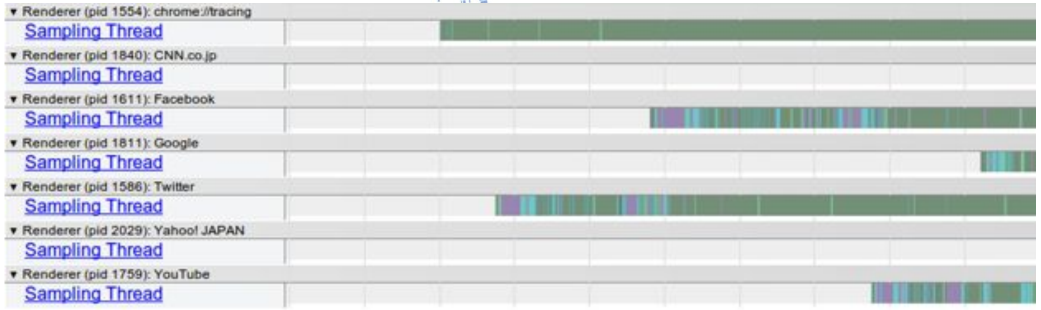


```
<Renderer Main>
[UNKNOWN]
dvmPlatformInvoke
content::Start(_JNIEnv*, _jclass*)
content::ContentMainRunnerImpl::Run()
content::RunNamedProcessTypeMain(std::string const&, content::MainFun
content::RendererMain(content::MainFunctionParams const&)
base::MessageLoop::Run()
base::RunLoop::Run()
base::MessageLoop::RunHandler()
base::MessagePumpDefault::Run(base::MessagePump::Delegate*)
base::MessageLoop::DoWork()
base::MessageLoop::DeferOrRunPendingTask(base::PendingTask const&)
base::MessageLoop::RunTask(base::PendingTask const&)
base::internal::Invoker<2, base::internal::BindState<base::internal::Runnab
base::internal::InvokeHelper<true, void, base::internal::RunnableAdapter<v
```


ave
Load
v8.log
1480 ms
1485 ms
1490 ms
1495 ms
1500 ms
1505 ms
1510 ms
32.267 ms
https://www.google.com/xjs/_/js/k=xjs.s.en_US.9CjFb4DKbRI.O/m=c,sb,cr,jp,jsa,elog,r,hsm...
https://www.google.com/xjs/_/js/k=xjs.s.en_US.9CjFb4DKbRI.O/m=c,sb,cr,jp,jsa,elog,r,hsm,j...
b nat...
_.h...
_.Dla https://www.g...
_.ek https://www.go...
Tja https://www.goo...
_.h.dn https:...
_.m

Each of these lines represents a sample

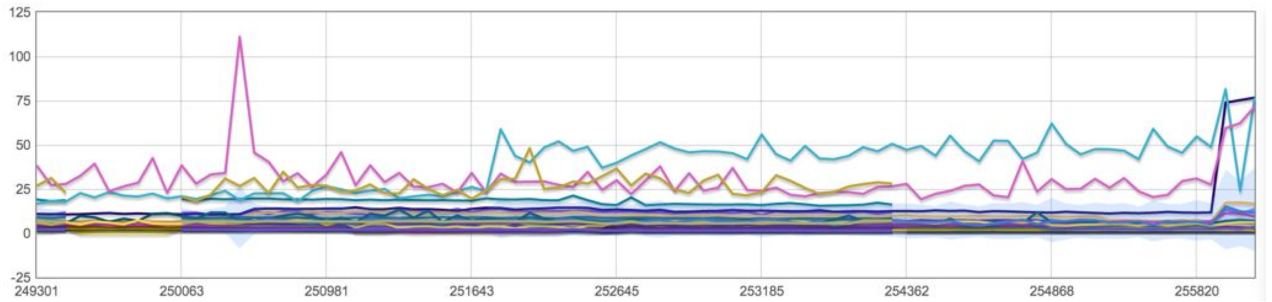
If 2 or more stack frames from consecutive samples on the same level are the same, they are merged into an event.


▾ Renderer (pid 1554): chrome://tracing
Sampling Thread
▾ Renderer (pid 1840): CNN.co.jp
Sampling Thread
▾ Renderer (pid 1611): Facebook
Sampling Thread
▾ Renderer (pid 1811): Google
Sampling Thread
▾ Renderer (pid 1586): Twitter
Sampling Thread
▾ Renderer (pid 2029): Yahoo! JAPAN
Sampling Thread
▾ Renderer (pid 1759): YouTube
Sampling Thread

Content & Benchmarks

tools/perf/run_benchmark

- thread_times.key_silk_cases
- smoothness.key_silk_cases
- rasterize_and_record_micro.key_silk_cases

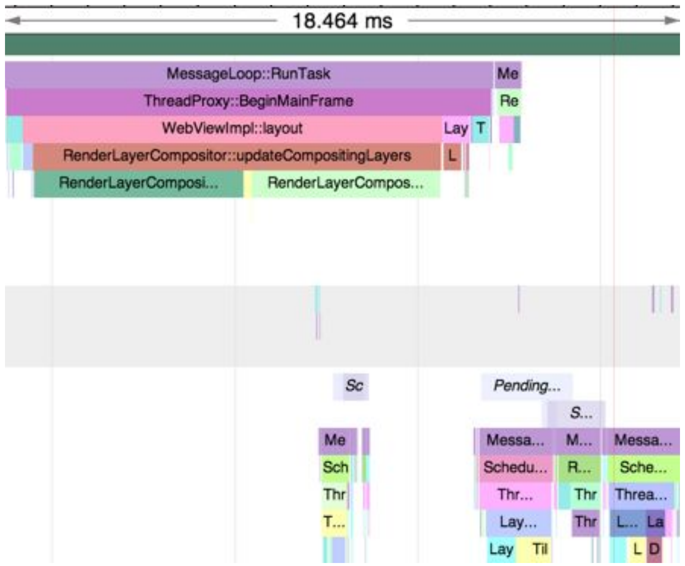


Our goal

- 60fps on mobile
- Mobile apps, not just web pages

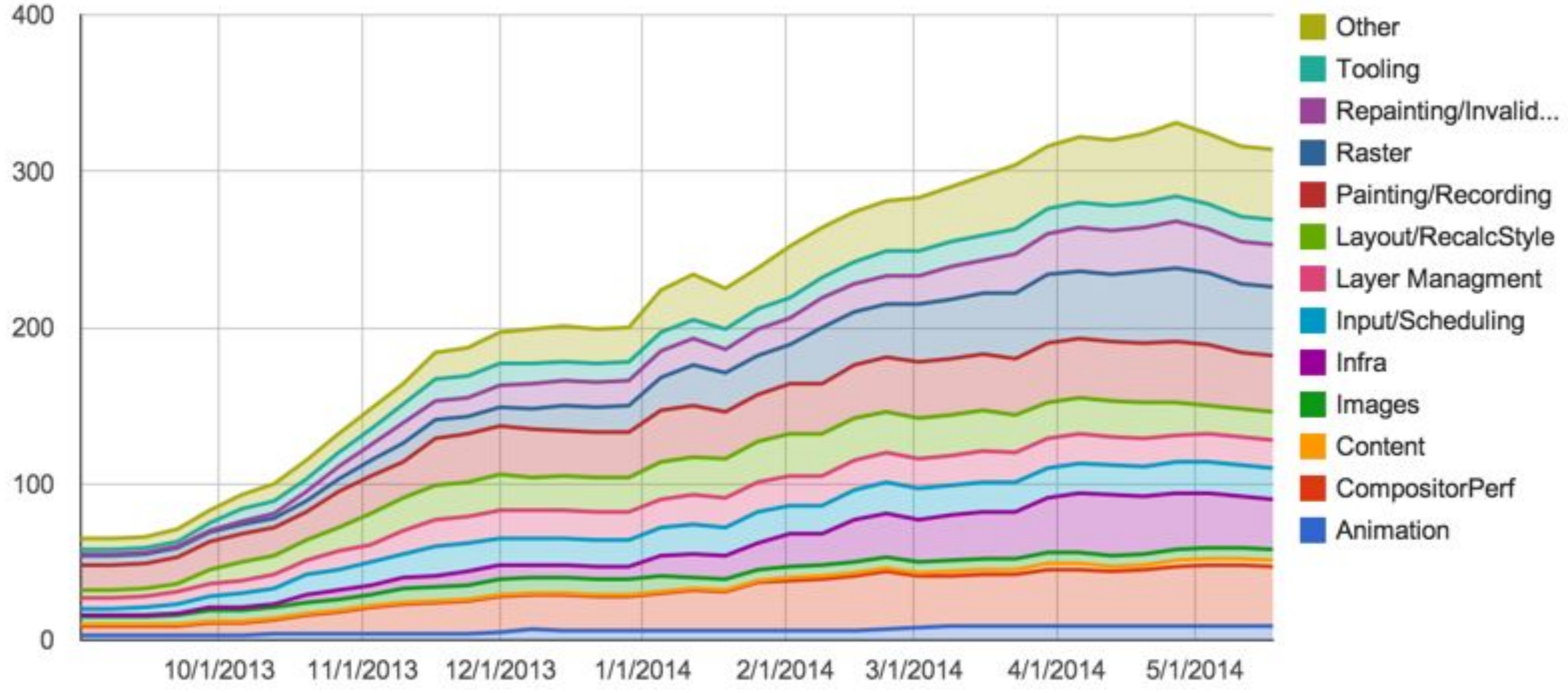
Oh, the horror!

- 30ms+ main thread tasks were common
- Hundreds of ms of accidental repaints
- Compositor+GPU was >10ms/frame
- Input & scrolling system was working against us, not for us
- Layer system was working against us in dozens of ways



Chrome 33: 18ms / frame

crbug.com Hotlist=Jank

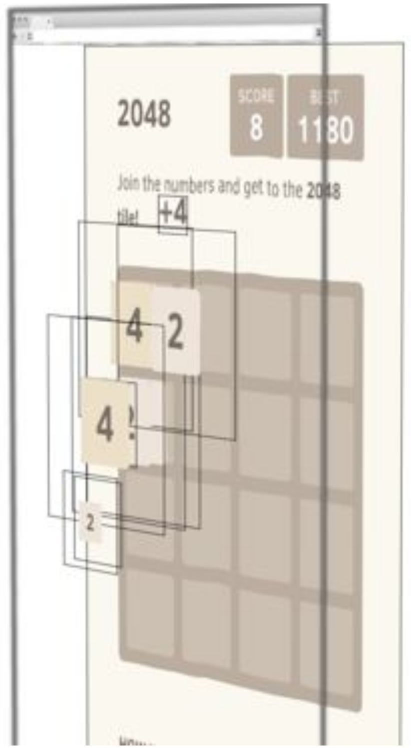


Blink

Layout is surprisingly cheap

A sampling of awesomeness

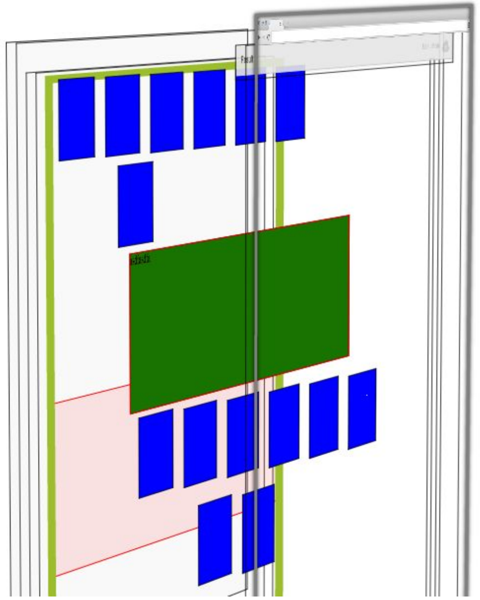
- layer maintenance
- state machine & chicken-egg
- recalcStyle / targeted style invalidation
- inline style & style parsing
- recording/RenderObject::Paint



Repaint storms: Enemy #1

Remember how costly repaint is?

- Tease apart repaint and layout in StyleDiff
- Repaint-after-layout
- Piles and piles o' bugs



Web Animations

Using CSS transitions/animations is horribly fragile in large teams

People were making things *worse* by trying to use css.

Element.animate & cancellation super urgent

Touch platform

Full of little irrational behaviors that, as a whole, are death

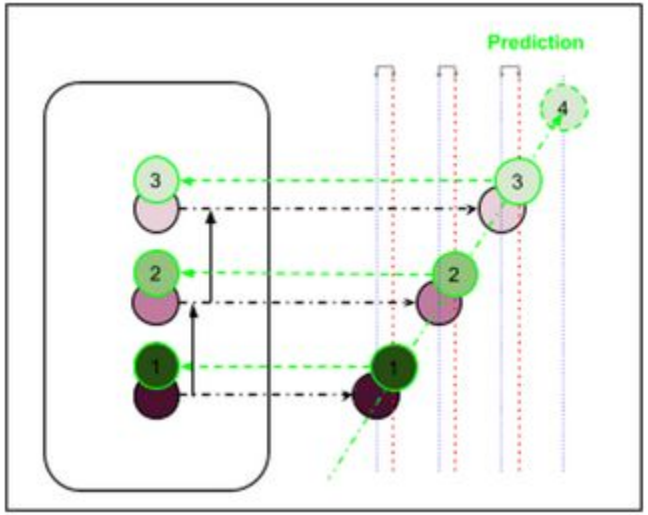
- Pull to refresh
- Hidey bars

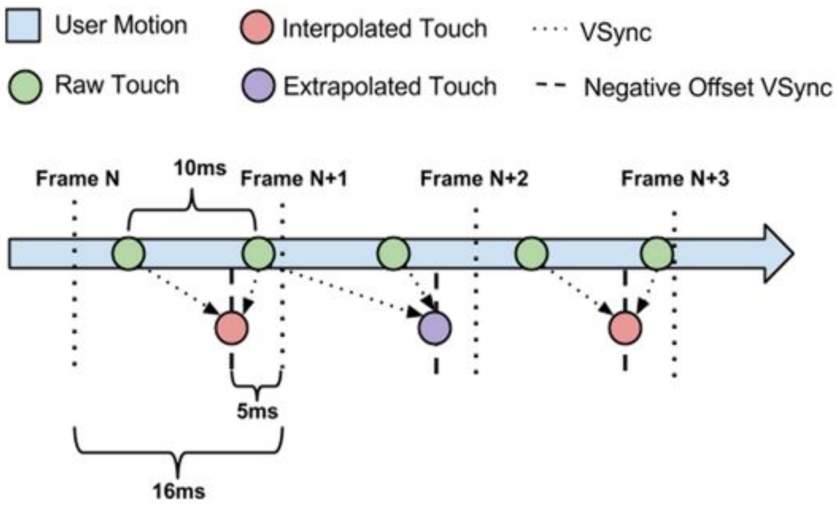
Some work

- Throttled async TouchMove model
- Best effort scheduling & onscroll-before-rAF
- Talking about overscroll event
- Omnibox & OSK hiding

Scheduling

Input, rendering and background work need to be coordinated





Compositor performance

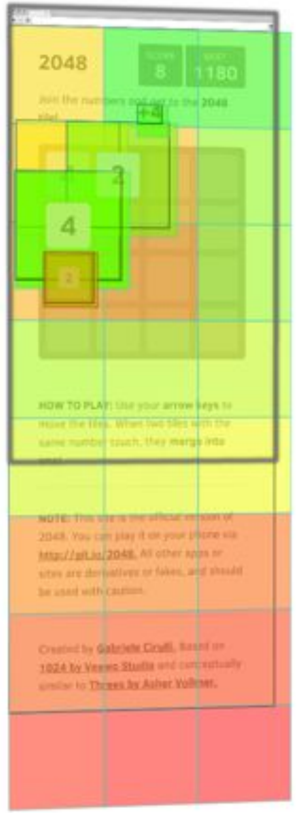
Drive down cost of compositor-side costs

- January: 12ms/frame
- Today: [7ms/frame]() ([and tracked on chromeperf]())

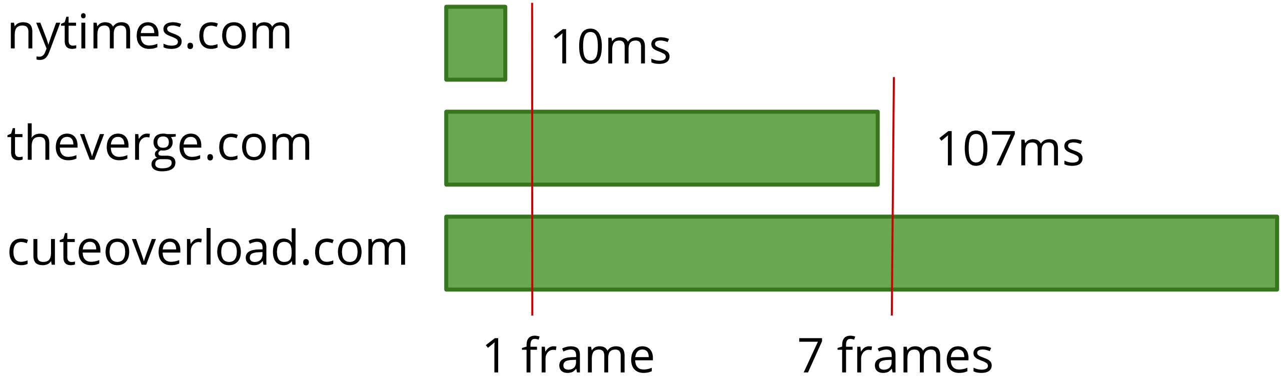
Key projects:

- TileManager overhaul
- Texture upload
- Command buffer & GPU-process tuning

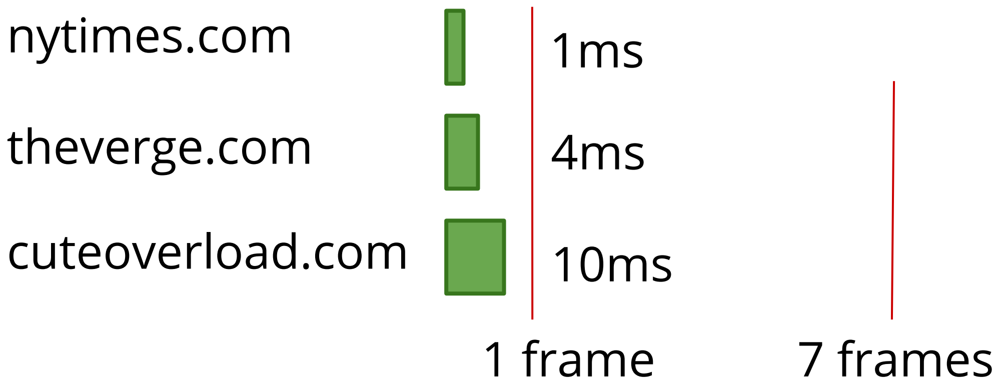
Lot o' micro… :)



Pause: we got here because of this….



What if raster was like this?



GPU Rasterization

- Use GL to rasterize

- Some sites benefit hugely
- Lots of devilish corner cases
  - Content axis
  - Device axis

- Don't know all the corner cases until we try

-

GPU Rasterization

- Focusing on mobile first

- And, only for sites with

  ```
  <meta name="viewport"
  content="width=device-width,
  minimum-scale=1.0,
  initial-scale=1.0,
  user-scalable=yes">
  ```

- GPU rasterization will be everywhere, eventually!

You can help!

    --force-gpu-rasterization

  Does your new device survive?

  Found a page slower than sw raster? Profile it, file a bug.

What else is "incurably janky"?

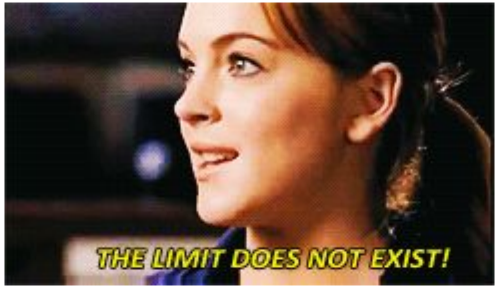
… and is it really incurable?

Our goal

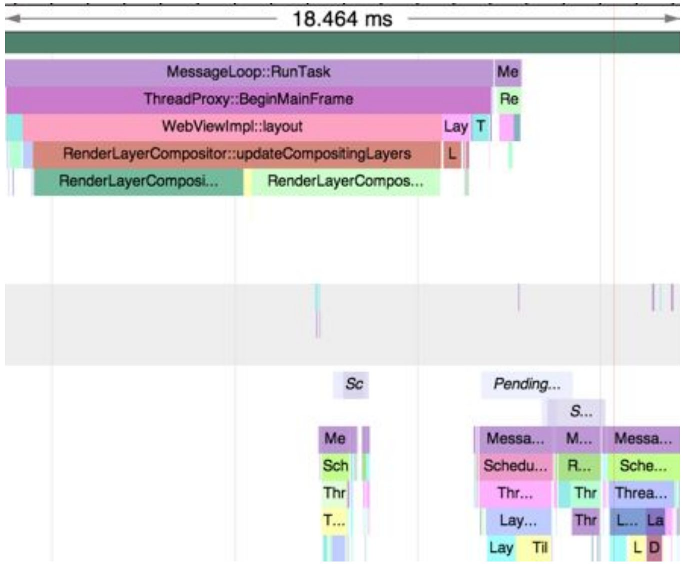
- 60fps on mobile

- Mobile apps, not just web pages

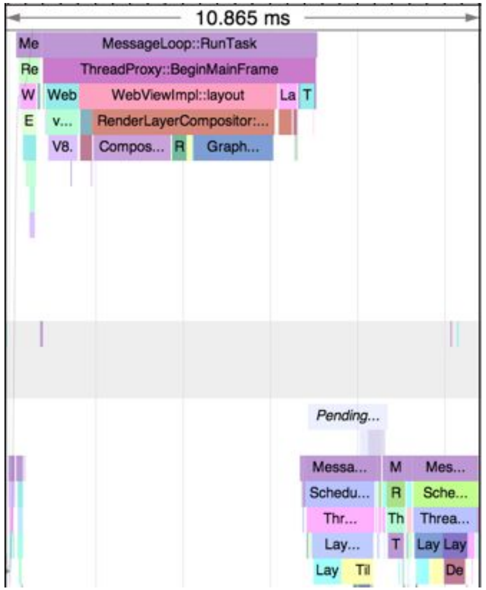
We're getting there!



Chrome 30: ಠ_ಠ



Chrome 33: 18ms / frame



Chrome 36: 10ms / frame

Threaded vs Non Threaded

The compositor thread is a blessing and a curse

- Really awesome when it works
- Super narrow fast path
- ~2-4ms overhead

Kill it? Keep it?

- Nothing inherently wrong with a fast path
- **IF main threaded solutions exist for all use cases**

^ not true today!

[some] Guiding principles

- All effects implementable with platform primitives
- 60fps is an expected behavior of the platform

- Laser focus on mobile content
- Do work proportional to what changed // visible
- Silo busting: lots of perf loss between subsystems
- Tools make everyone more effective

Three grand challenges for 2014

- Pull to refresh as good as the pros
- Jank free, checkerboard free data-driven infinite scroll
- Dump a blob of js+html into a div and animate it in at 60fps