DEVIEW 2021

웹과 브라우저를 위한 성능 프로파일링과 개선

박정우 NAVER Whale

1. 렌더링 파이프라인과 함께 웹 로딩 성능 개선하기

2. Vsync와 함께 춤을, 안드로이드 브라우저 스크롤 개선하기

3. Histograms / Tracing으로 브라우저 실행 성능 파악하기

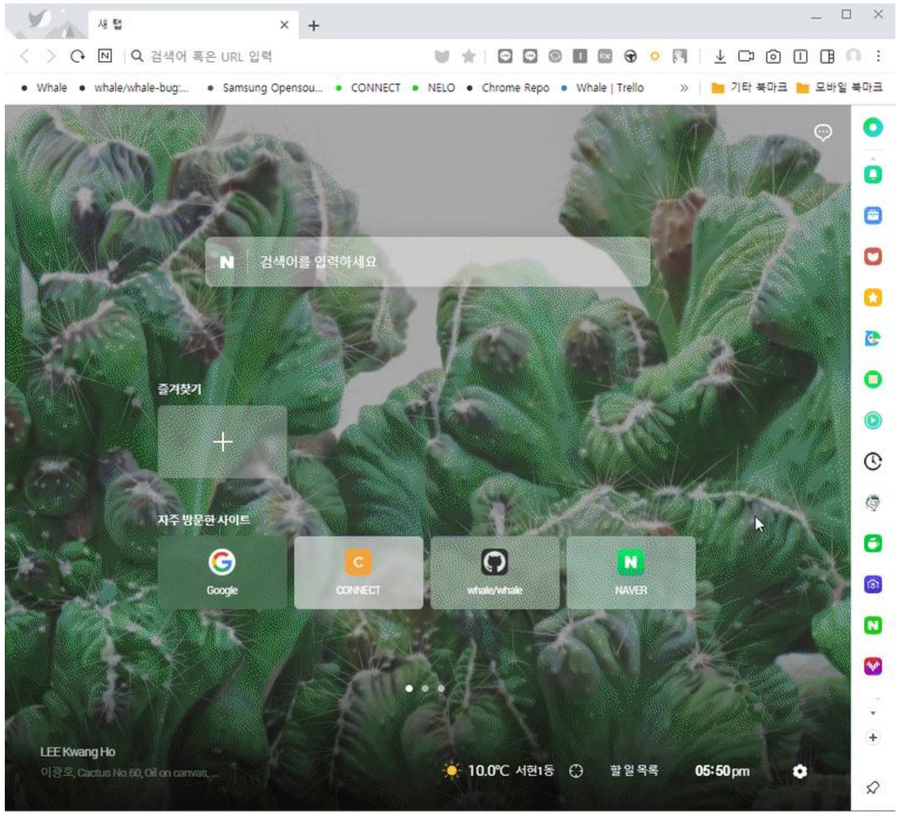
DEVIEW 2021

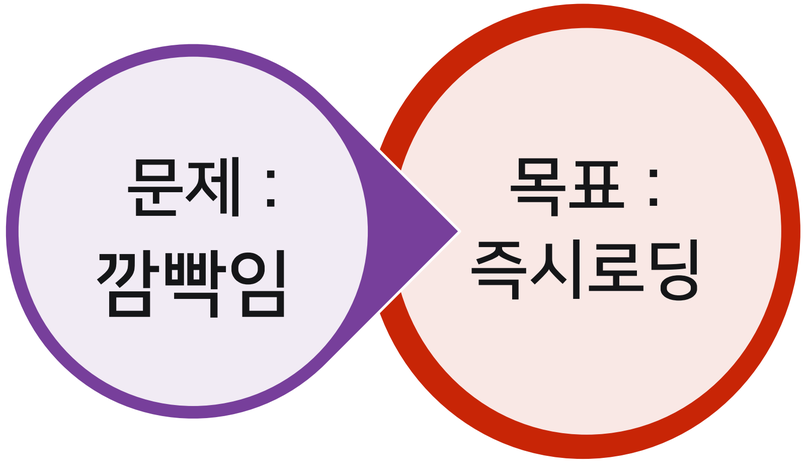
렌더링 파이프라인과 함께 웹 로딩 성능 개선하기

1.1 새 탭 웹컨텐츠 로딩 성능

DEVIEW 2021

✓ 새 탭 클릭 시 깜빡임 현상





새 탭이 뜰 때 월페이퍼가 이미 렌더링된 느낌

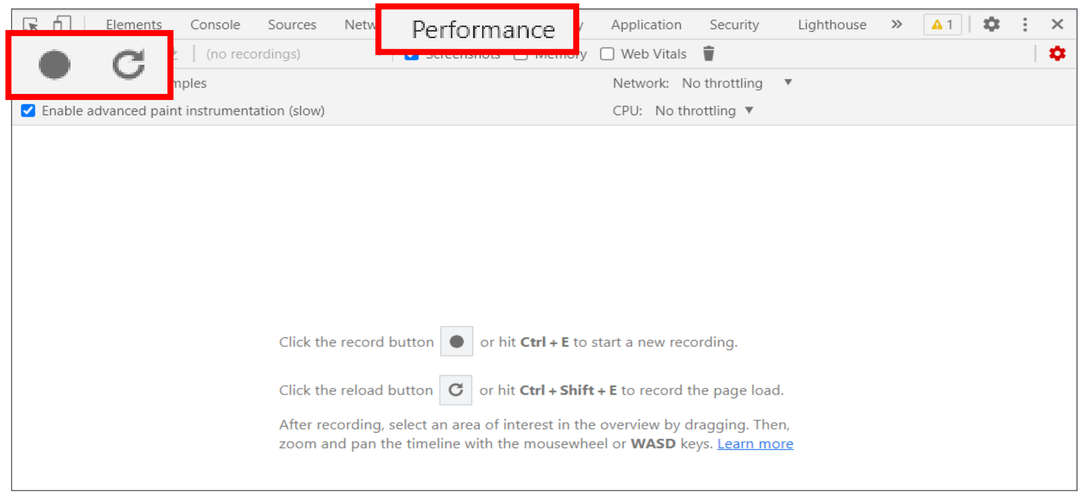
1.2 웹 성능 측정 방법들

DEVIEW 2021

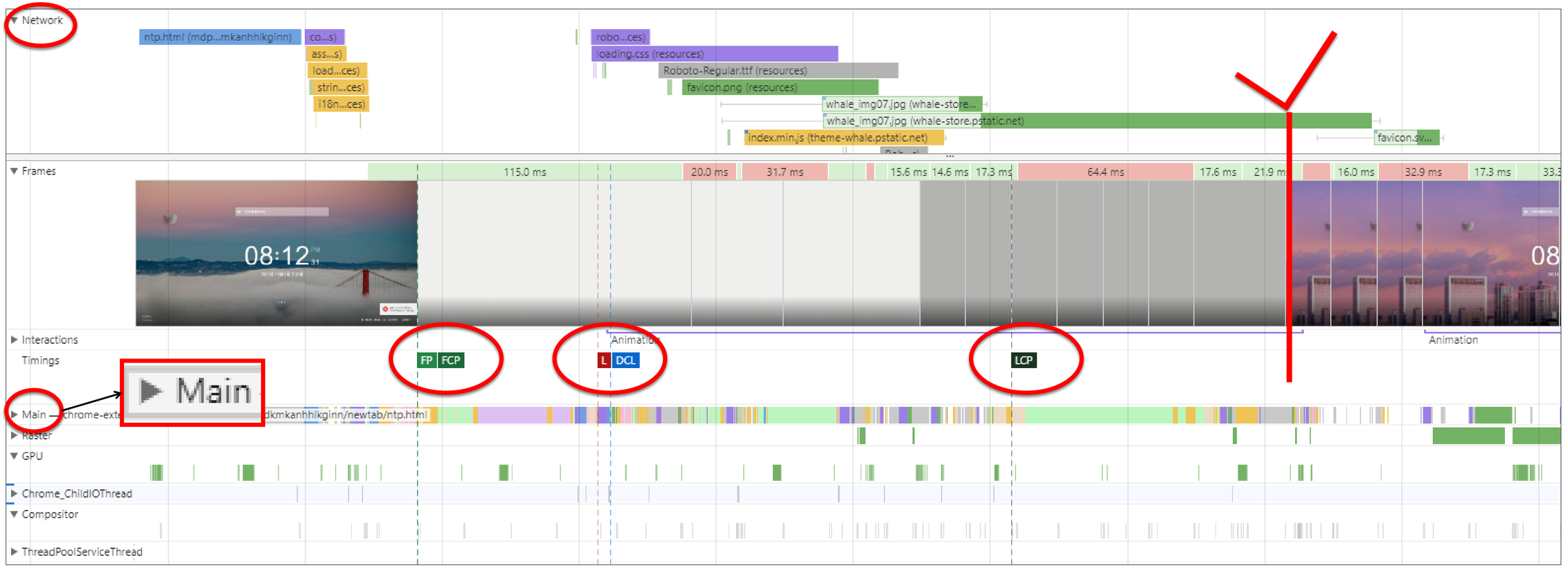
- Devtools Performance -> Detail approach
- Lighthouse -> Report
- Performance API -> Record

F12 Key

Record / Refresh and recording



1.3 새 탭 페이지 Performance



Network 탭 : 다운로드 요청과 완료 시간을 볼 수 있음

Main 탭 : 렌더러의 메인 스레드로, 다양한 렌더링과 더불어 스크립트 실행도 함

FP, FCP, L, DCL, LCP : 로딩 시 하나의 기점이 되는 지표들

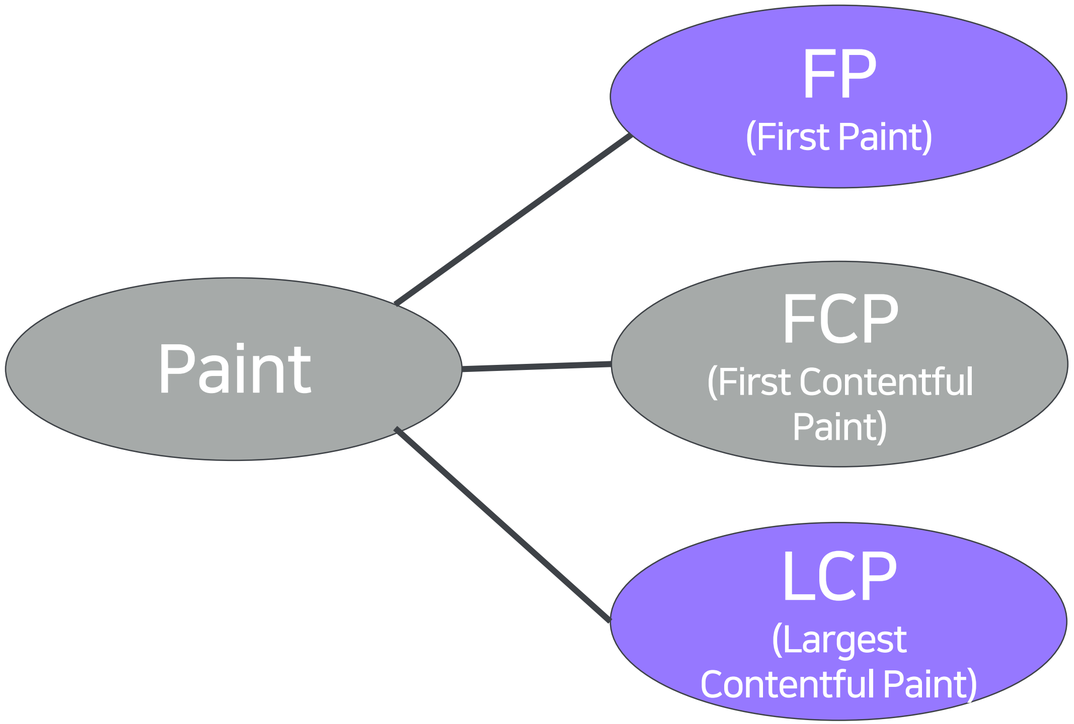
1.2 유저 메트릭

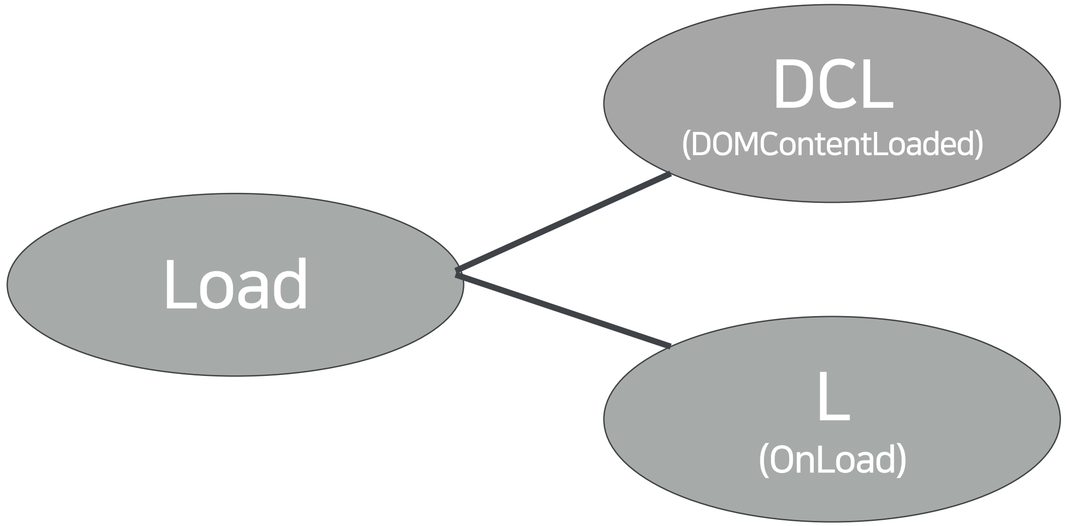
✓ 로딩에 관한 지표들

-> 페인트와 로드 카테고리로 나뉨

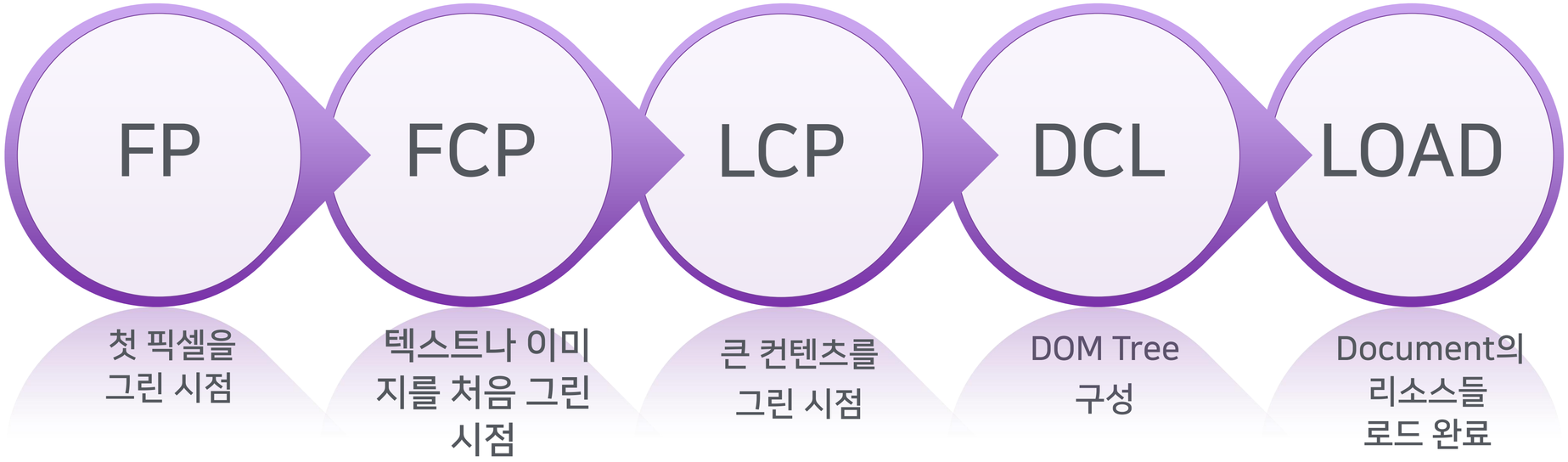
FP : First paint

LCP : Largest contentful paint





1.3 유저 메트릭



FP (First paint) : 페이지 네비게이션 후, 첫 픽셀을 그린 순간

FCP (First contentful paint) : 첫 엘리먼트를 그린 순간

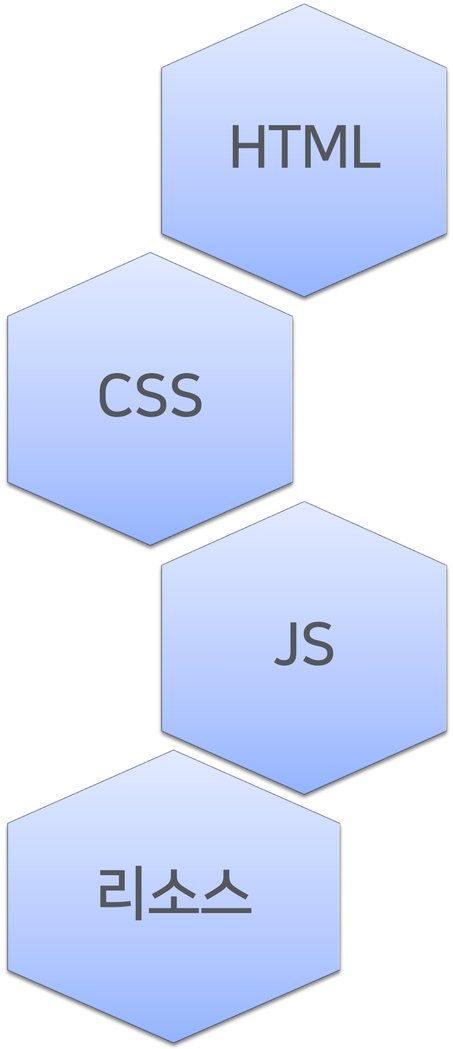
LCP (Largest contentful paint) : 가장 큰 엘리먼트를 그린 순간

DCL (Dom Content Loaded) : DOM Tree를 구성하고, 스크립트(+ defer 스크립트)를 실행 완료했을 때

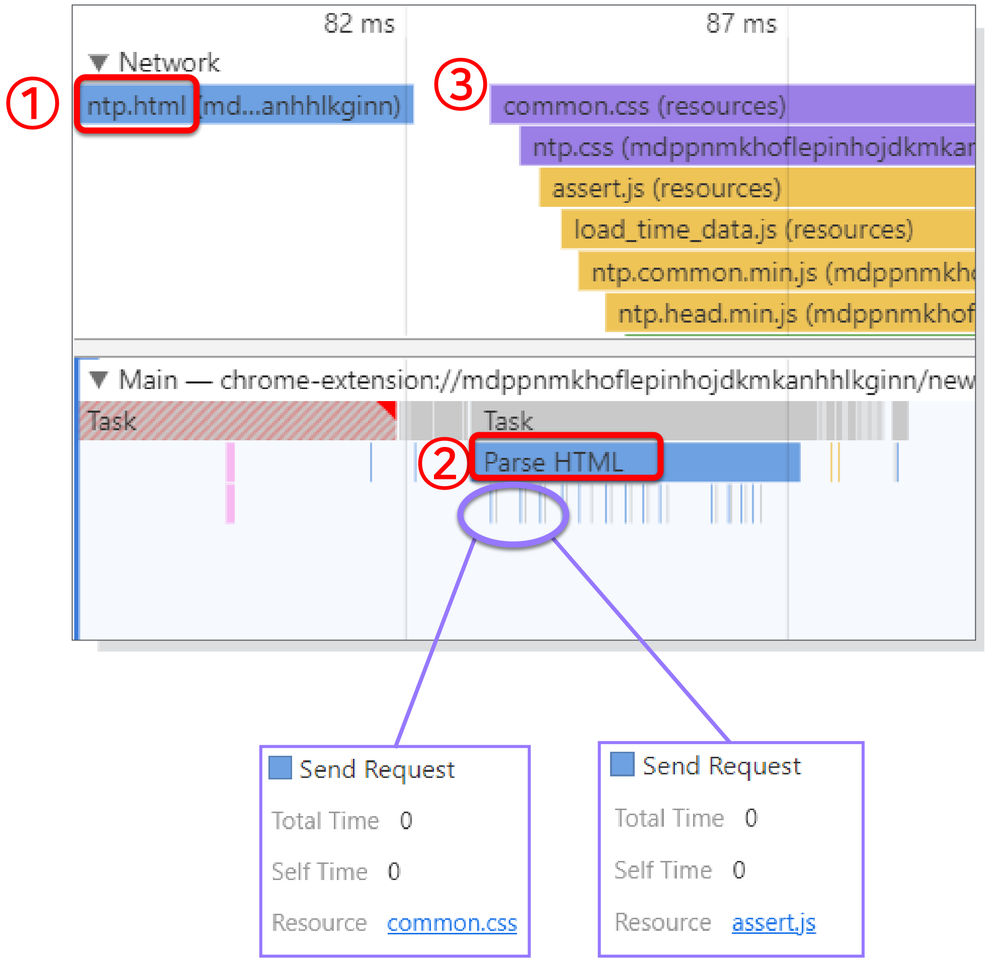
-> defer 스크립트가 없다면 FP 전에 DCL이 올 수 있음.

LOAD : 다큐먼트의 리소스들을 모두 로드 완료했을 때

1.3 First Paint로 가는 여정



웹컨텐츠들이 화면에 나타나기 까지엔
브라우저의 렌더링 파이프라인 과정이 있다!


82 ms
87 ms
Network
①
ntp.html (md...anhhlkginn)
③
common.css (resources)
ntp.css (mdppnmkhoflepinhojdkmkar
assert.js (resources)
load_time_data.js (resources)
ntp.common.min.js (mdppnmkh
ntp.head.min.js (mdppnmkhof
Main — chrome-extension://mdppnmkhoflepinhojdkmkanhhlkginn/new
Task
Task
②
Parse HTML
Send Request
Total Time 0
Self Time 0
Resource common.css
Send Request
Total Time 0
Self Time 0
Resource assert.js

DOM

DEVIEW 2021

```
<body>
  <h1> Hello Deview! </h1>
  <div>
    <p> see you </p>
    <p> at screen </p>
  </div>
</body>
```

HTMLDocumentParser

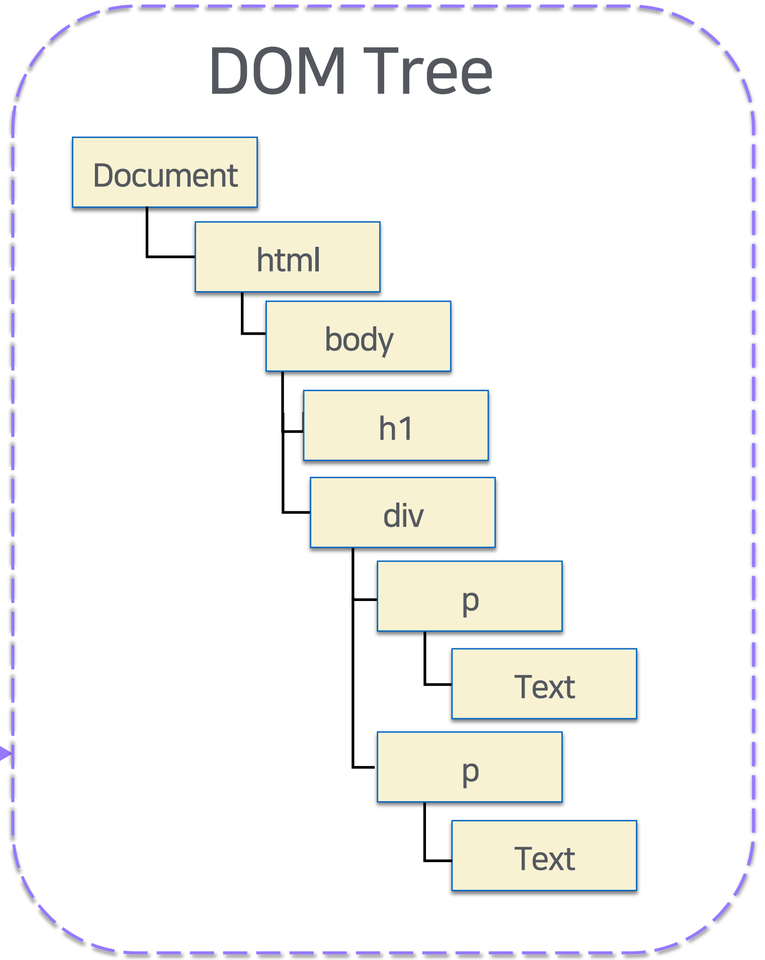
has

- HTMLTokenizer
- HTMLInputStream
- HTMLParserScriptRunner
- HTMLTreeBuilder

- Run <script>
- Queue <defer script>

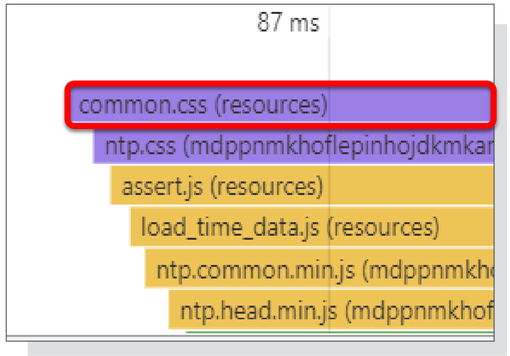
script script script

<deque> scripts_to_execute_after_parsing



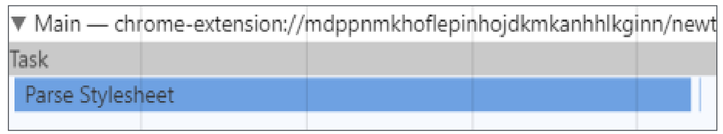
점진적으로 구성 가능

Network



Loaded

Renderer

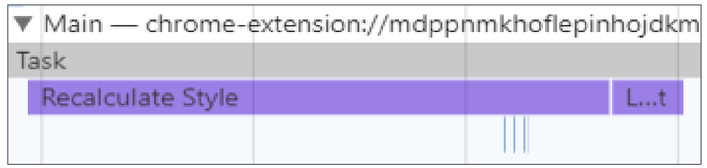


CSSParser

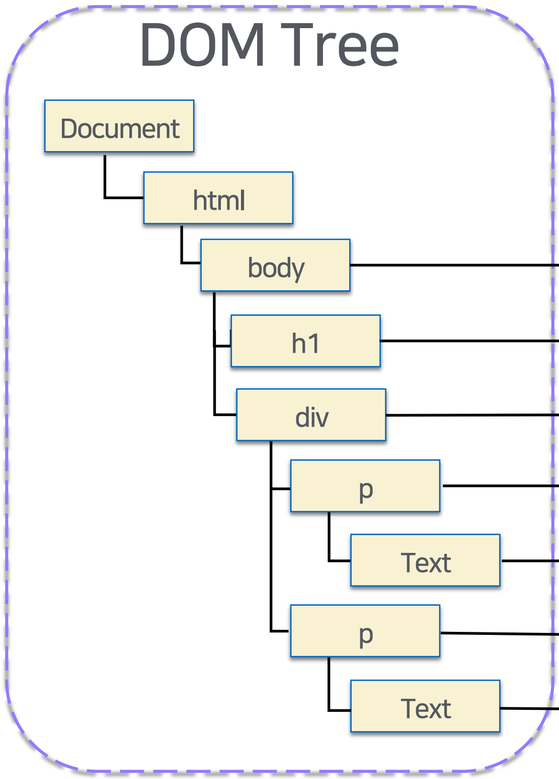
AppendRule

StyleSheetContents

Renderer

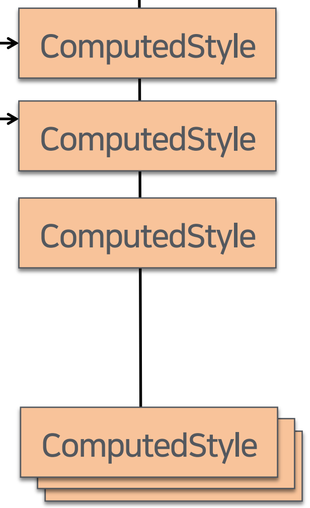


DOM Tree

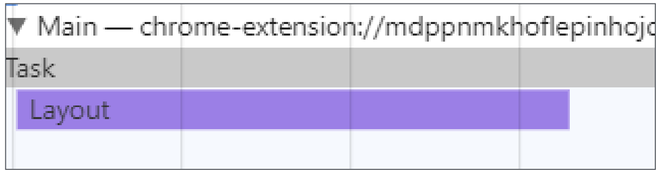


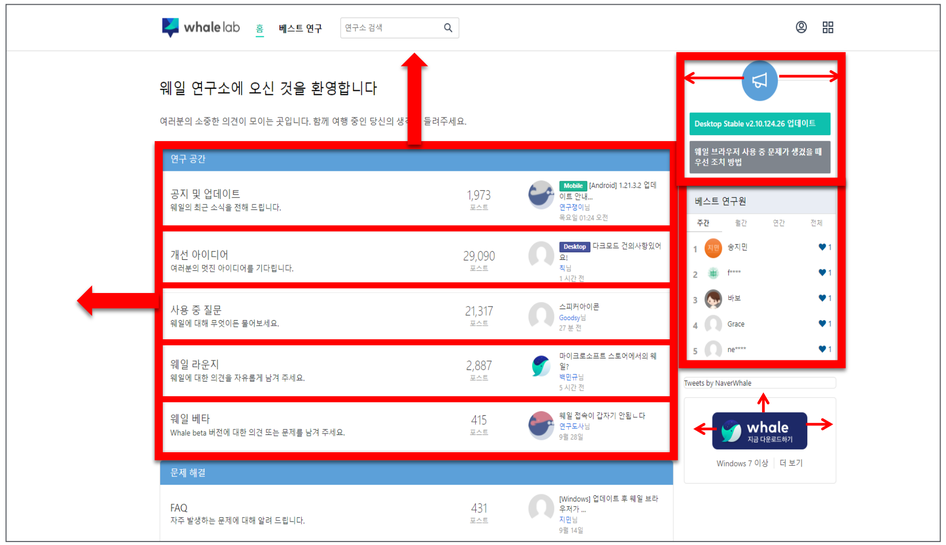
StyleResolver

Select



완성이 되야 다음 단계가 가능

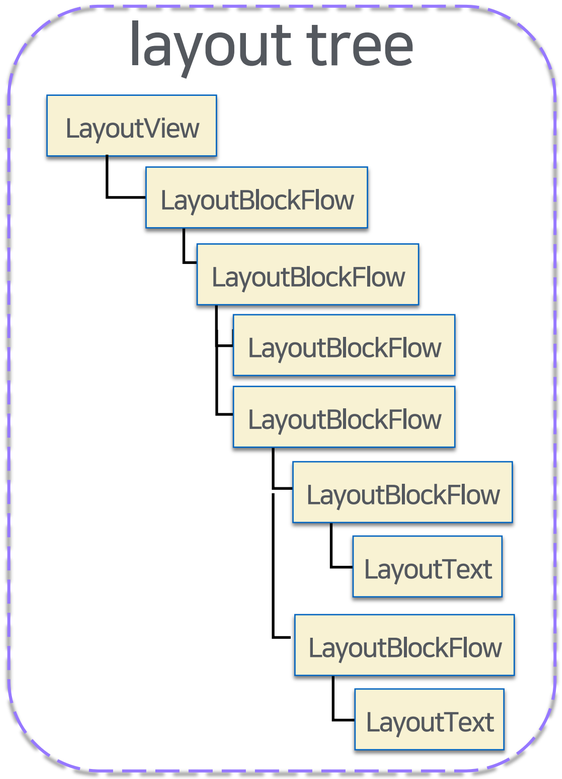




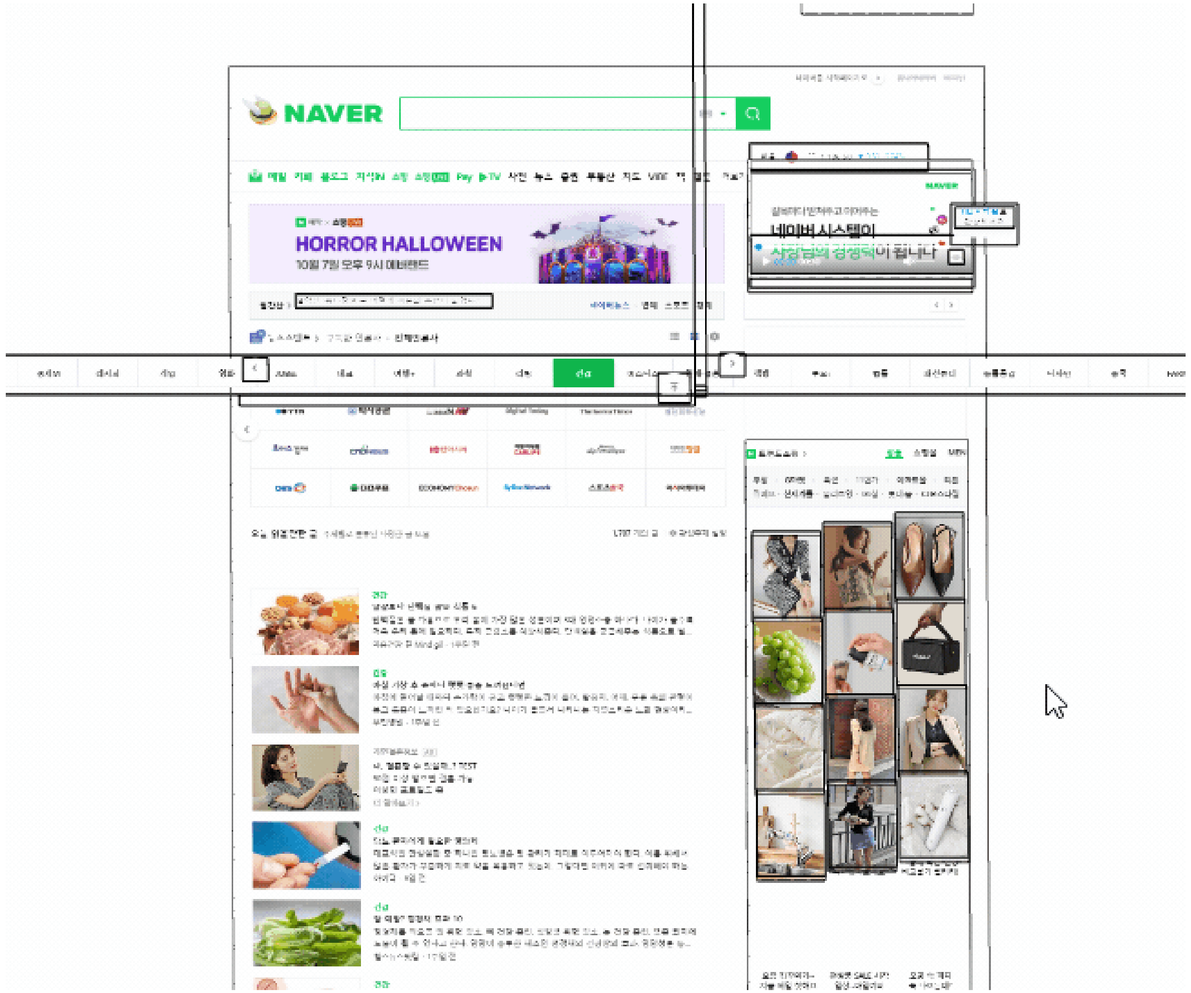
DOM tree (1~100%)

+

CSS style rules (100%)



-> LayoutNG



레이어 보는 법 : Devtools -> More tools -> Layers

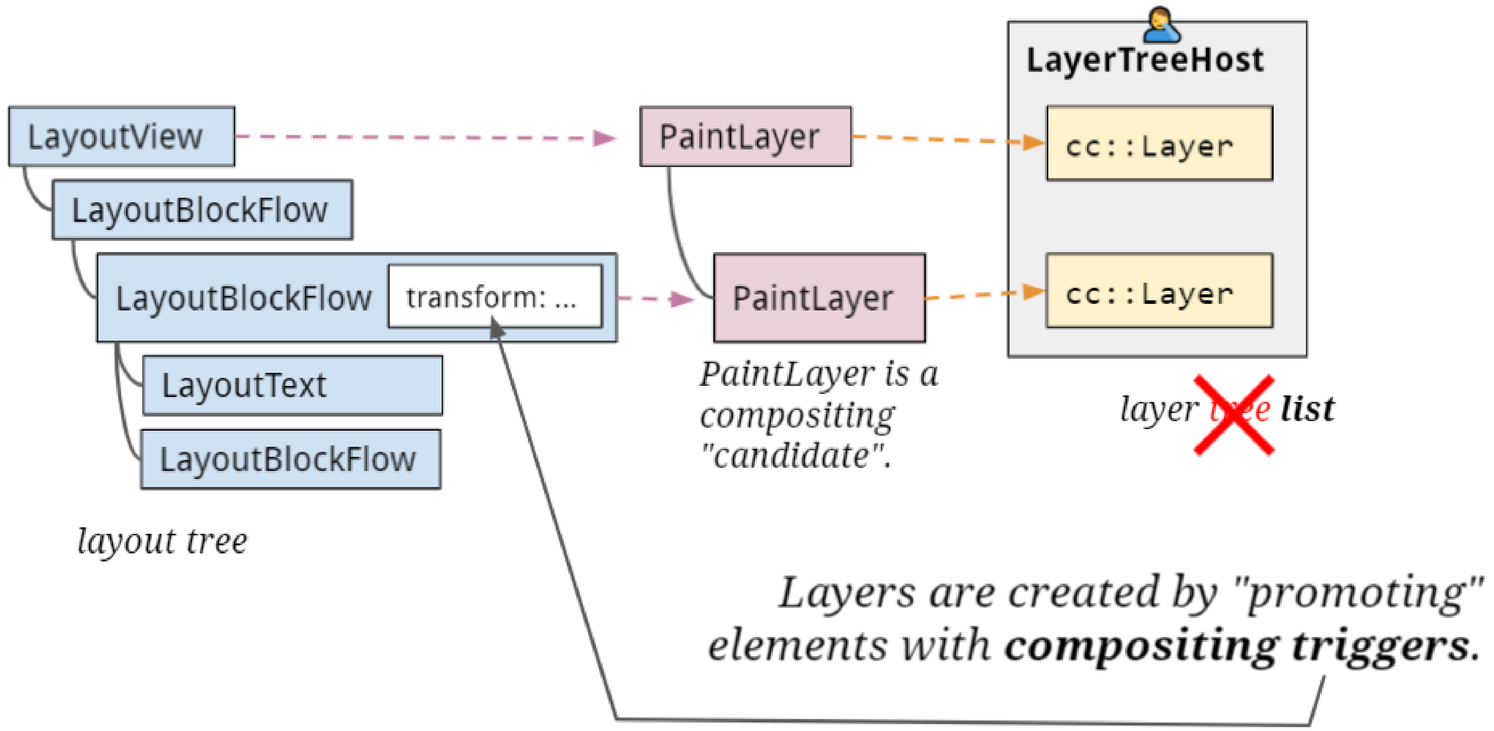
Layer 생성 조건

1. It's the root object for the page.
2. CSS position properties (relative, absolute)
3. Has overflow, an alpha mask or reflection
4. CSS filter
5. CSS 3D Transform, Animations
6. <canvas>, <video>
7. will-change
8. Browser internal layers

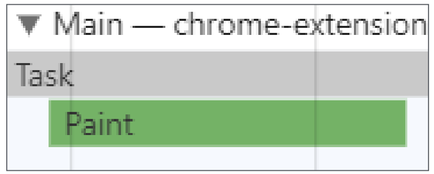
[웨일 이형욱: 웹브라우저 성능의 모든 것]

Prepaint Paint invalidation & property tree

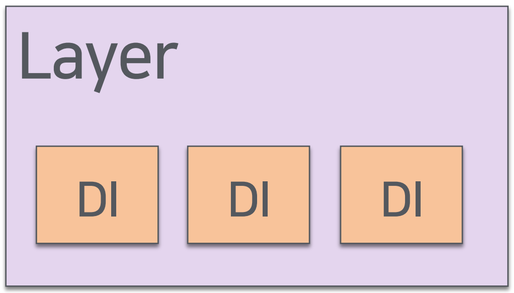
Compositing Assignment Build layer tree



[Steve Kobes : Life of Pixel]

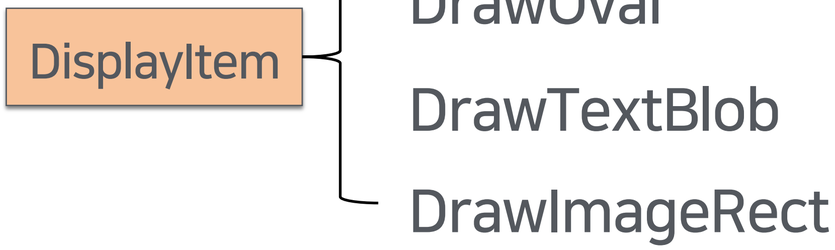


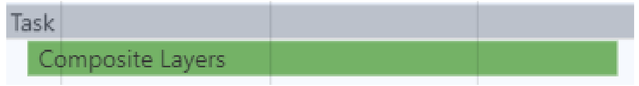
Paint is recording of display items



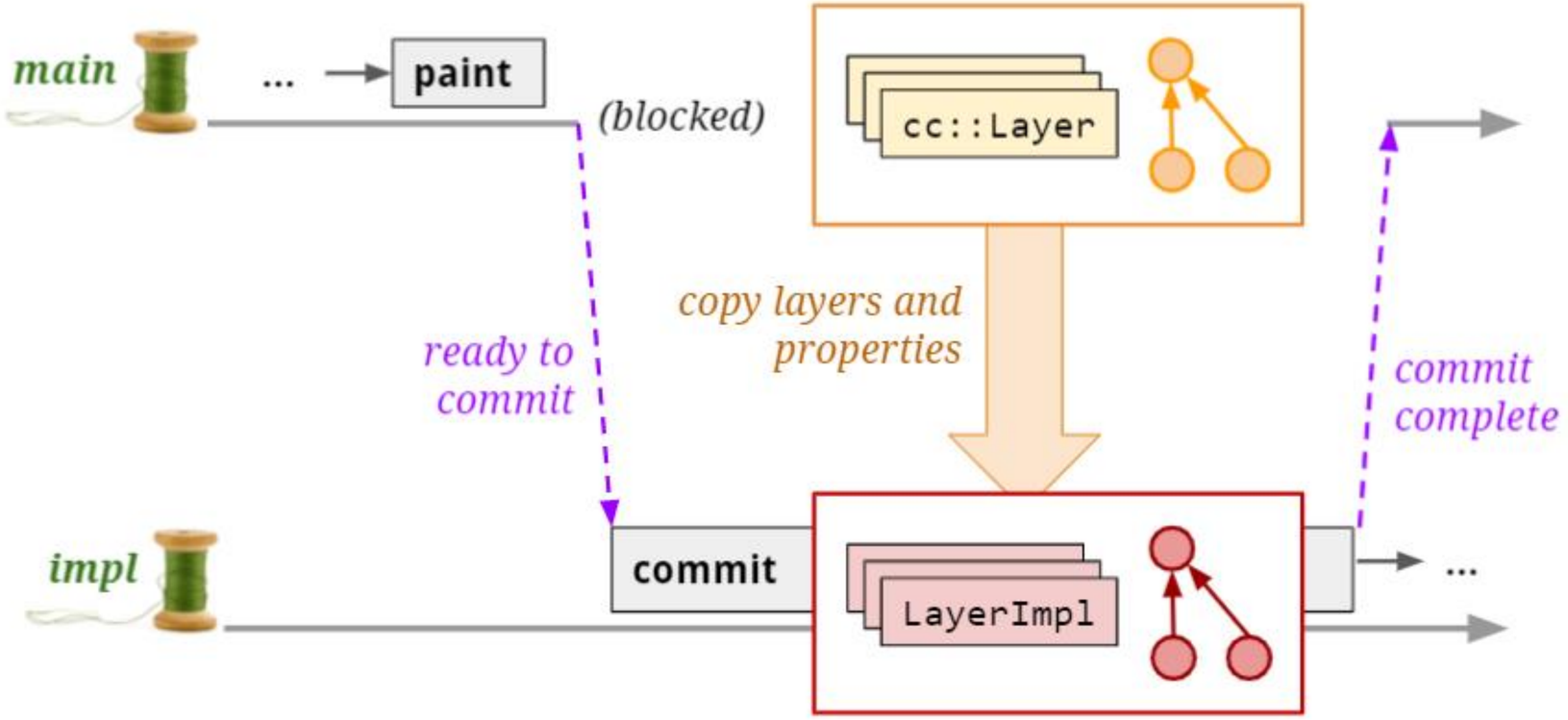
Layer is paint unit

Draw operation



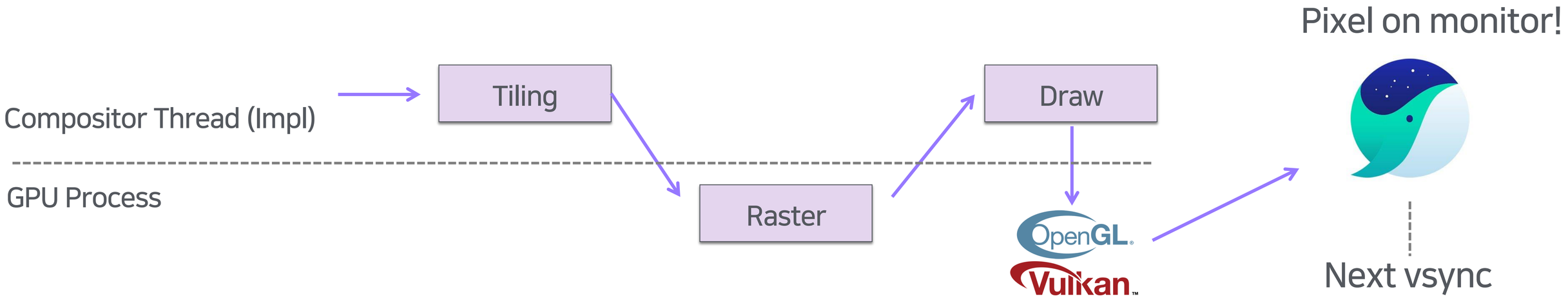


~~Composite~~ -> Commit



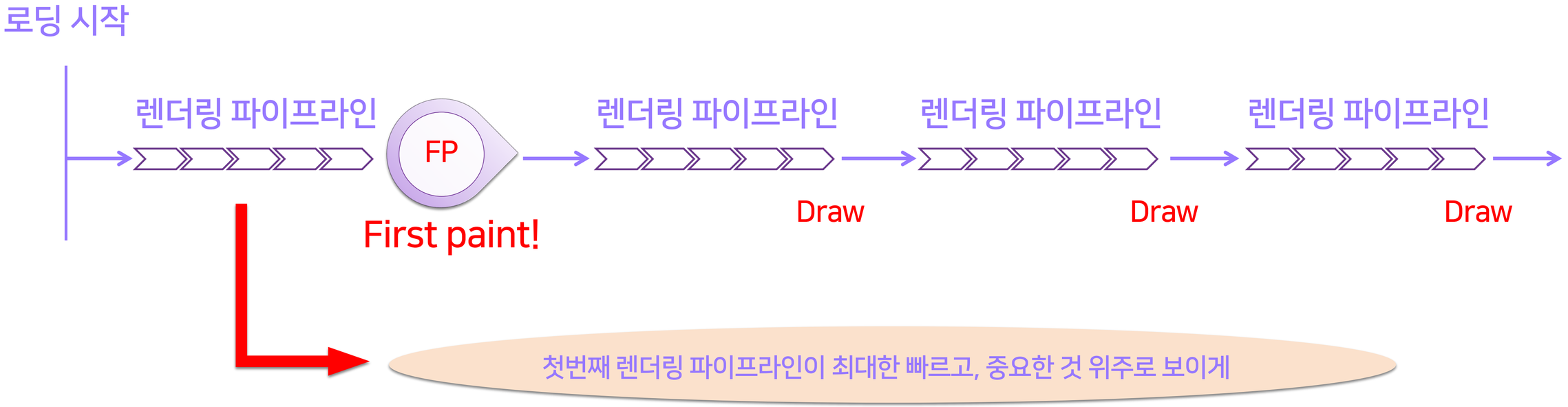
[Steve Kobes : Life of Pixel]

DOM → STYLE → LAYOUT → LAYER → PAINT → RASTER & DRAW



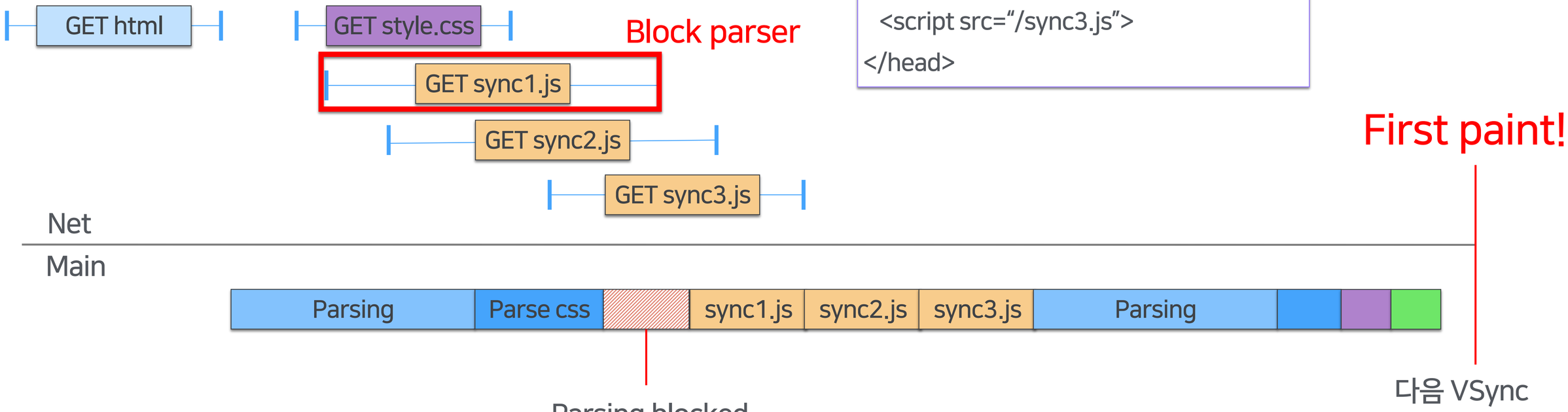
1.5 렌더링 파이프라인


DOM
STYLE
LAYOUT
LAYER
PAINT
RASTER&
DRAW


로딩 시작
렌더링 파이프라인
FP
First paint!
렌더링 파이프라인
Draw
렌더링 파이프라인
Draw
렌더링 파이프라인
Draw
첫번째 렌더링 파이프라인이 최대한 빠르고, 중요한 것 위주로 보이게

1.5 FirstPaint를 빠르게

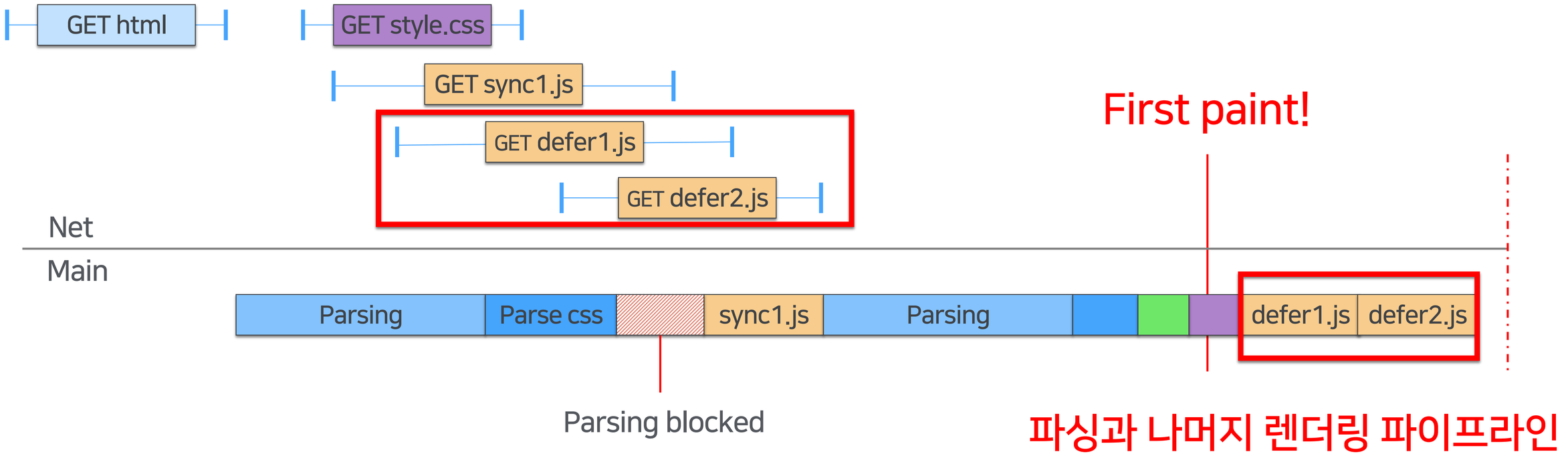

```
<head>
  <link rel="stylesheet" href="/style.css"
  <script src="/sync1.js">
  <script src="/sync2.js">
  <script src="/sync3.js">
</head>
```



<body>에 있는 script면 파싱은 멈추지만

Dom tree가 충분히 파싱됐고 css가 로드되면 렌더링은 진행 가능!

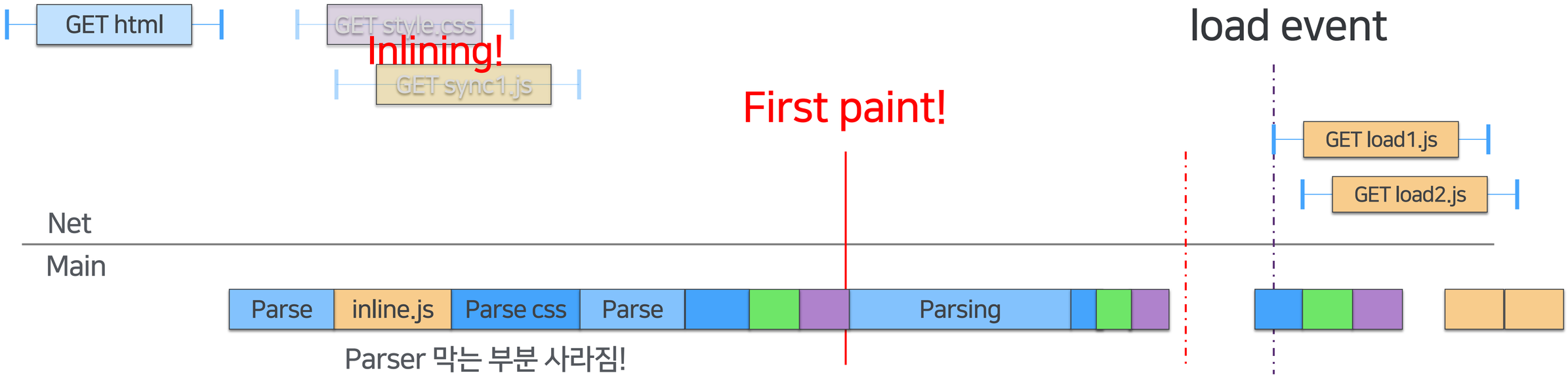
1.5 Defer script



파싱과 나머지 렌더링 파이프라인 완료 후 defer 실행

크리티컬 리소스 가정 : style.css, sync1.js

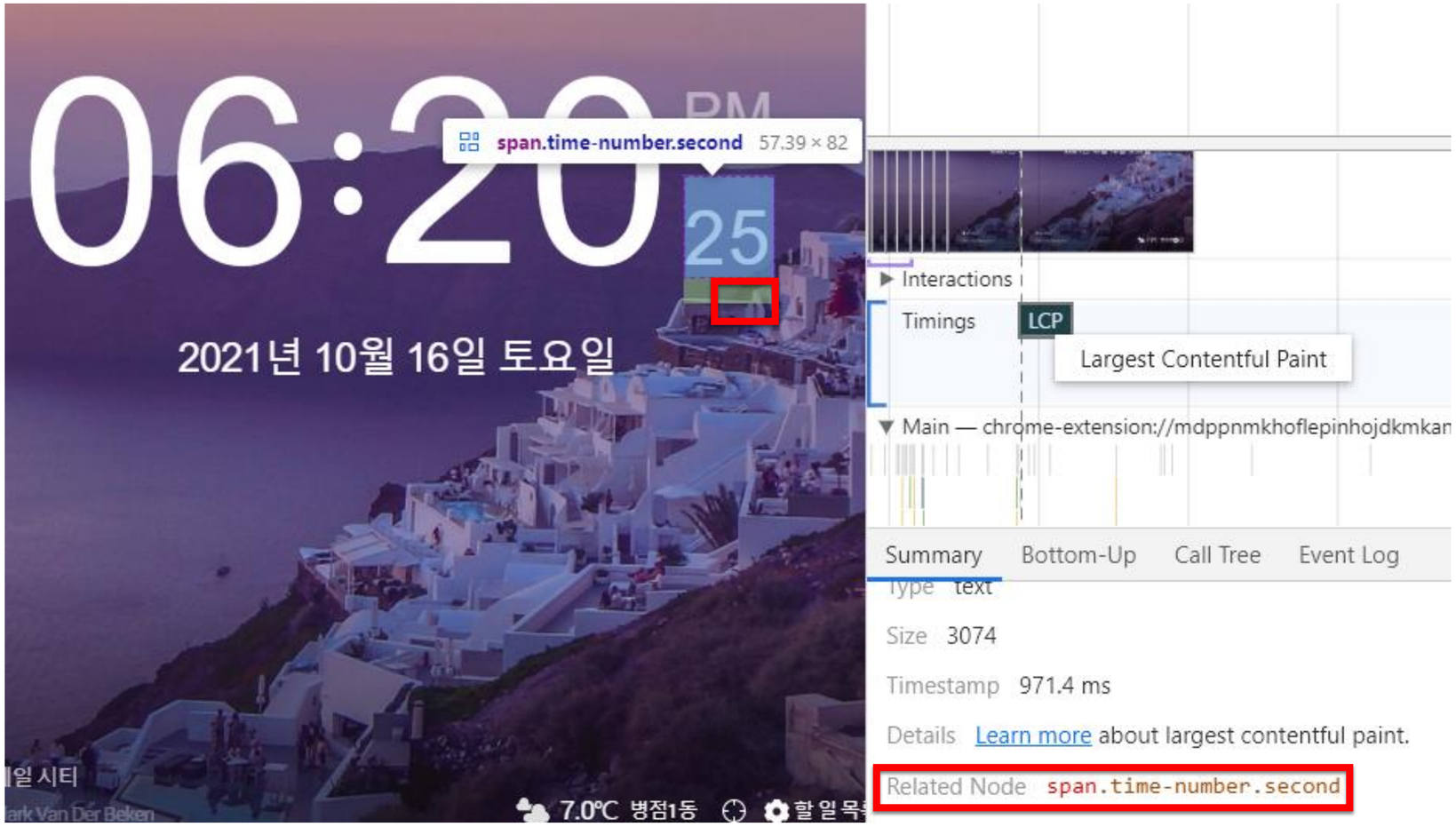
1.5 Inline & onload



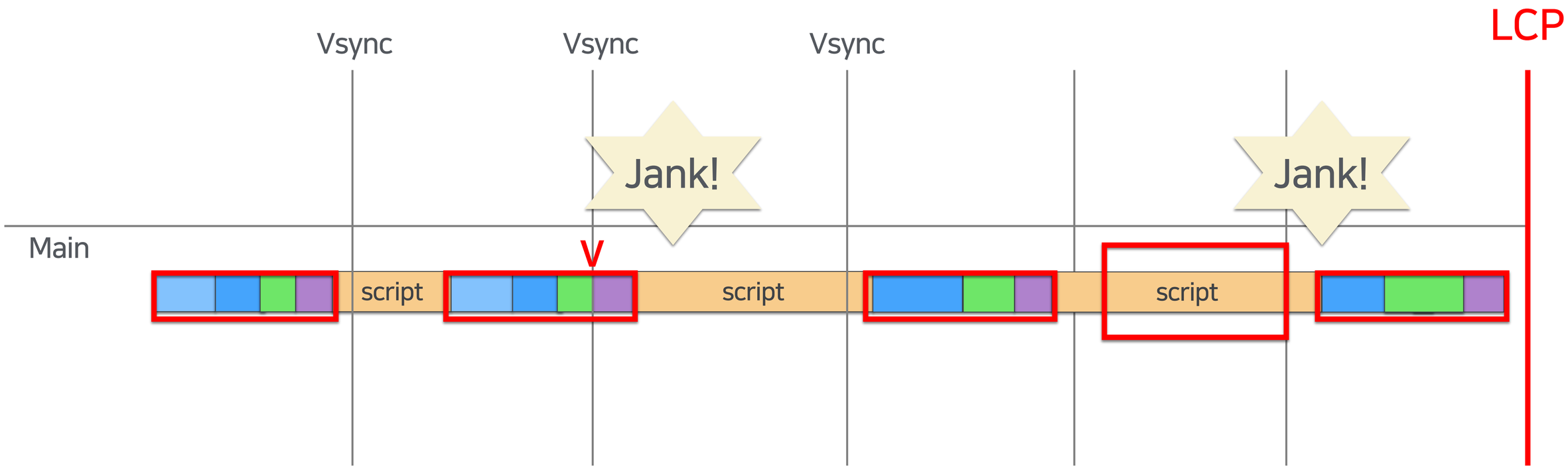
-> First Paint까지 과정은 이미 웨일 FE 개발자 분들이 탄탄하게 작업 완료

1.6 LCP (Largest Contentful paint)

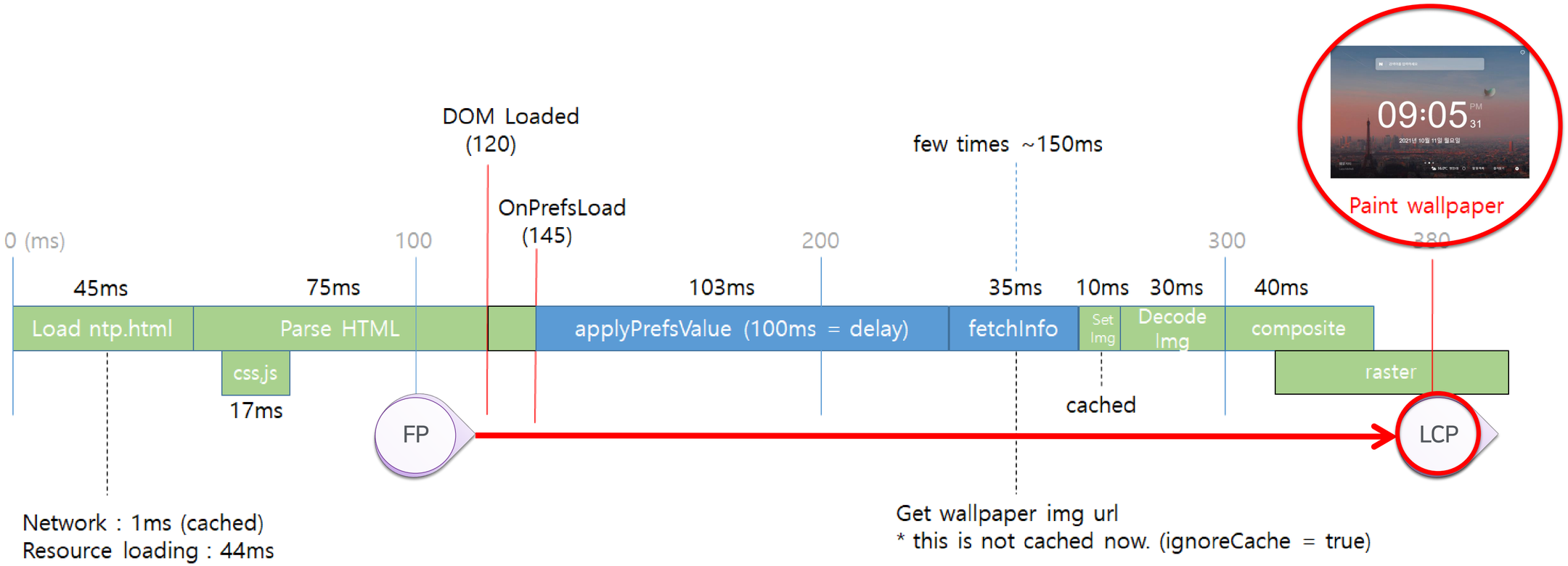
사이트마다 정확하지 않을 수 있다. 직접 Performance를 돌려야 한다 !



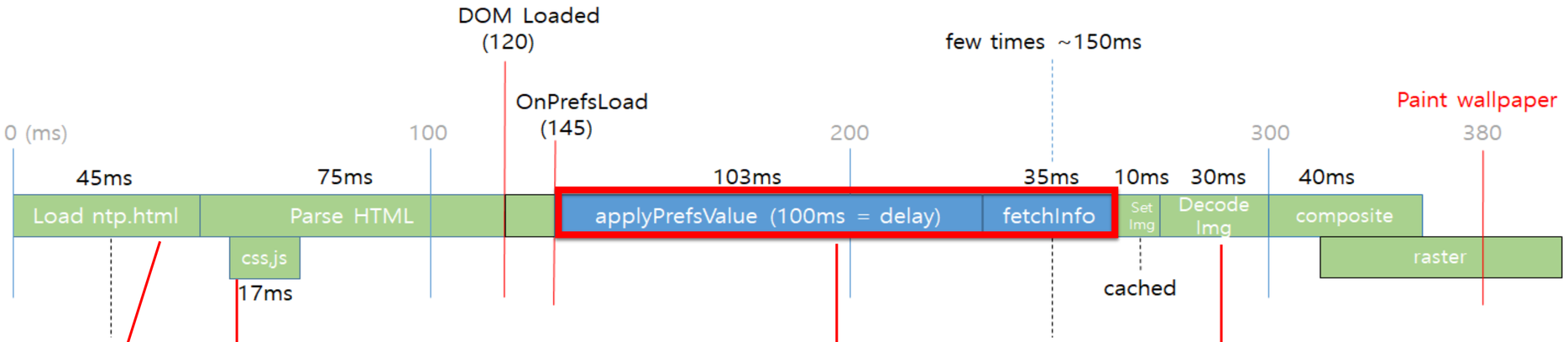
1.6 스크립트로 메인스레드는 바쁘다



1.6 웰 페이퍼 그리기(LCP)까지의 경로



1.7 정비



꼭 필요한 리소스부터 preload

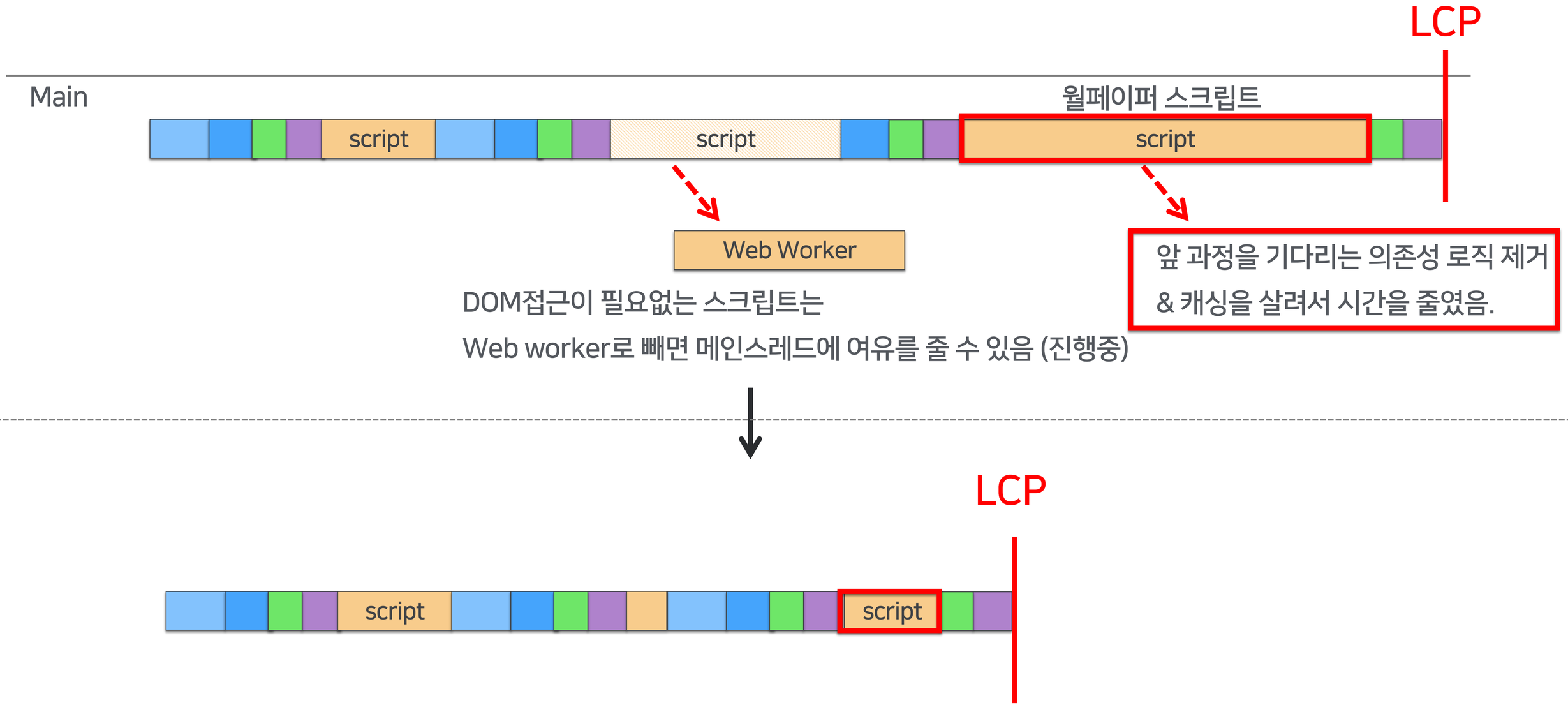
Download / decode image : WebP

자바스크립트 로직으로 월페이퍼 사용자 설정 (원하는 테마, 교체 주기 등)을 웨일 서버로부터 얻기 위해 필요

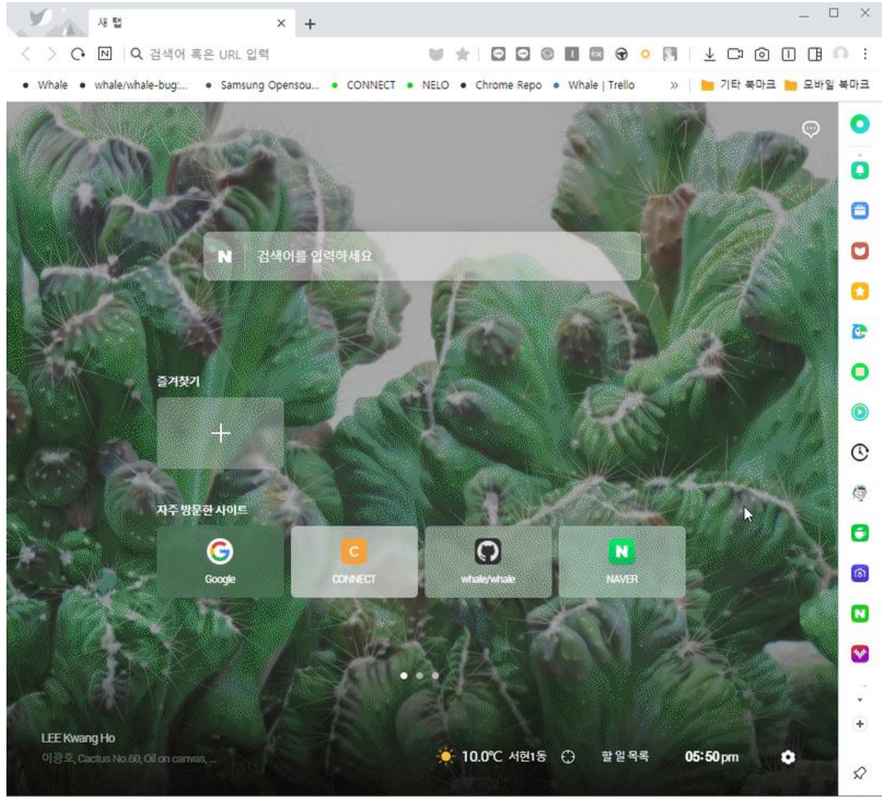
HTML 로딩 / 파싱 시간 줄이기

- 이미 필요한 압축과 서버설비는 잘 되어있음
- 페이지 분할 / Native lazy loading (그런데 새 탭은 한 페이지)
- 필요할 때 Document.write() -> Parse가 다시..
- DOM 노드 개수 줄이기

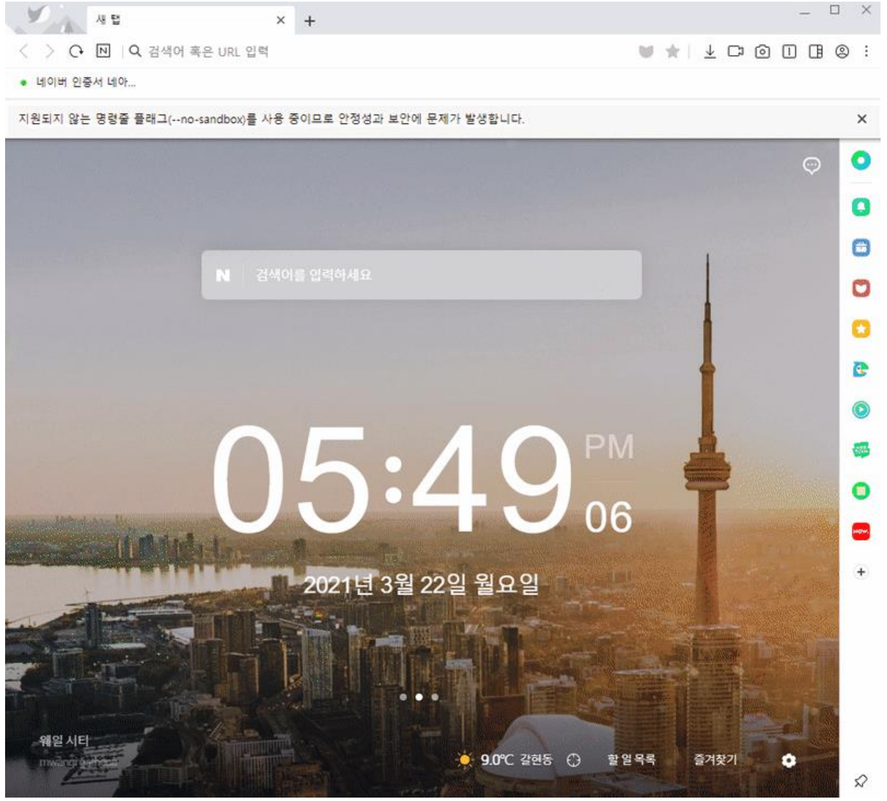
1.7 정비


LCP
Main
월페이퍼 스크립트
script
script
script
Web Worker
DOM접근이 필요없는 스크립트는
Web worker로 빼면 메인스레드에 여유를 줄 수 있음 (진행중)
앞 과정을 기다리는 의존성 로직 제거
& 캐싱을 살려서 시간을 줄였음.
LCP
script
script

1.7 뉴탭 페이지 Before & After



Before



After

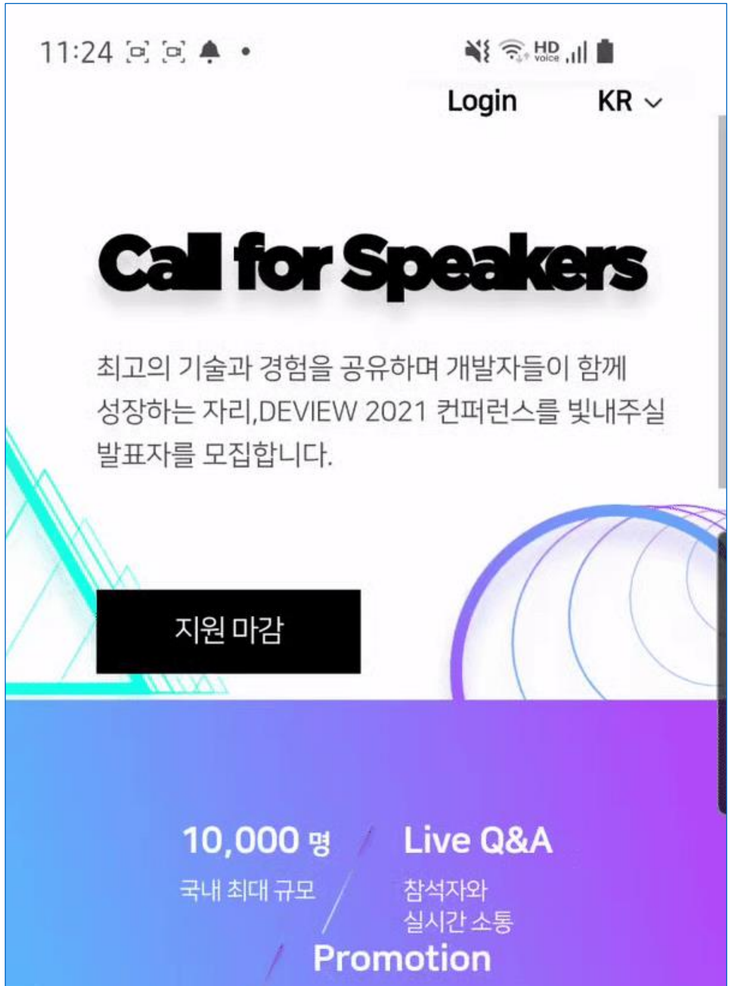
DEVIEW 2021

Vsync와 함께 춤을!
안드로이드 브라우저 스크롤 개선하기

2.1 안드로이드 스크롤 성능

DEVIEW 2021

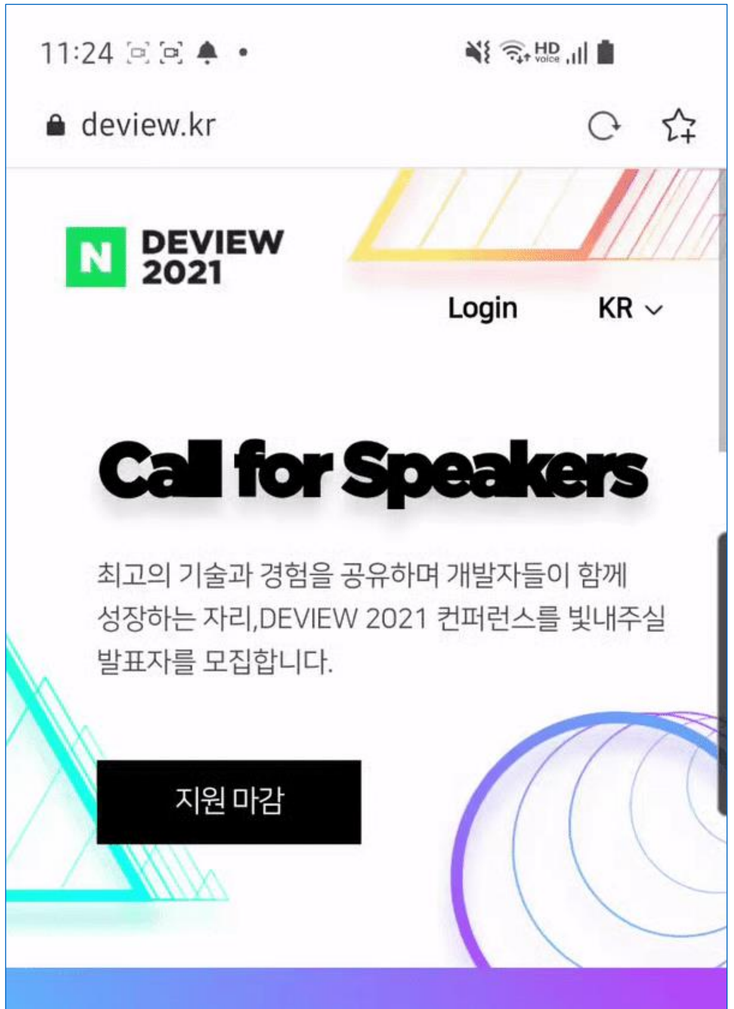
✓ scroll (panning)



두가지에서

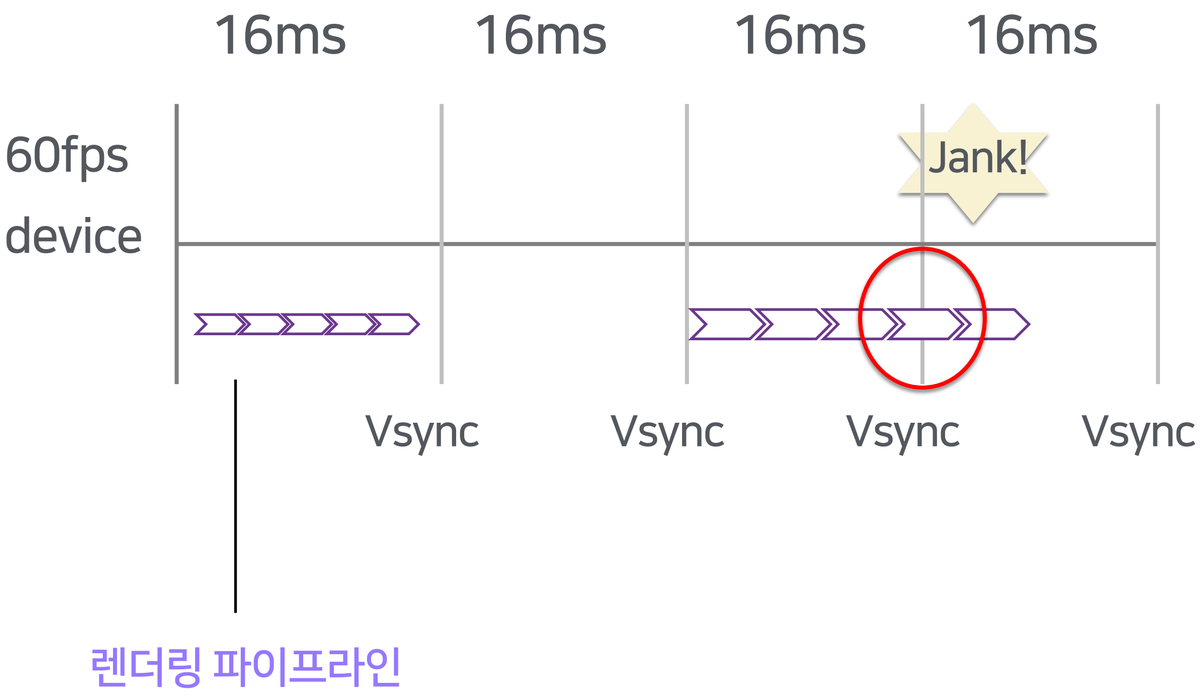
미세한 버벅거림 발생

✓ fling



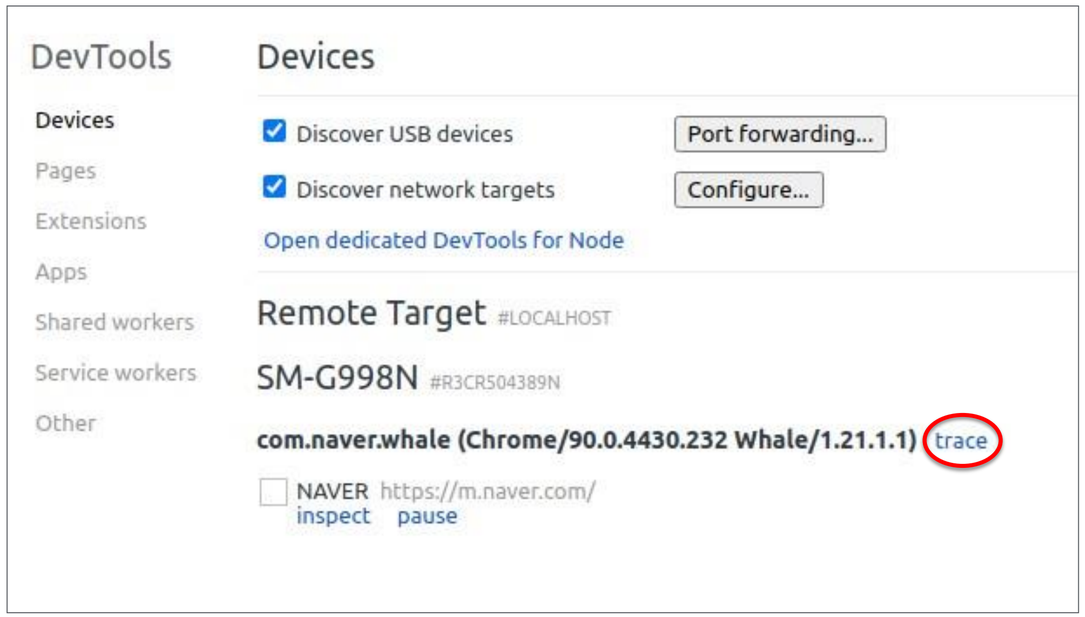
2.1 스크롤 떨림 현상 파악

✓ 안드로이드 개발자옵션 – 프로필 HWUI 렌더링 – 막대로 표시



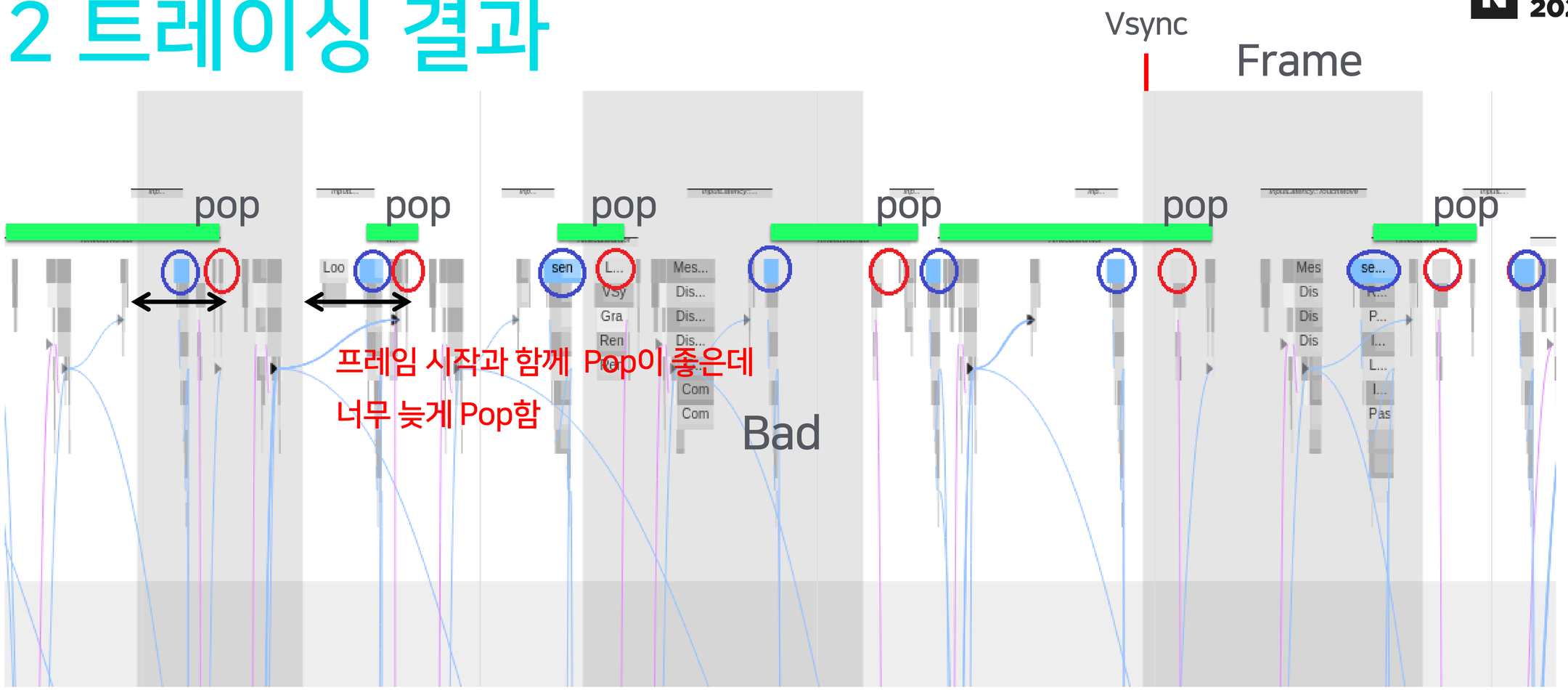
2.2 안드로이드 트레이싱

- 안드로이드 폰 (개발자 옵션, USB 디버깅 허용)
- 안드로이드 폰을 PC에 USB 연결
- 웨일, 크롬, 엣지 등 주소창에 chrome://inspect 입력



2.2 컴포지터 스레드 스크롤

2.2 트레이싱 결과

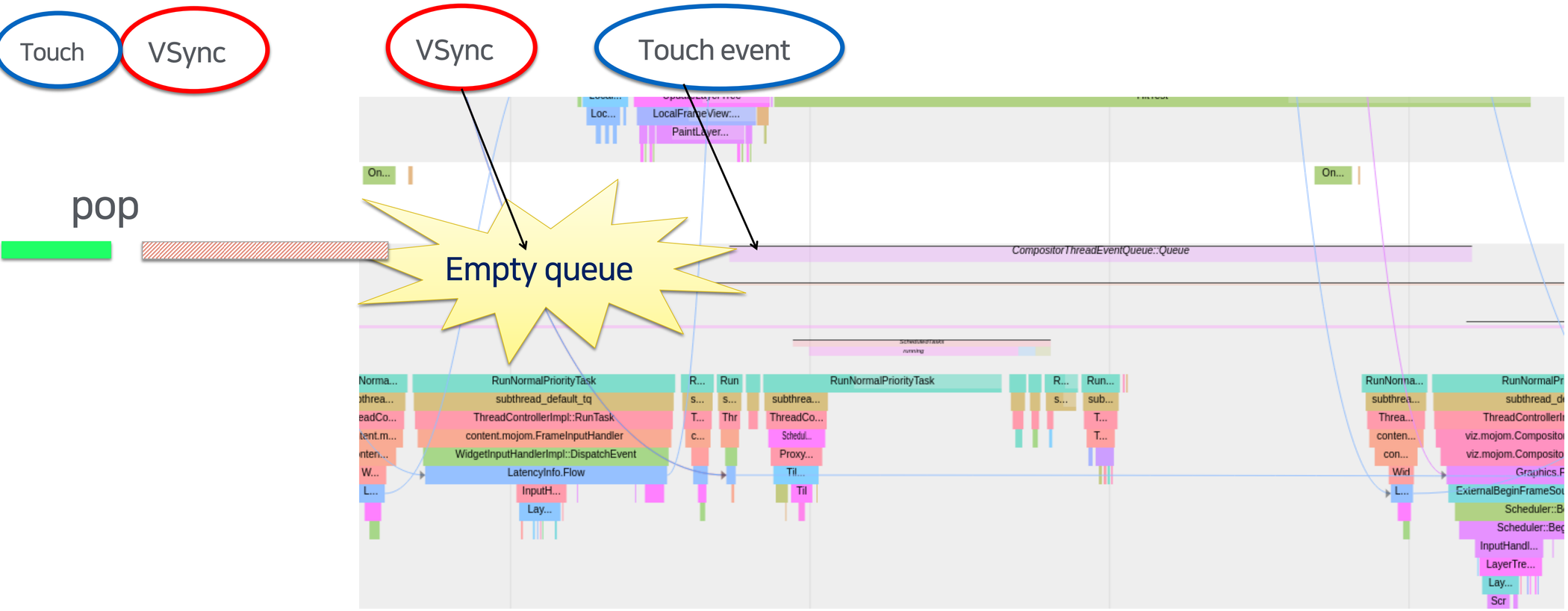


CompositorThreadEventQueue : 터치 이벤트를 모아서 합쳐, 필요할 때 forward함

push Send TouchEvent : 안드로이드 네이티브 터치 이벤트

pop OnVsync : Vsync 시그널 받은 후 큐에서 이벤트를 빼내어 처리를 시작함

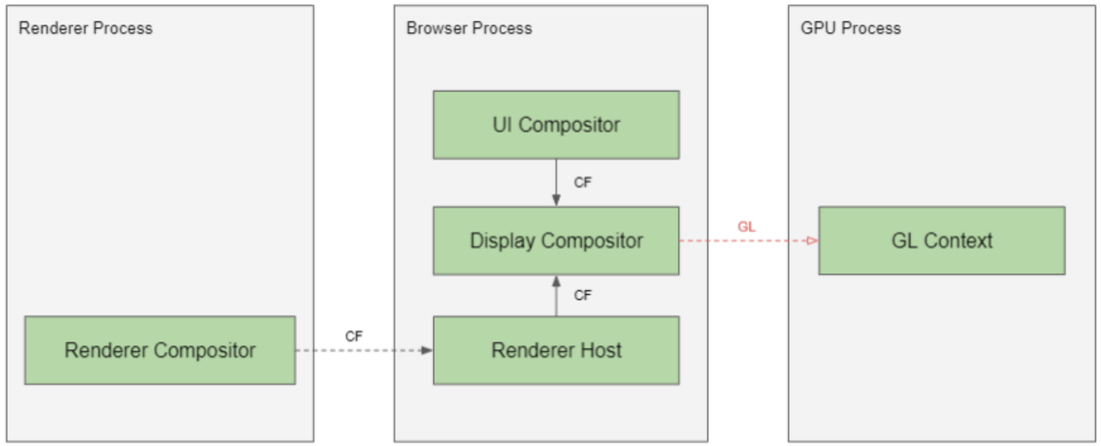
2.2 빈 큐로 인해 Jank가 발생



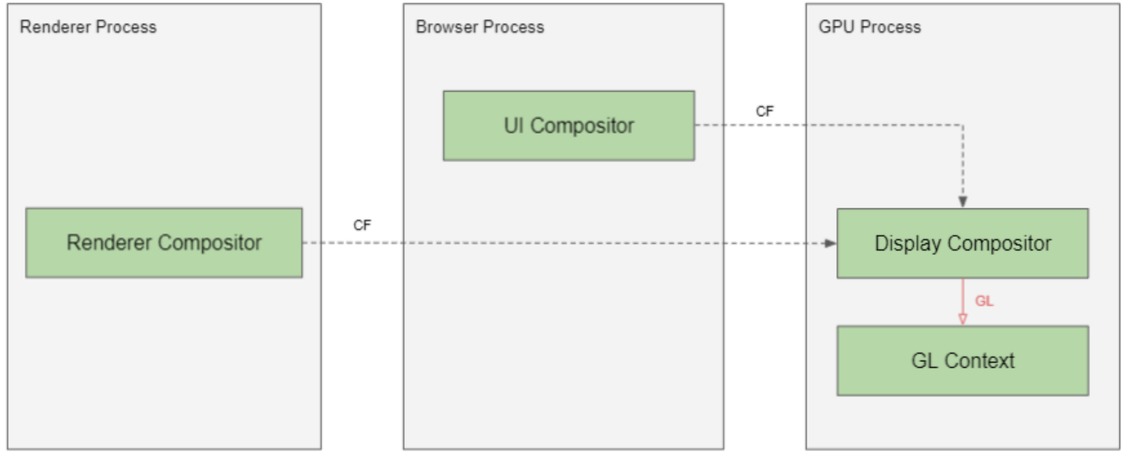
Jank로 표시되지 않고, 프레임 생성할 필요가 없다고 하는 경우도 있음

2.3 크로미움 OOPD 기능

Compositing Message Flow (Before)



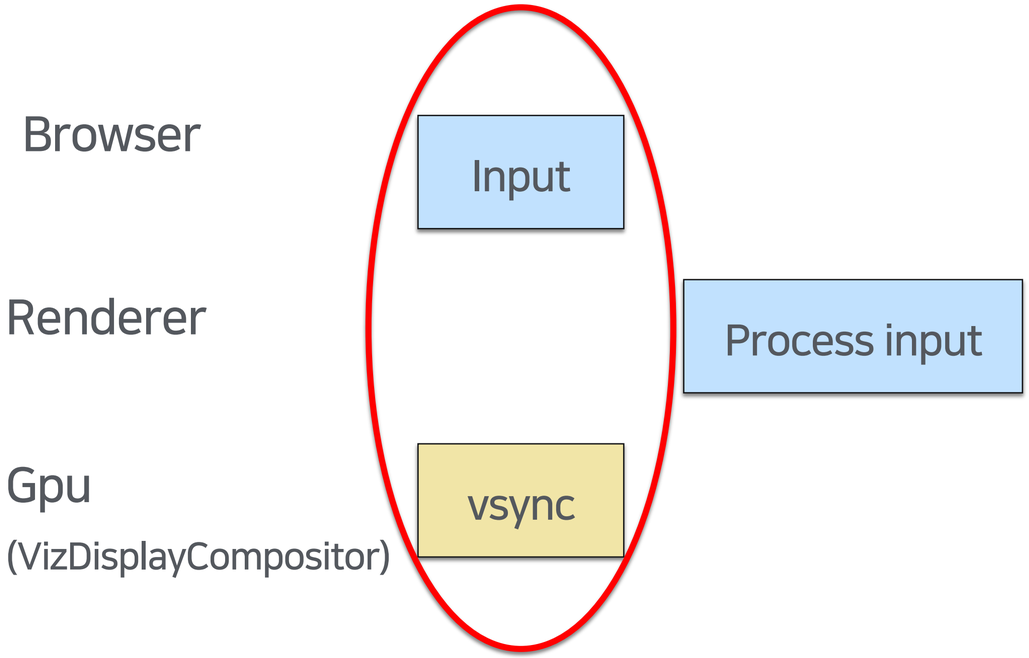
Compositing Message Flow (OOP-D)



[OOP-D Brownbag (public)]

2.3 크로미움 OOPD 적용 초기

문제 : Viz thread에서 Vsync 신호를 받게 되면서
입력과 Vsync의 순서를 나란히 보장할 수 없었고,
컴포지터스레드에서 Vsync신호를 늦게 받을 때가 있었음.

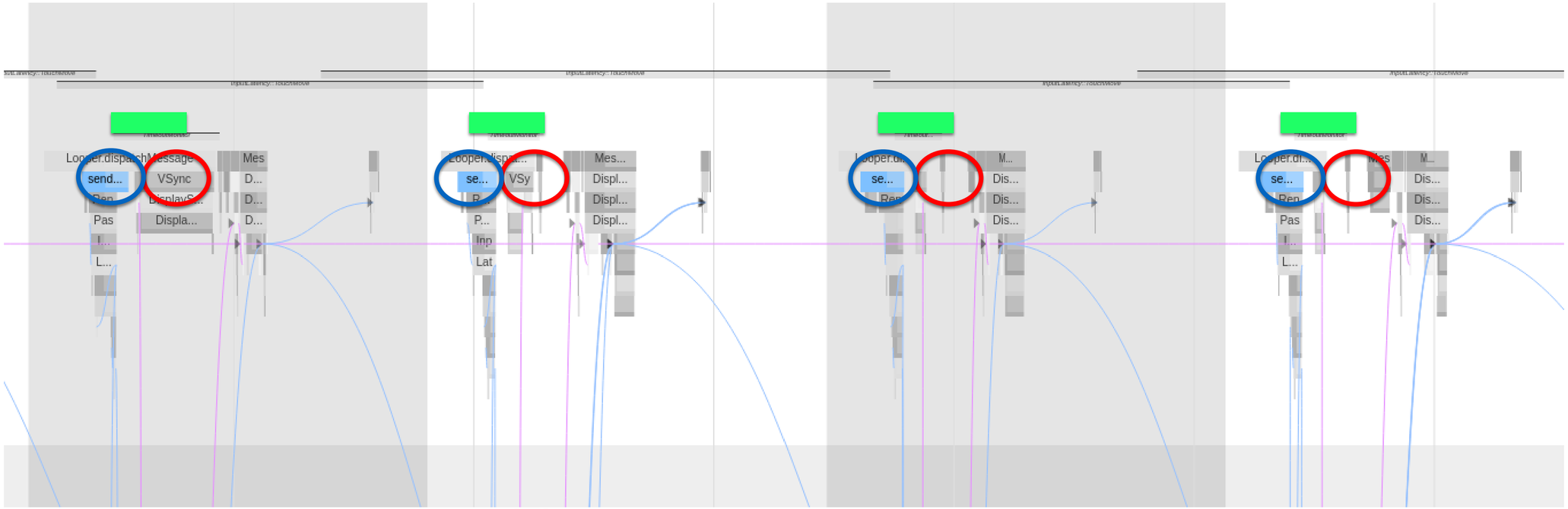


해결했던 방법

1. 안드로이드 네이티브 입력 버퍼링 옵션 (입력 강제 스케쥴)
2. 브라우저에서 Vsync 받도록 원복
3. 나중에라도 이벤트가 들어오면 Forward

(1,2번을 하면 충분해서 3번은 나중에 뺌)

2.4 OOPD 부작용 해결



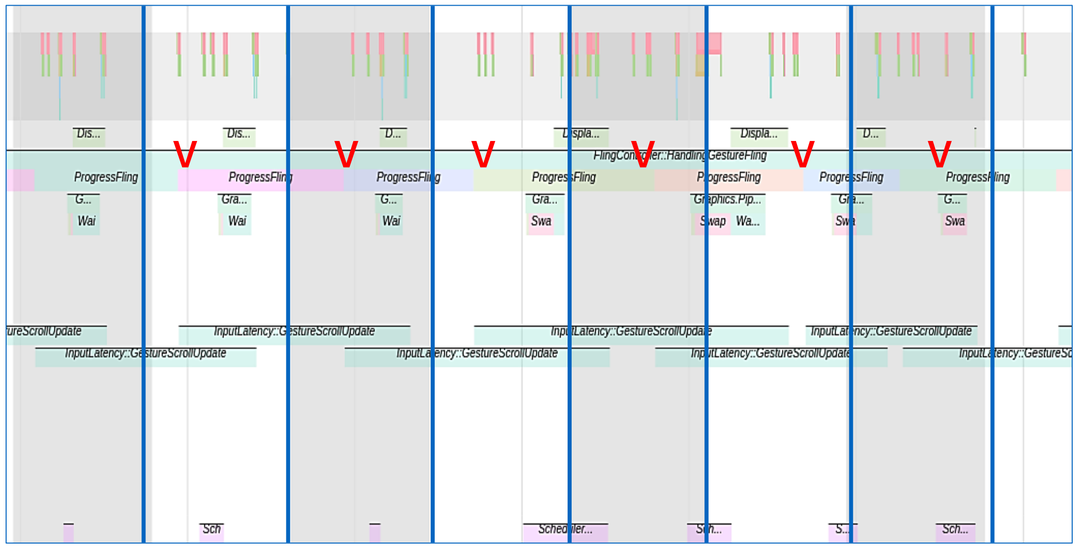
항상 Send TouchEvent 다음 , Vsync 가 오게하고

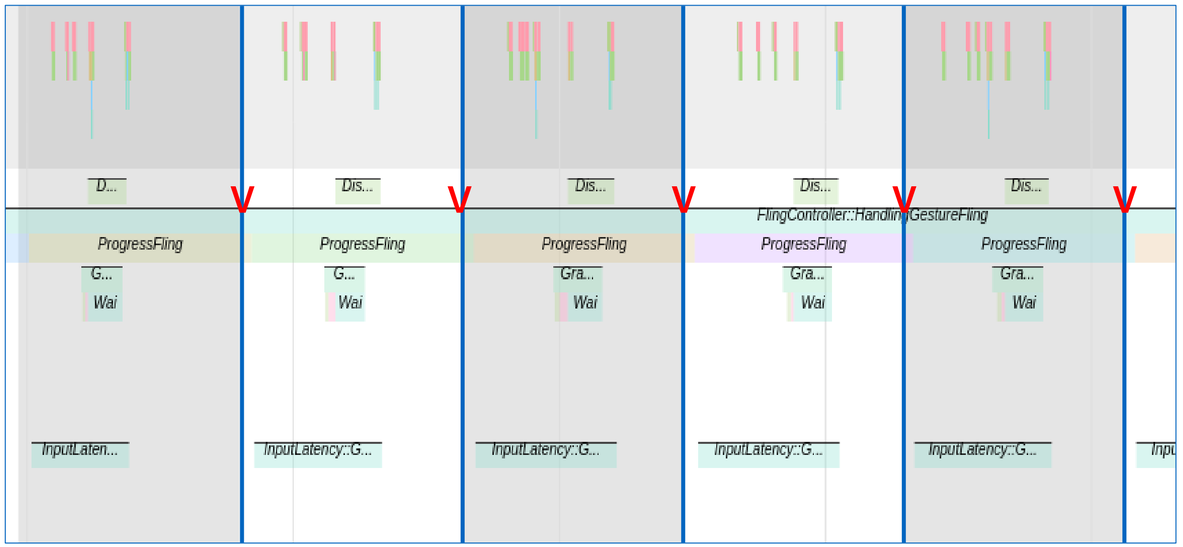
프레임 시작 초기에 큐에서 이벤트를 빼내어, Jank 현상을 방지함.

참고 : 꽤 시일이 지나고 현재 이 OOPD 부작용은 오픈소스에서 고쳐짐.

2.5 안드로이드 플링 성능

: Vsync tick　　V : Progress Fling start





늦게 시작하는 Progress Fling

Fling 처리 로직을 Vsync 바로 다음으로 배치

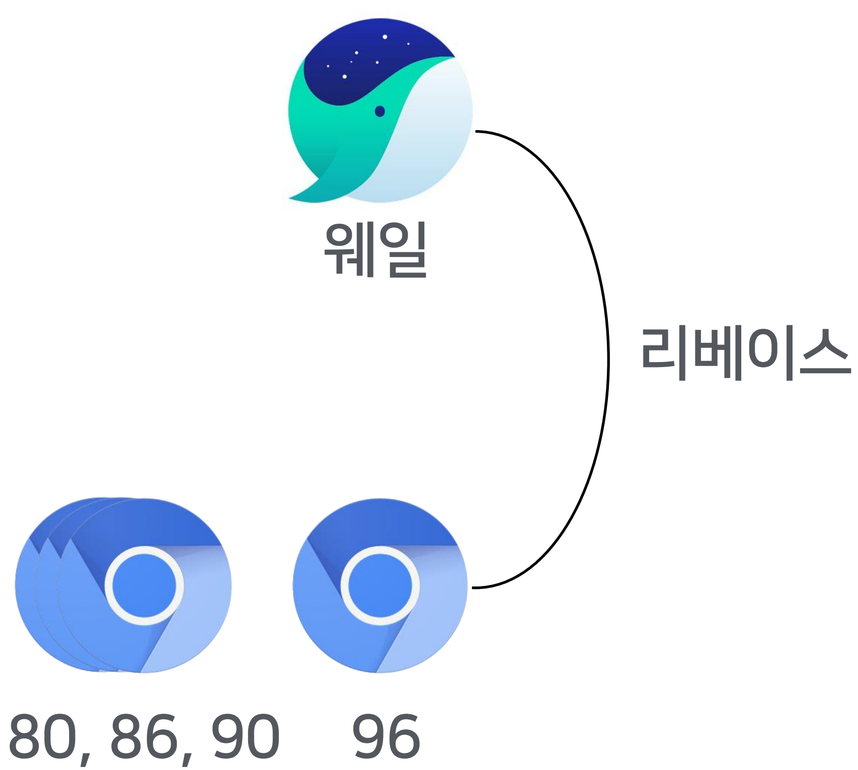
Vsync와 거의 동시에 시작

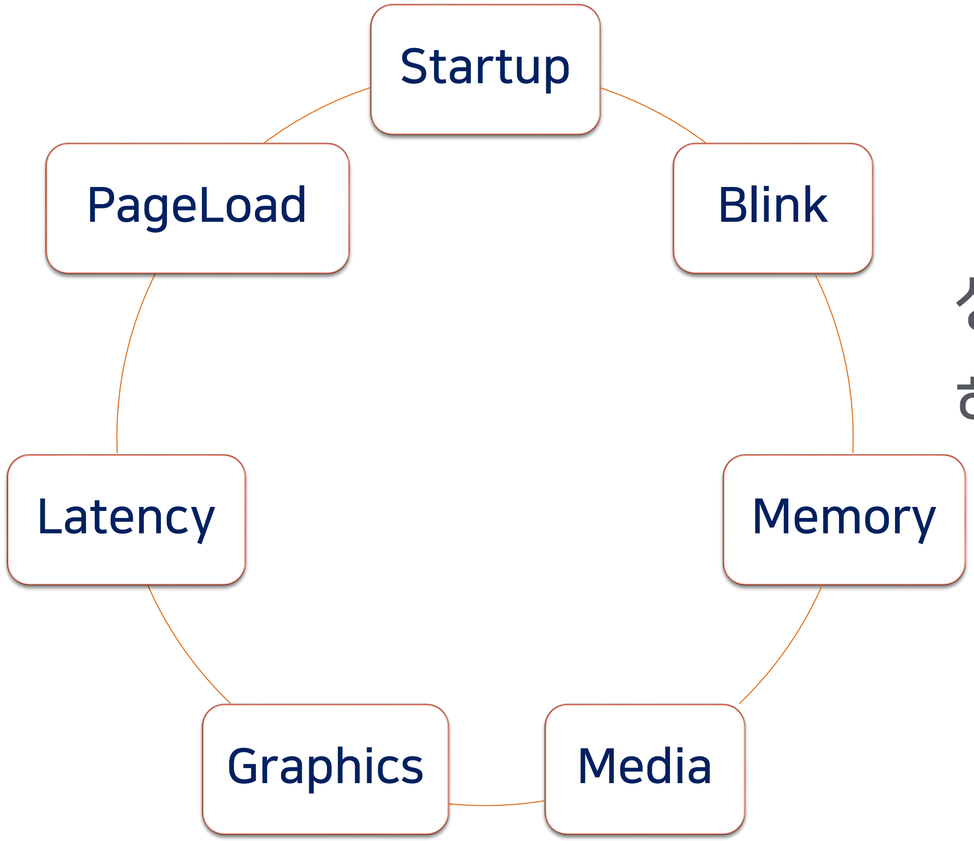
DEVIEW 2021

Histograms / Tracing으로 브라우저 성능 파악하기 - 웨일 실행 성능

3.1 Histograms

✓ 브라우저 엔진에서 기록하는 다양한 데이터들

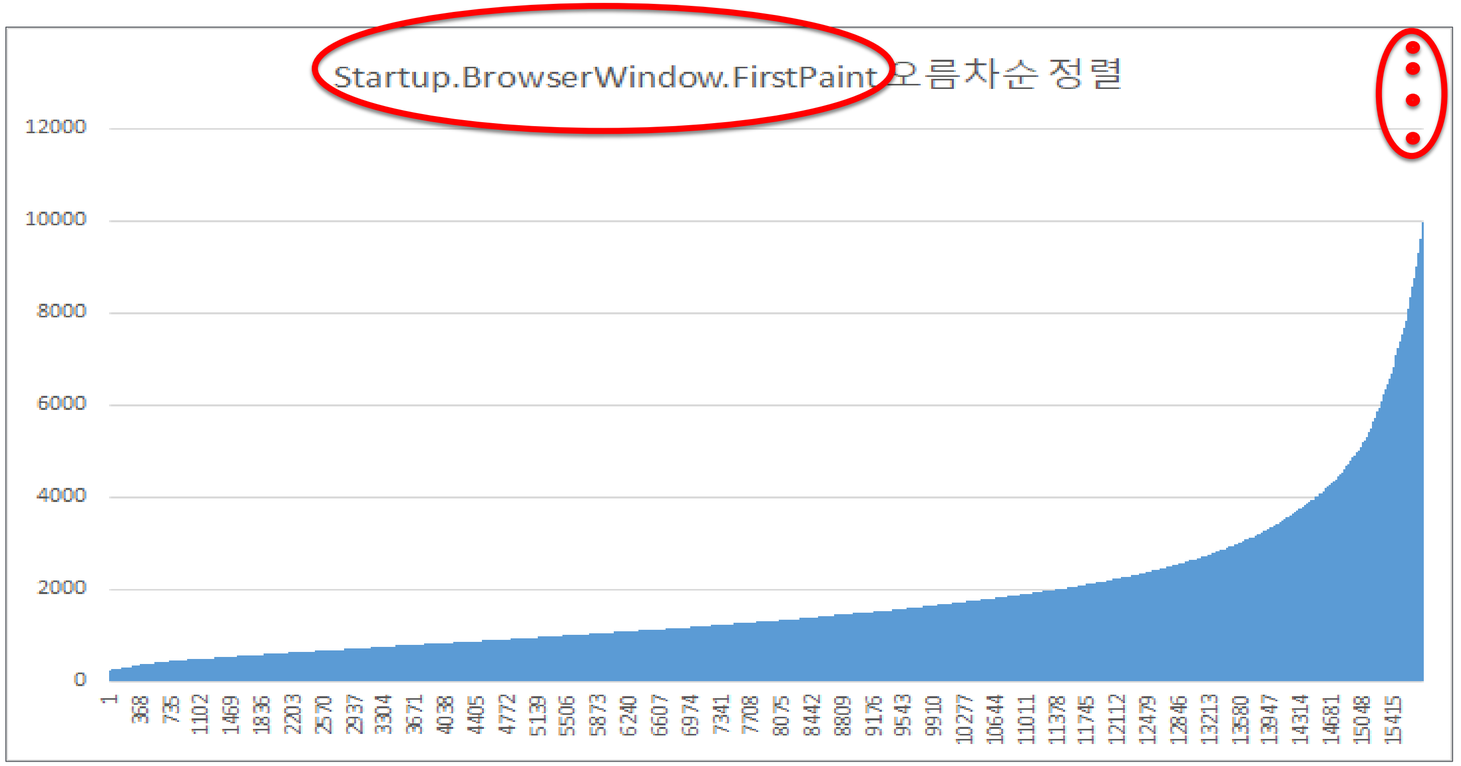




성능 개선을 위한
히스토그램 수집

3.1 Histograms

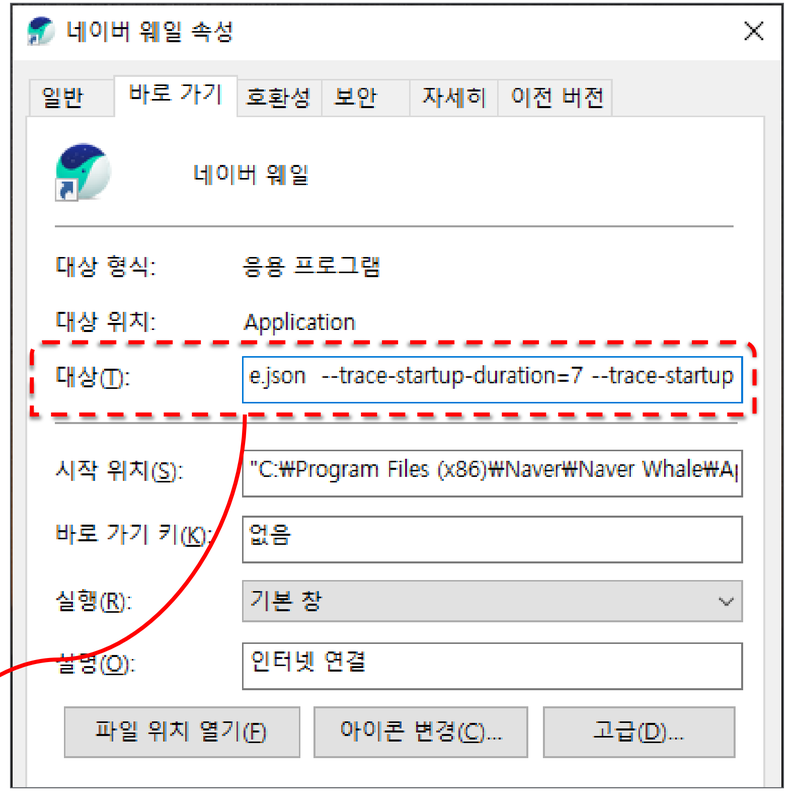
✓ 눈에 띄게 안 좋은 실행 성능 발견



- 실행 성능 최하위 사용자(0.5% 미만)의 수치가 눈에 띄게 좋지 않았음

- 사내에서도 같은 문제가 발생하는 PC가 있음 -> 크롬도 동일하게 느렸고, 엣지만 정상

3.2 실행 시점 프로파일링

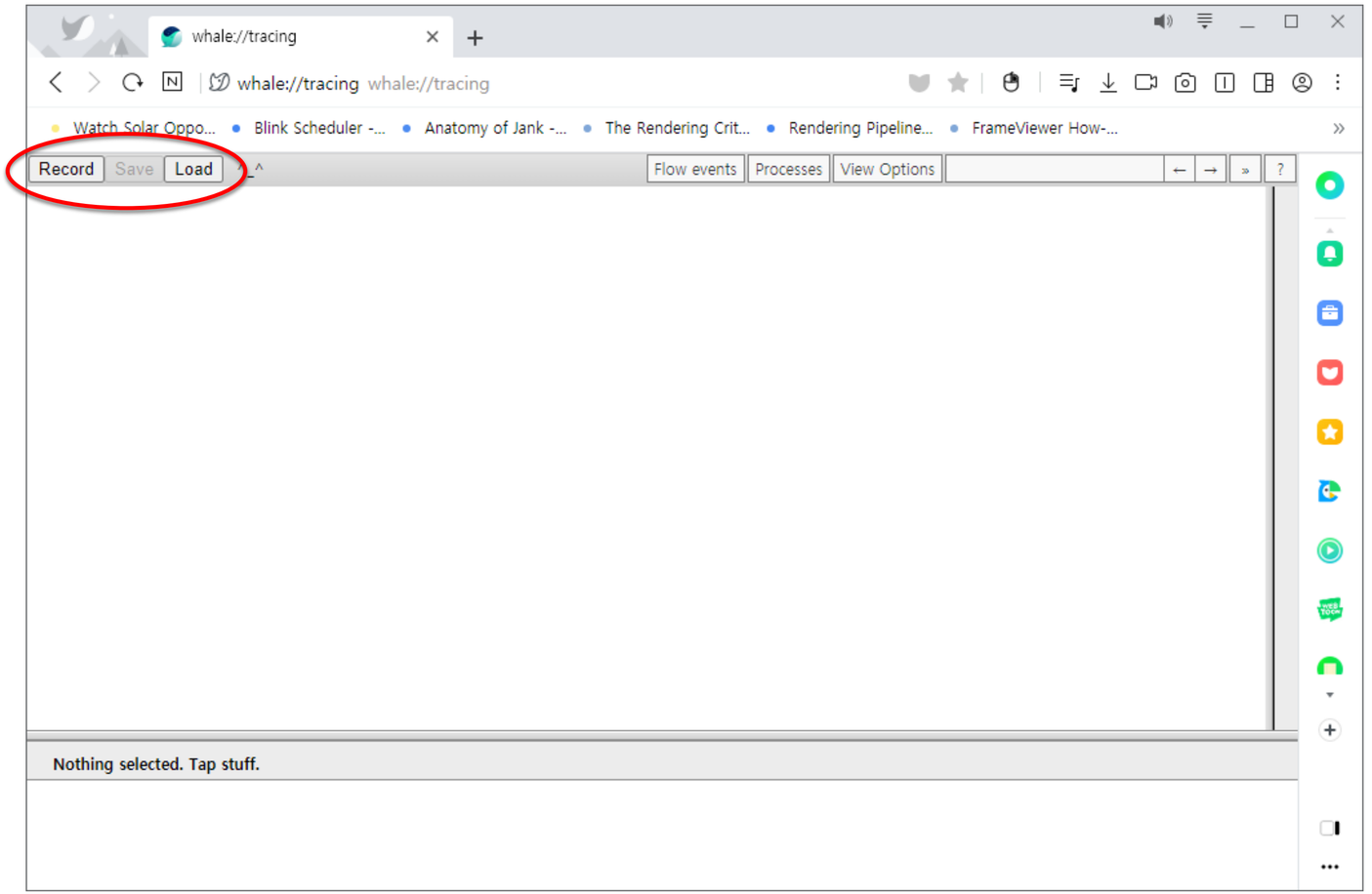


--trace-startup-file=C:₩Users₩tmp₩trace.json --trace-startup-duration=7 --trace-startup

3.2 브라우저 실행 프로파일링 ①

DEVIEW 2021

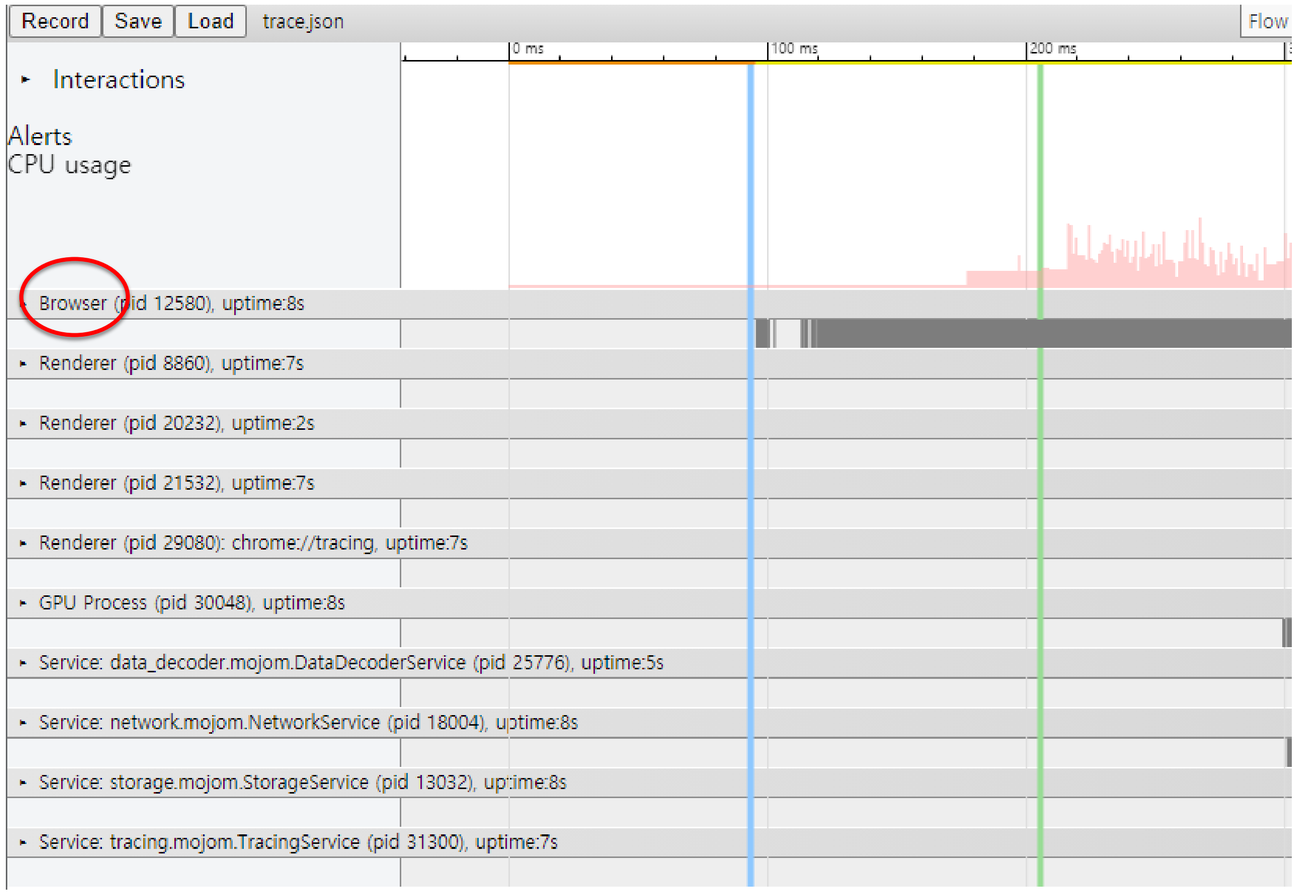
- Tracing



chrome://tracing

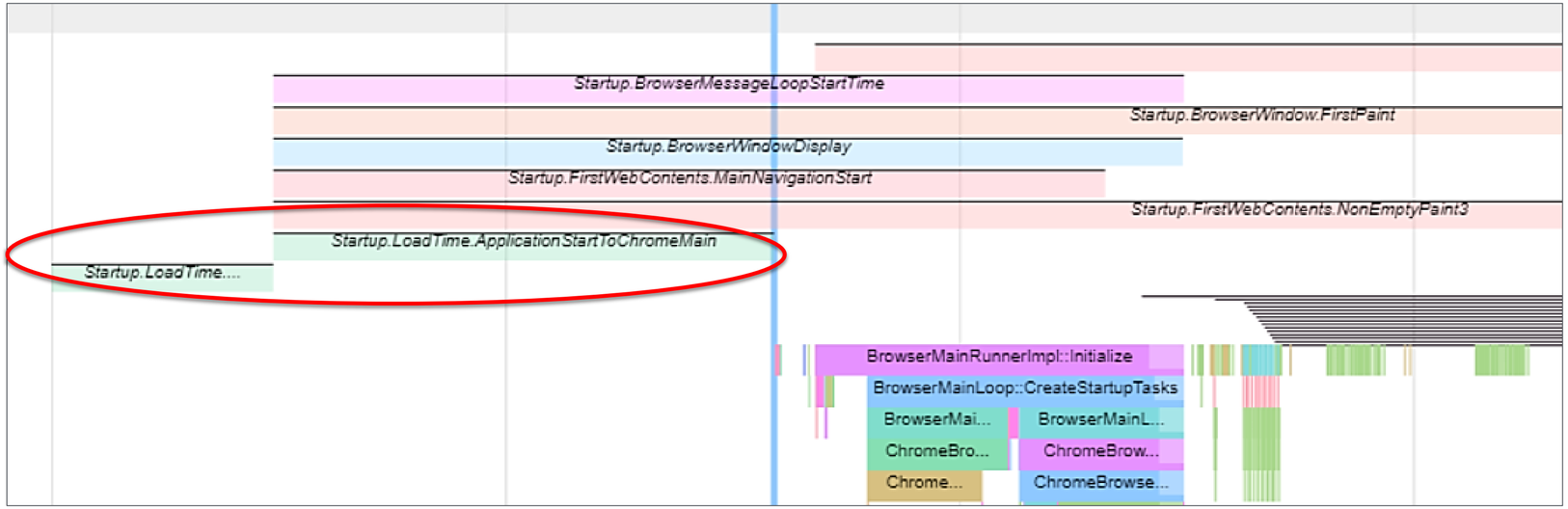
3.2 Tracing 구성

- Browser
- Renderers
- GPU Process
- Services



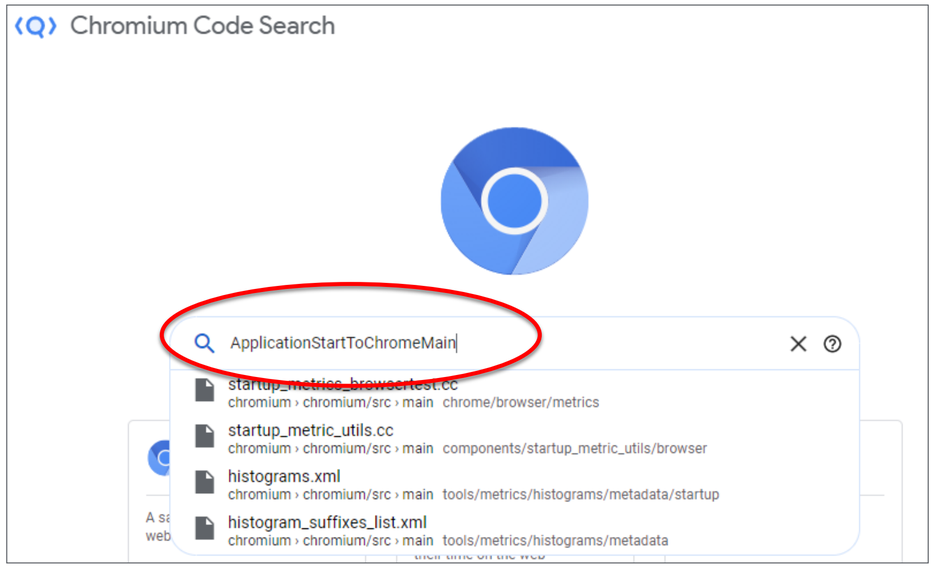
3.3 실행 과정

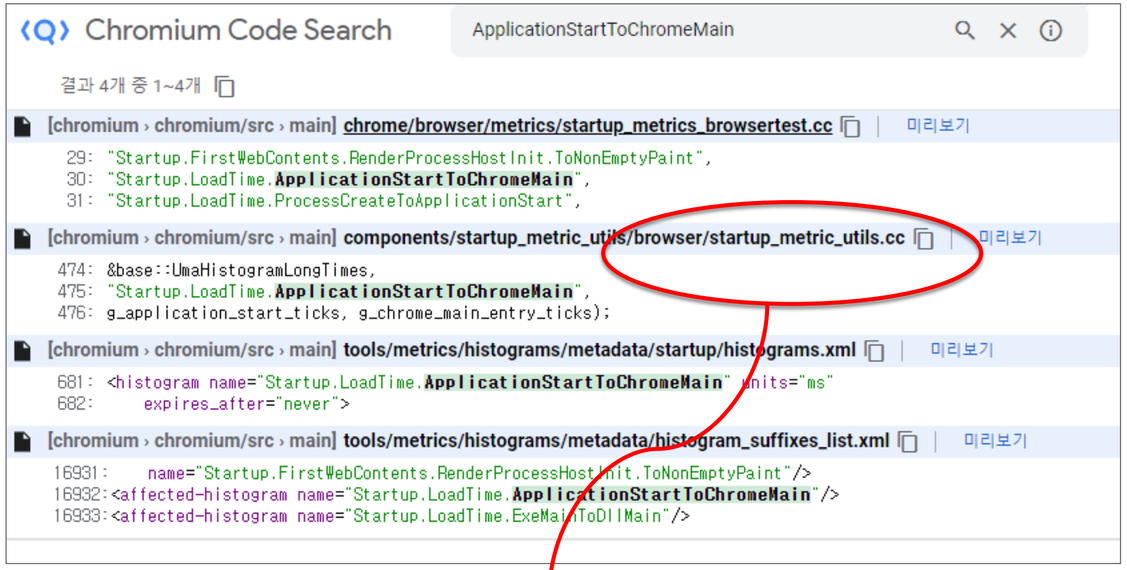
- Startup.LoadTime.ProcessCreateToApplicationStart
- Startup.LoadTime.ApplicationStartToChromeMain
- BrowserMainRunnerImpl::Initialize
- Startup.BrowserWindowDisplay : 윈도우가 뜨는 시점
- WebContentsImpl Loading : 시작페이지, 익스텐션 로딩 등
- Startup.BrowserWindow.FirstPaint : 글로벌 첫 페인트



3.3 소스 보기

- 크로미움 소스 다운로드? -> 일반 성능 데스크탑 빌드 4시간 이상 소요
- https://cs.chromium.org -> 크로미움 코드 서치 (간단한 커밋도 가능)






```
468         "Startup.LoadTime.ProcessCreateToApplicationStart",
469         g_process_creation_ticks, g_application_start_ticks);
470
471     // Application start to ChromeMain().
472     DCHECK(!g_chrome_main_entry_ticks.is_null());
473     UmaHistogramWithTraceAndTemperature(
474         &base::UmaHistogramLongTimes,
475         "Startup.LoadTime.ApplicationStartToChromeMain",
476         g_application_start_ticks, g_chrome_main_entry_ticks);
477   }
478 }
479
```

3.3 소스 보기

```
46
47 base::TimeTicks g_application_start_ticks;
48
49 base::TimeTicks g_chrome_main_entry_ticks;
50
51 base::TimeTicks g_message_loop_start_ticks;
52
```

기록 참조

g_chrome_main_entry_ticks

입력하여 파일 경로로 필터링하세요.

정의(1개 표시됨)

components/startup_metric_utils/browser/startup_metric_utils.cc (어커런스 1개)

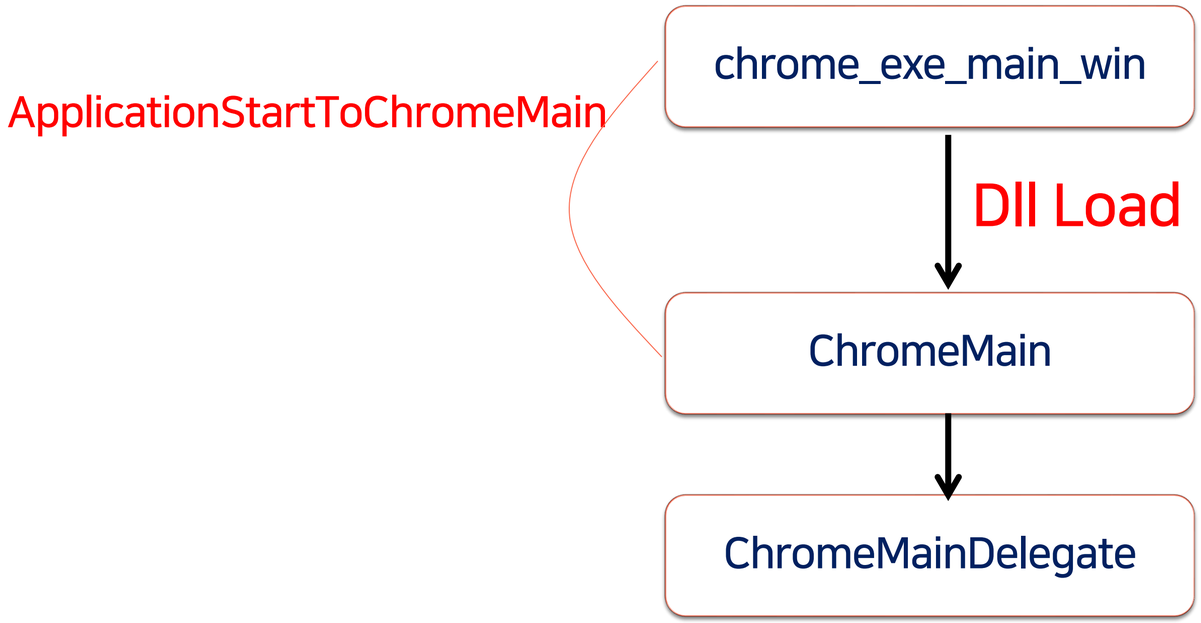
```
49: base::TimeTicks g_chrome_main_entry_ticks;
```

참조(8개 표시됨)

components/startup_metric_utils/browser/startup_metric_utils.cc (어커런스 8개)

```
377: if (g_chrome_main_entry_ticks.is_null())
381: "startup", "Startup.BrowserMainEntryPoint", 0, g_chrome_main_entry_ticks);
419: DCHECK(g_chrome_main_entry_ticks.is_null());
420: g_chrome_main_entry_ticks = ticks;
421: DCHECK(!g_chrome_main_entry_ticks.is_null());
472: DCHECK(!g_chrome_main_entry_ticks.is_null());
476: g_application_start_ticks, g_chrome_main_entry_ticks);
594: return g_chrome_main_entry_ticks;
```

Exe entry point



ApplicationStartToChromeMain = DLL loading time

3.4 원인 파악하기

-> 크로미움 기반 Edge와 웨일, 크롬은 DLL 로딩 부분에서 소스가 일치할 것이다

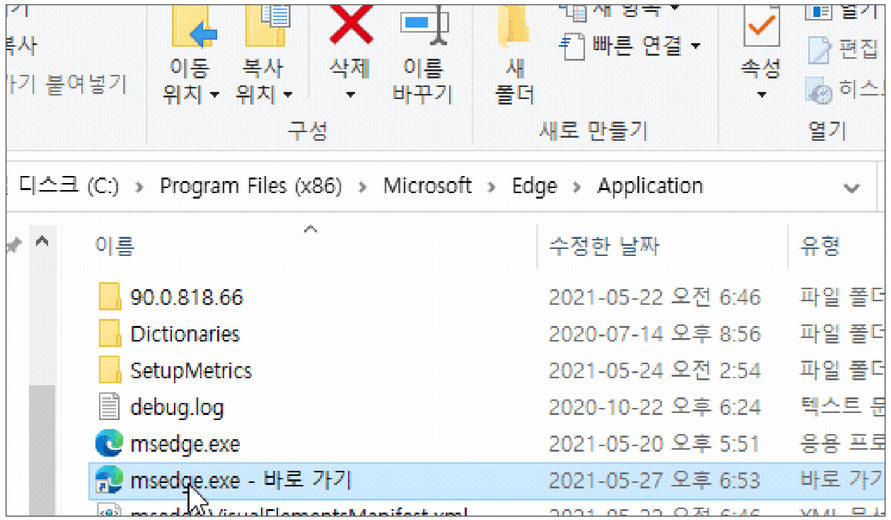
-> 왜 웨일과 크롬이 느렸을까? 엣지만 빠르고.. 그렇다면

-> 다른 것은?? 정말 단순하게.. 이름? (웹 상에선 UserAgent 이슈가 꽤 있다)

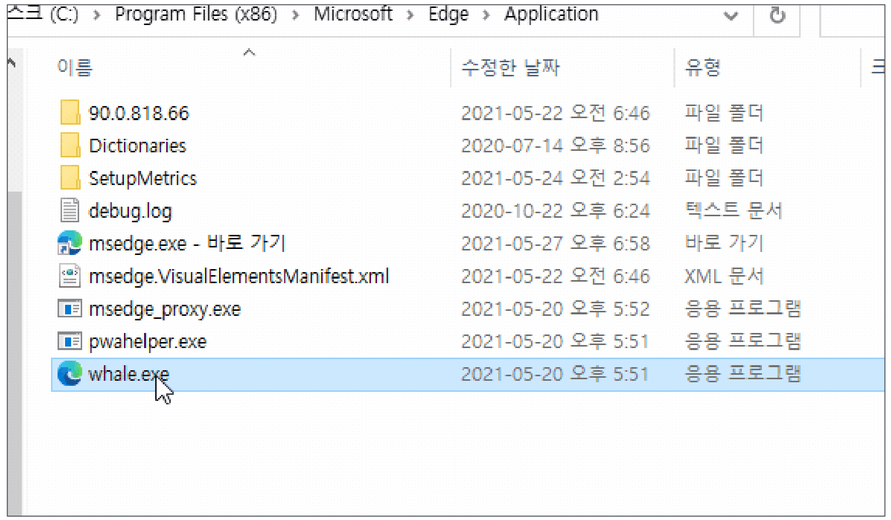
3.4 이름 변경으로 문제 조건 확인

① 웨일/크롬 대비 실행시간이 빨랐던 Edge

② Edge를 크롬이나 웨일로 이름을 바꾸면 같이 느려진다



Edge 브라우저
(실행시간 : 1초)



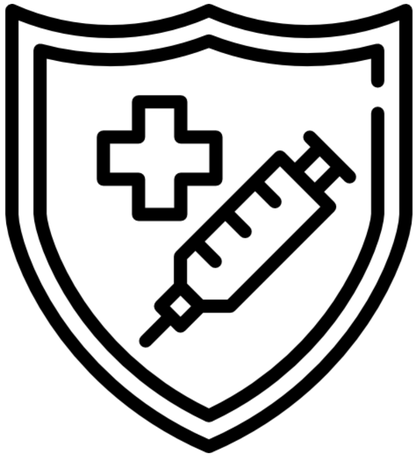
웨일(크롬)로
이름을 바꾼 Edge
(실행시간 : 4초)

3.4 문제 조건으로 의심하기

-> 이름이 문제 발생의 필요조건이다.

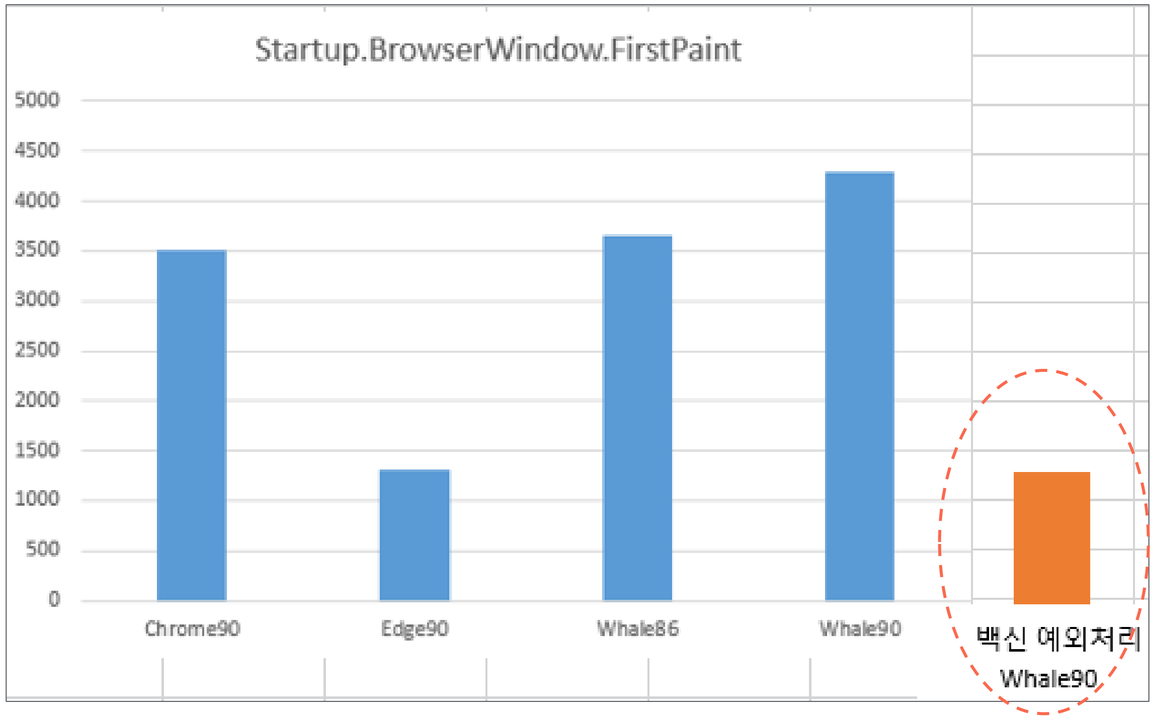
- ✓ 무언가 브라우저 이름에 따라 실행에 영향을 끼친다
  (화이트 리스트, 블랙 리스트?)

악성 프로그램도 있지만..
-> 백신을 의심



3.5 백신문제 해결

- 예외처리 적용
- 사이닝 적용
- 평판 탐지 대기



모 백신이 설치된 PC에서의 성능 비교

Q & A

Thank You