DEVIEW
2019

안드로이드 앱의 다중 웹뷰 환경에서 성능 병목 진단 및 최적화 사례

이성원
Whale
NAVER

CONTENTS

DEVIEW
2019

1. 웹브라우저 엔진을 탑재한 앱의 웹뷰 사용
2. 다중 웹뷰를 활용한 사용자 반응성 개선
3. 앱 시동 시 성능 분석
4. 성능 개선 방안 개발 및 결과

DEVIEW
2019

1.웹브라우저 엔진을 탑재한 앱의 웹뷰 사용

1.1 안드로이드 앱 안의 웹뷰

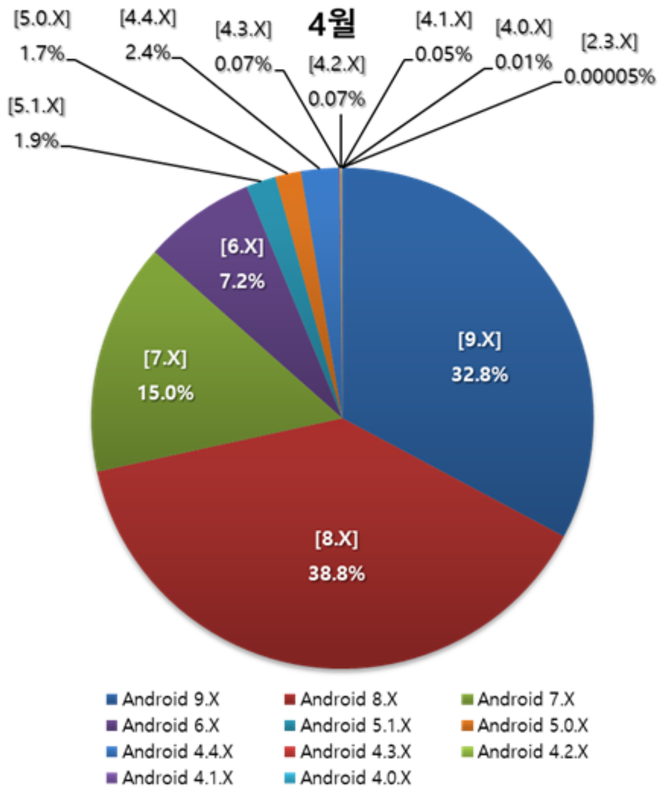
DEVIEW 2019

앱 안에서 웹페이지를 rendering하여 출력

1.2 웹브라우저 엔진 Vs. 시스템 웹뷰

엔진 유지 보수 비용 Vs. API 파편화 이슈 대응

엔진 내장한 앱 크기 증가 Vs. 버전에 따른 성능 및 기능의 제약



	WebView v30	WebView v33	WebView v36
WebGL	x	x	✓
WebRTC	x	x	✓
WebAudio	x	x	✓
Fullscreen API	x	x	x
Form validation	x	✓	✓
Filesystem API	x	x	x
File input type	x	x	x
<datalist>	x	✓	✓

1.3 XWhale

Intel의 Crosswalk 프로젝트를 계승

안드로이드 앱에 브라우저 엔진으로 탑재

시스템 웹뷰를 대체하고 추가 기능을 제공하는 API 제공

네이버앱과 네이버 카페 등 적용

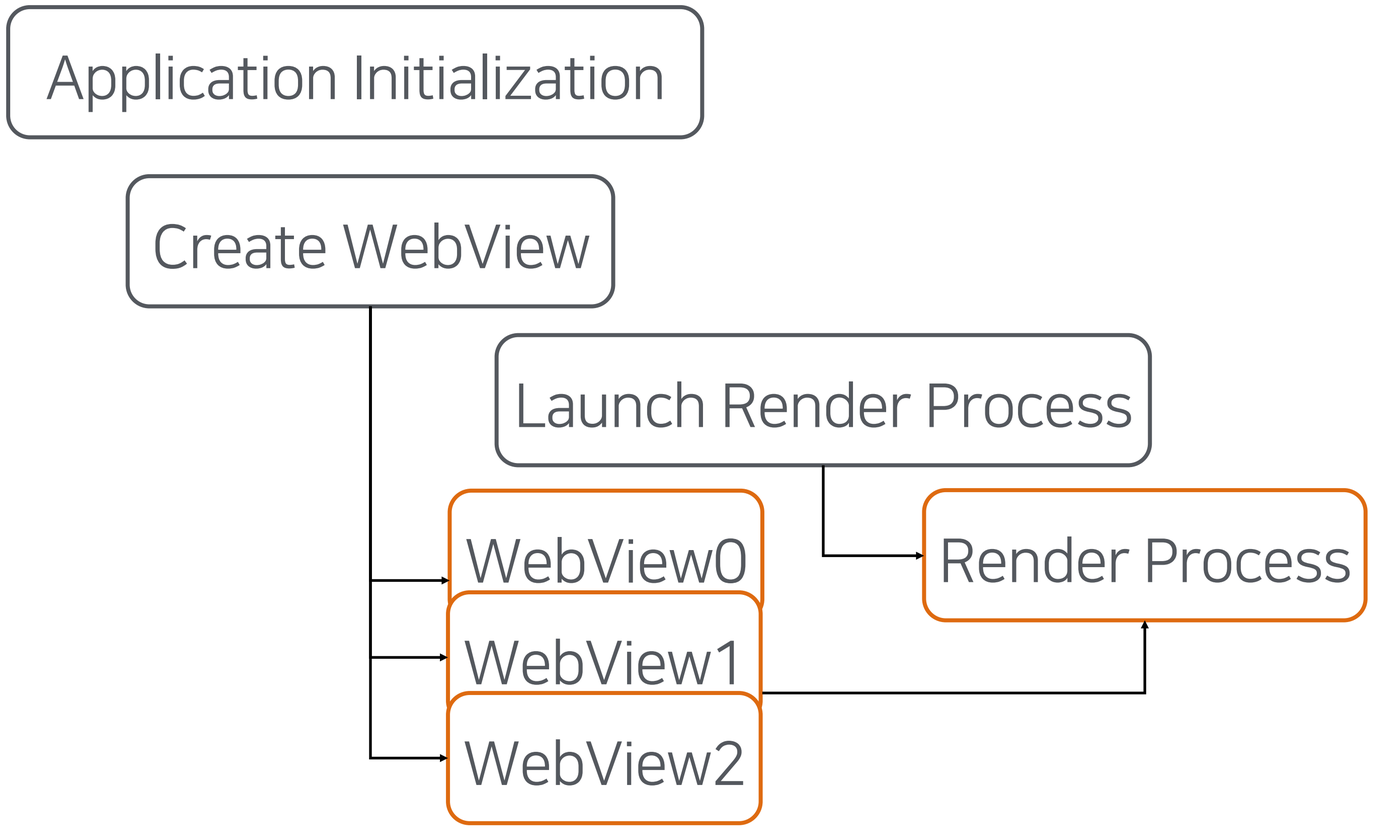
DEVIEW
2019

2.사용성 개선을 위한 다중 웹뷰 활용

2.1 웹브라우저 엔진의 다중 웹뷰 생성

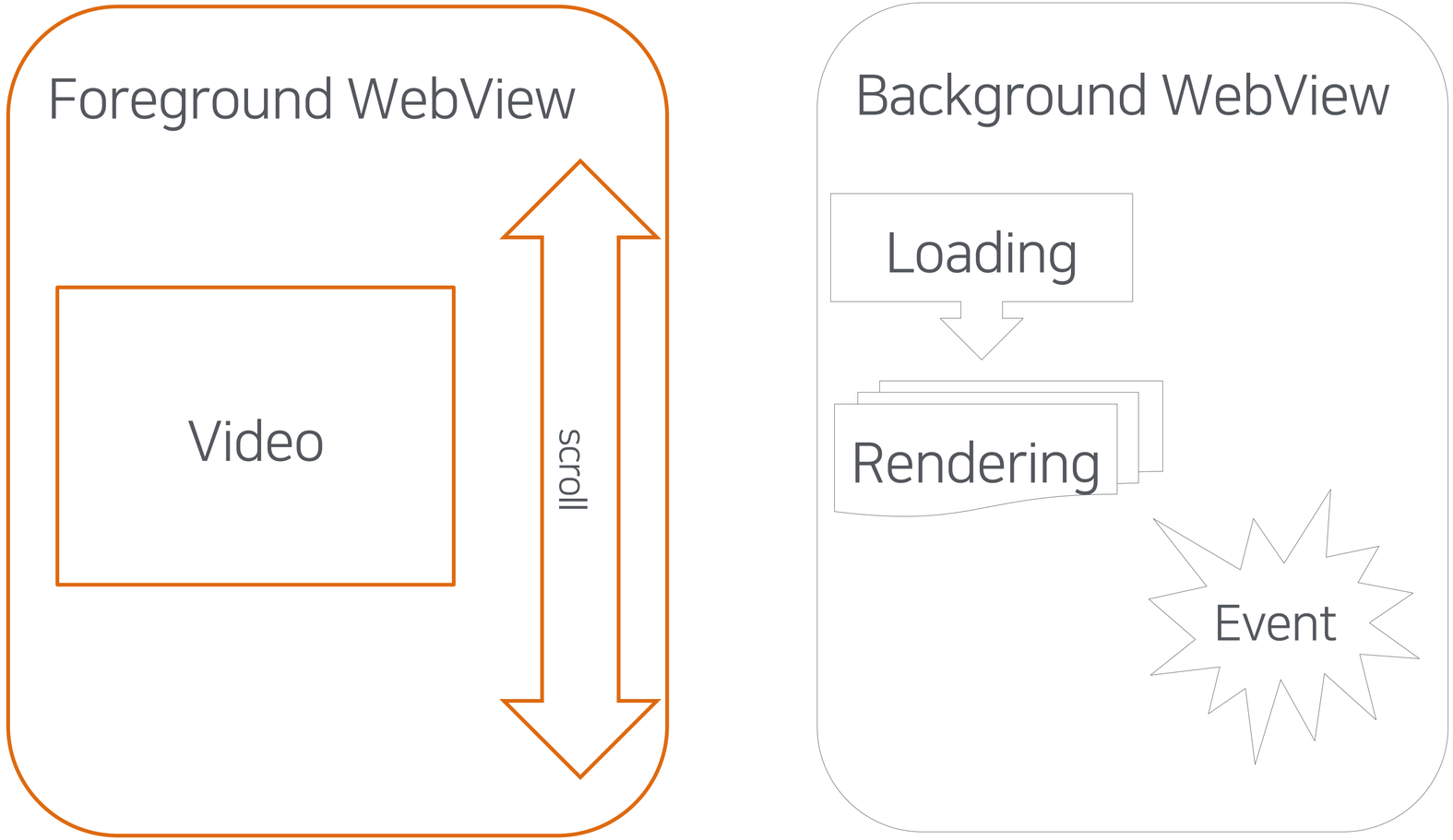
앱 초기화 단계 시 복수의 웹뷰 사전 생성

사후 render process에 등록

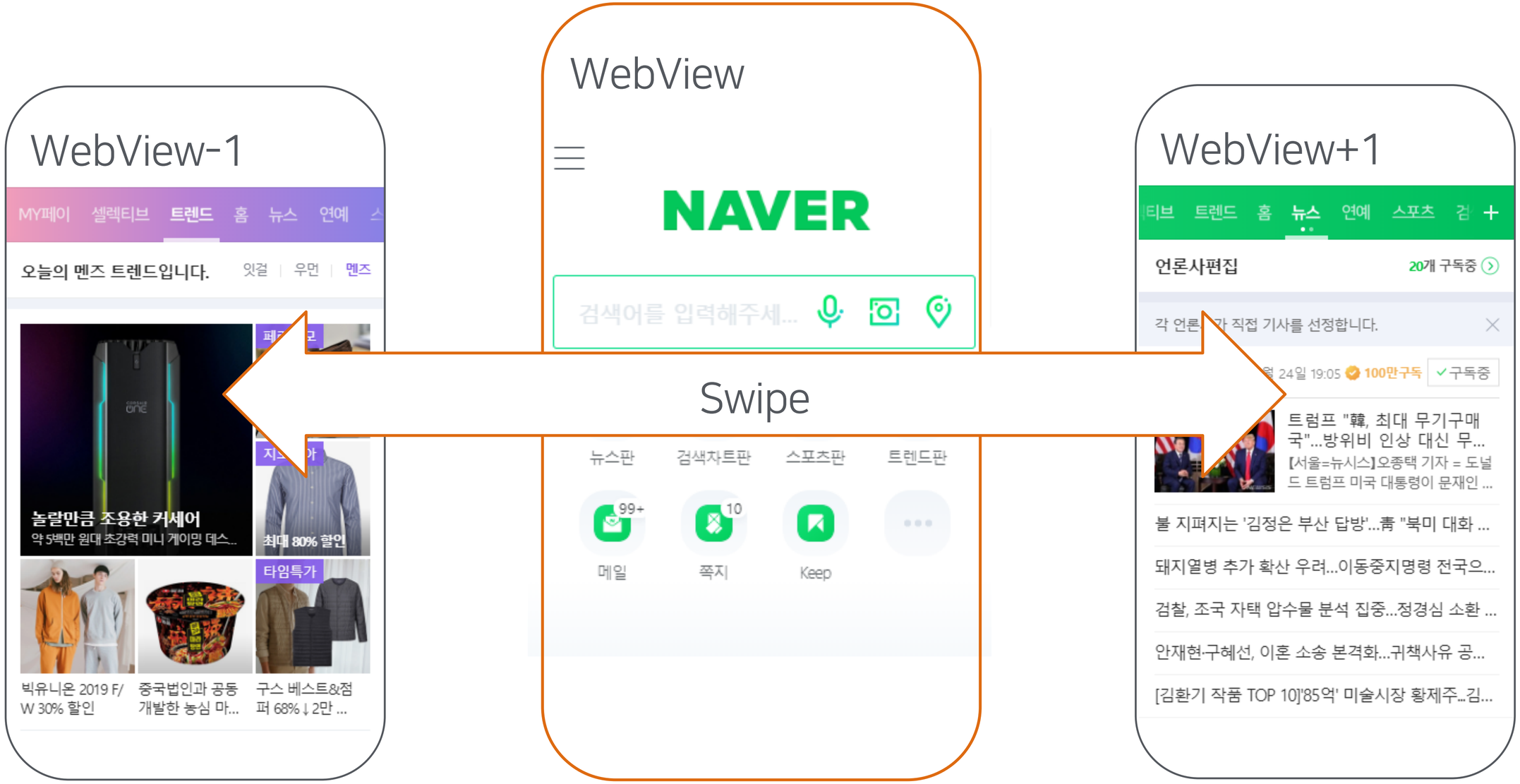


2.2 다중 웹뷰의 효과

백그라운드에서 로딩 및 rendering 수행하여 반응성 개선



2.3 다중 웹뷰에 의한 Background 로딩



DEVIEW
2019

3.앱 구동 성능 분석

3.1 다중 웹뷰 생성에 따른 부담

DEVIEW
2019

브라우저 엔진 초기화

Renderer 프로세스 생성

웹뷰 생성

웹페이지 로딩

3.1.1 Profiling 환경

앱 launch 부터 웹페이지 로딩 완료 까지

1. Synchronous initialization => 1st webview creation => Navigation start => onload event end
2. Cold / Warm launch

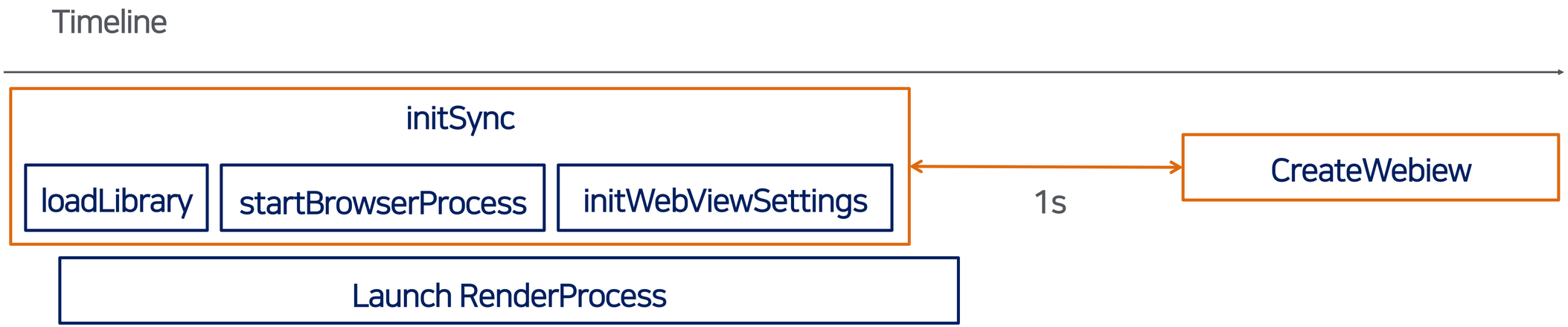
네이버 메인 및 주제판 (뉴스, 연예, 스포츠, 검색)

단말기 (H: high end, M: middle tier , L: low end)

3.1.2 웹뷰 생성 지연

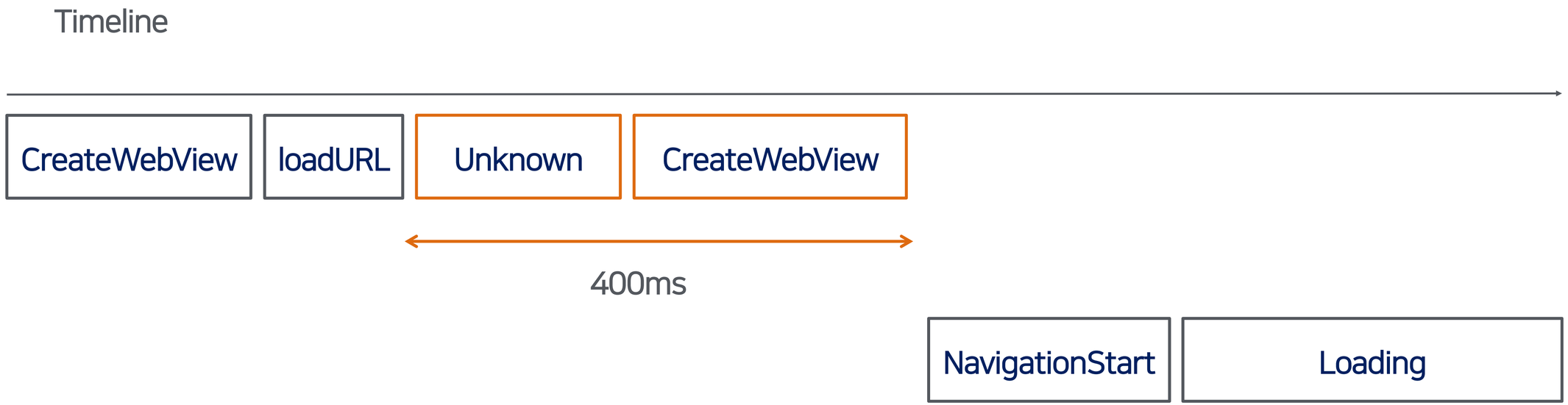
DEVIEW 2019

Synchronous 초기화 부터 첫번째 웹뷰 생성까지 (L, M)



3.1.3 웹페이지 로딩 지연

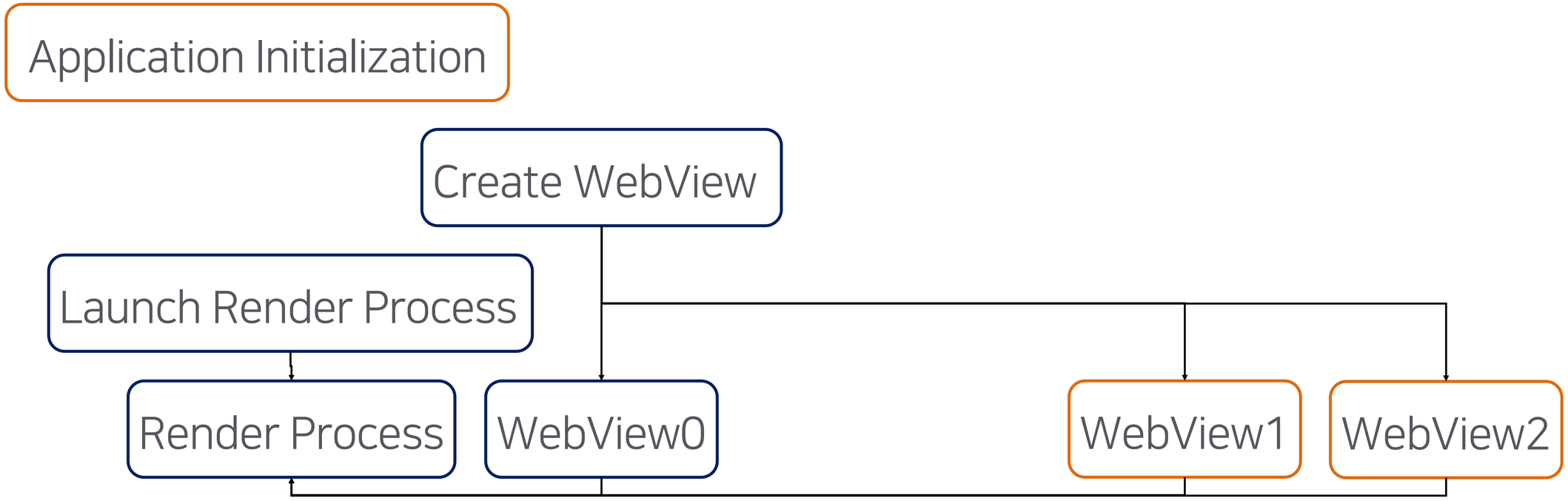
브라우저 프로세스의 provisional load가 생성한 IPC에 의해
Renderer 프로세스의 navigation이 시작되기 까지 지연 (M)



3.1.4 비동기 초기화 및 예비 웹뷰 생성 지연

앱의 비동기 초기화 적용 : 400ms / 10% 개선 기대 (L)

Background WebView의 생성 연기 : 400ms / 13% 개선 기대 (M)



3.2 웹페이지 로딩의 병목 구간

CSS parsing

JavaScript 실행

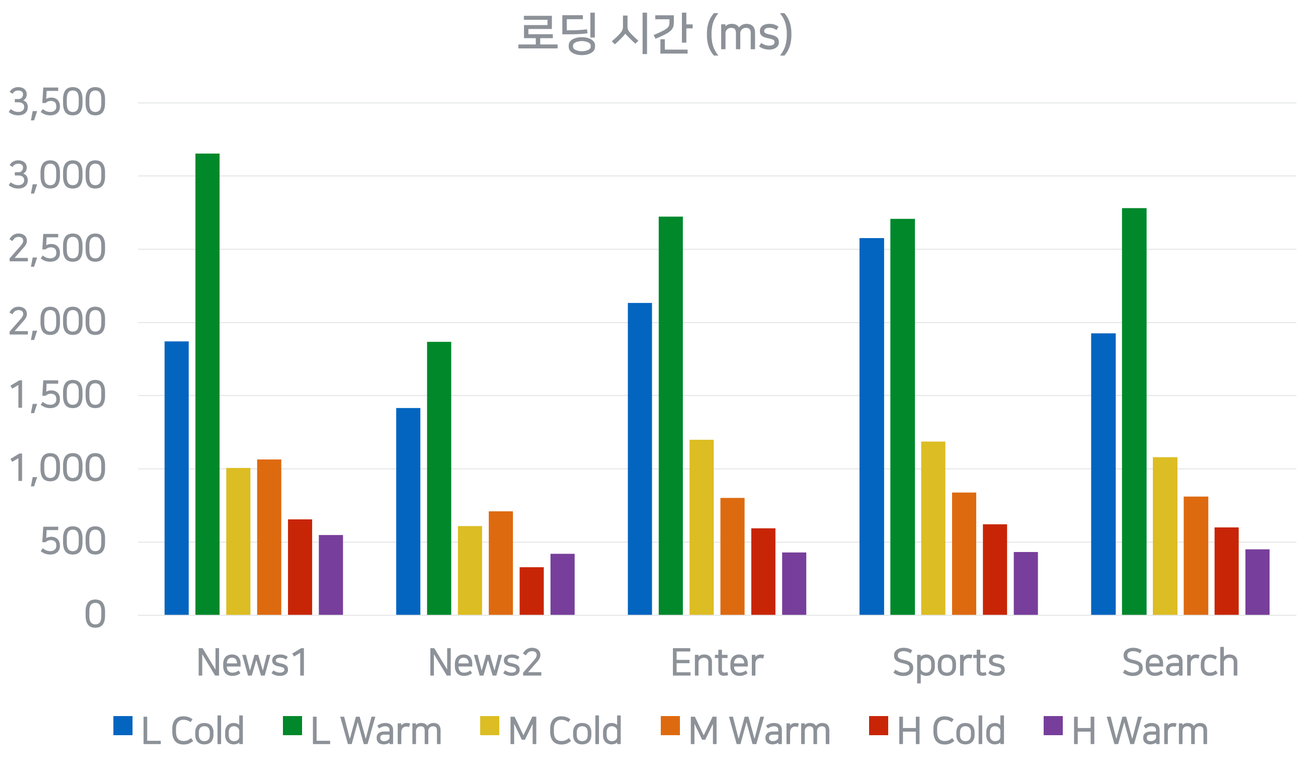
Layout 변동

Resource request에 대한 response 지연

3.2.1 웹페이지 로딩 성능

DEVIEW 2019

Navigation start에서 onload event end까지

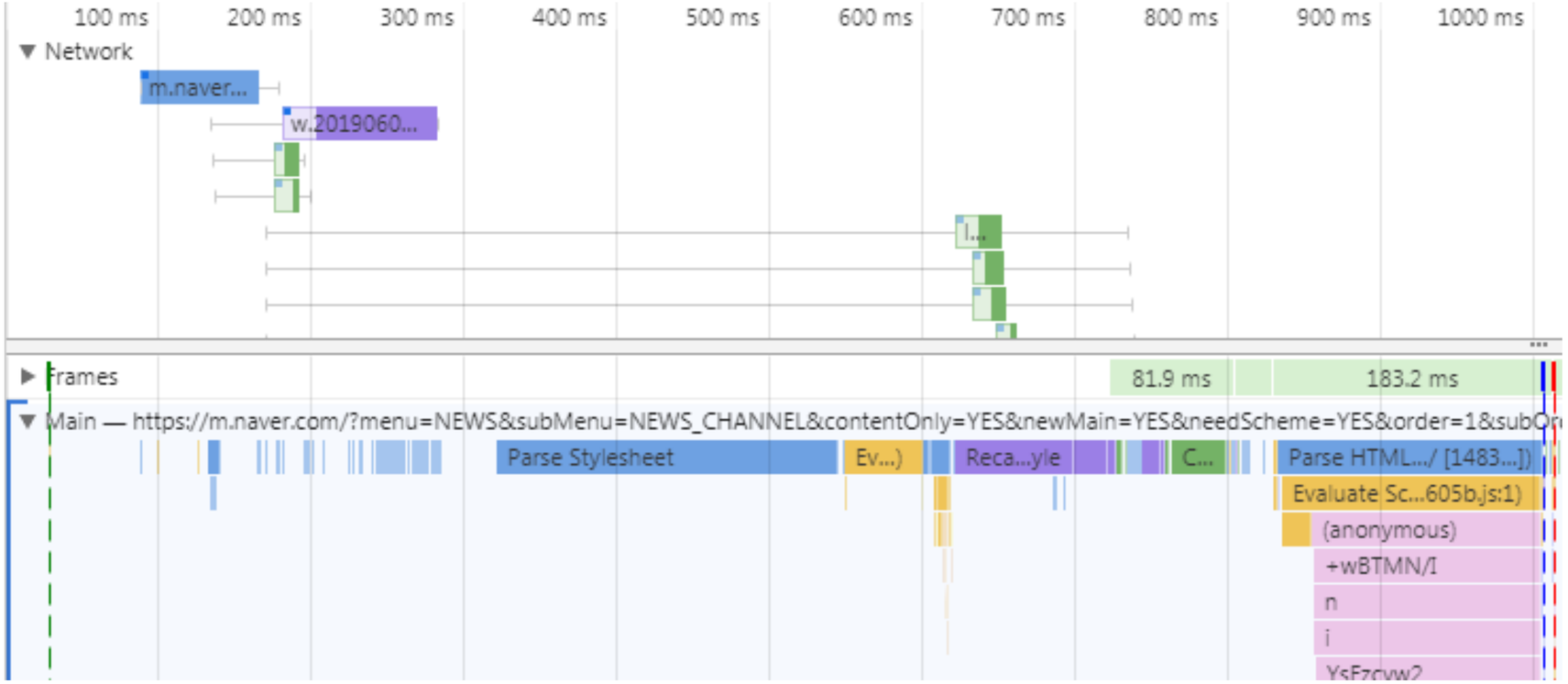


3.2.2 중복된 CSS Parsing

DEVIEW 2019

각 주제판에서 같은 CSS를 매번 parsing 하면서 긴 시간 소모

loading은 caching이 되는데, parsing은?



3.2.3 높은 비용의 JavaScript

DEVIEW 2019

Layout의 큰 변화를 야기하는 JavaScript

DOM node에 대한 전면적인 변경

Load event 이후로 지연?

부분적인 layout 조정?

3.3 사용자 중심의 성능 지표

W3C navigation timing APIs

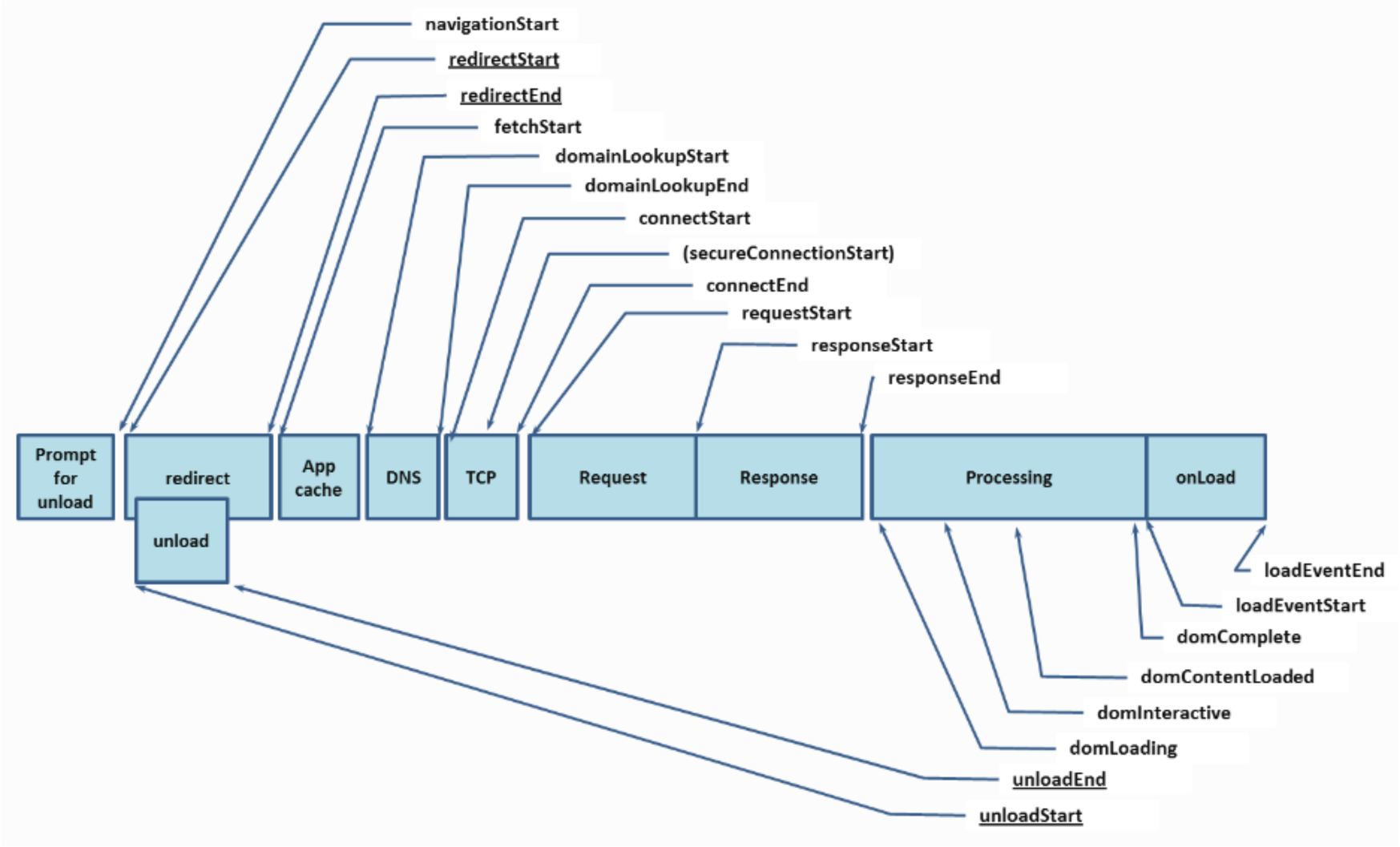
RAIL 성능 모델

First meaningful paint

Time to interactive

User metrics histogram

3.3.1 Navigation Timing



https://www.w3.org/TR/navigation-timing/

3.3.2 RAIL 성능 모델

Response: 사용자 입력으로 부터 paint 까지 100ms 이내

Animation: 각 frame의 갱신은 16ms 이내

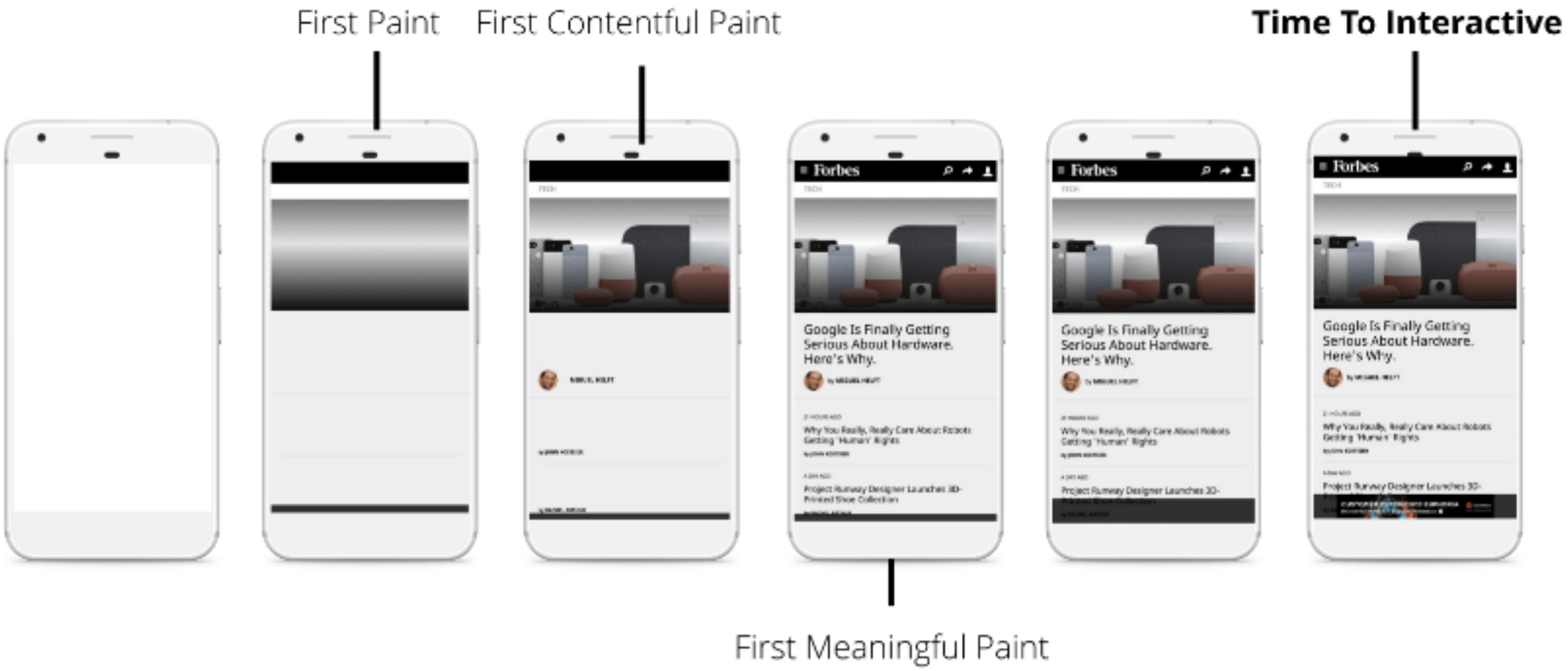
Idle: 유휴 시간에 처리할 background task는 50ms 이내

Load: first meaningful paint까지 1000ms 이내

https://developers.google.com/web/fundamentals/performance/rail

3.3.3 Paint 성능 지표의 의미

DEVIEW 2019



3.3.4 First Meaningful Paint

DEVIEW 2019

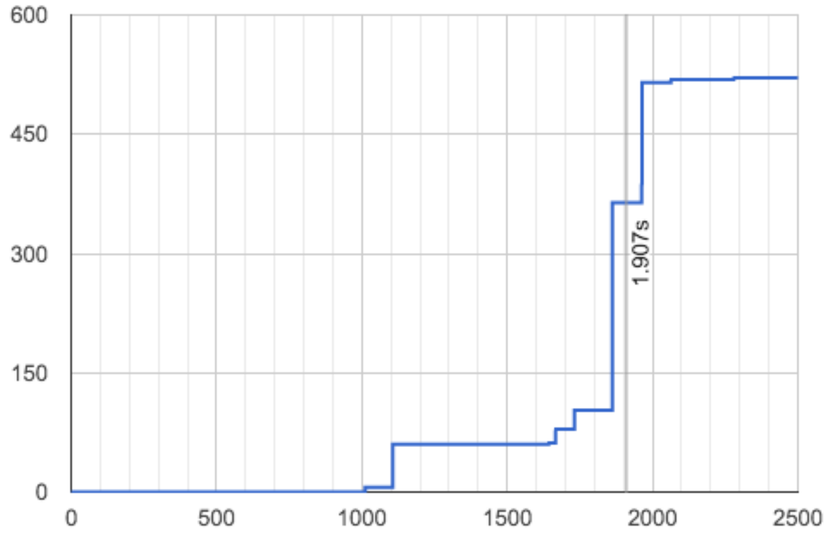


Figure 1: Number of layout object added during page load of Google SRP

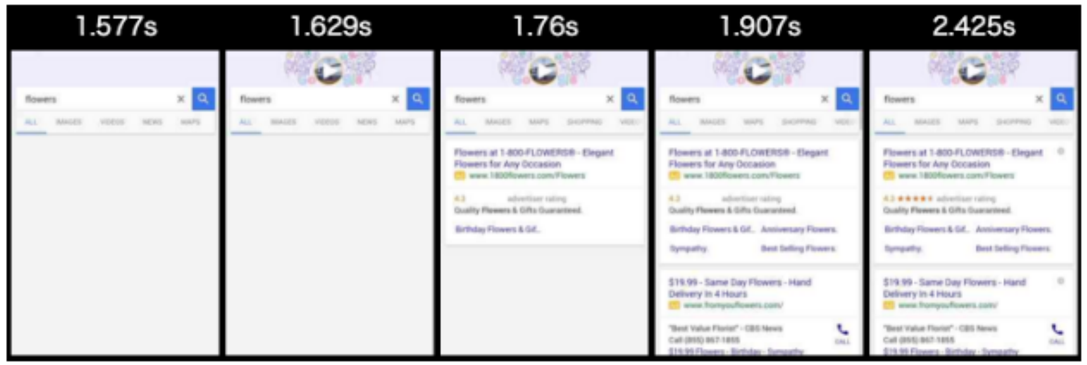
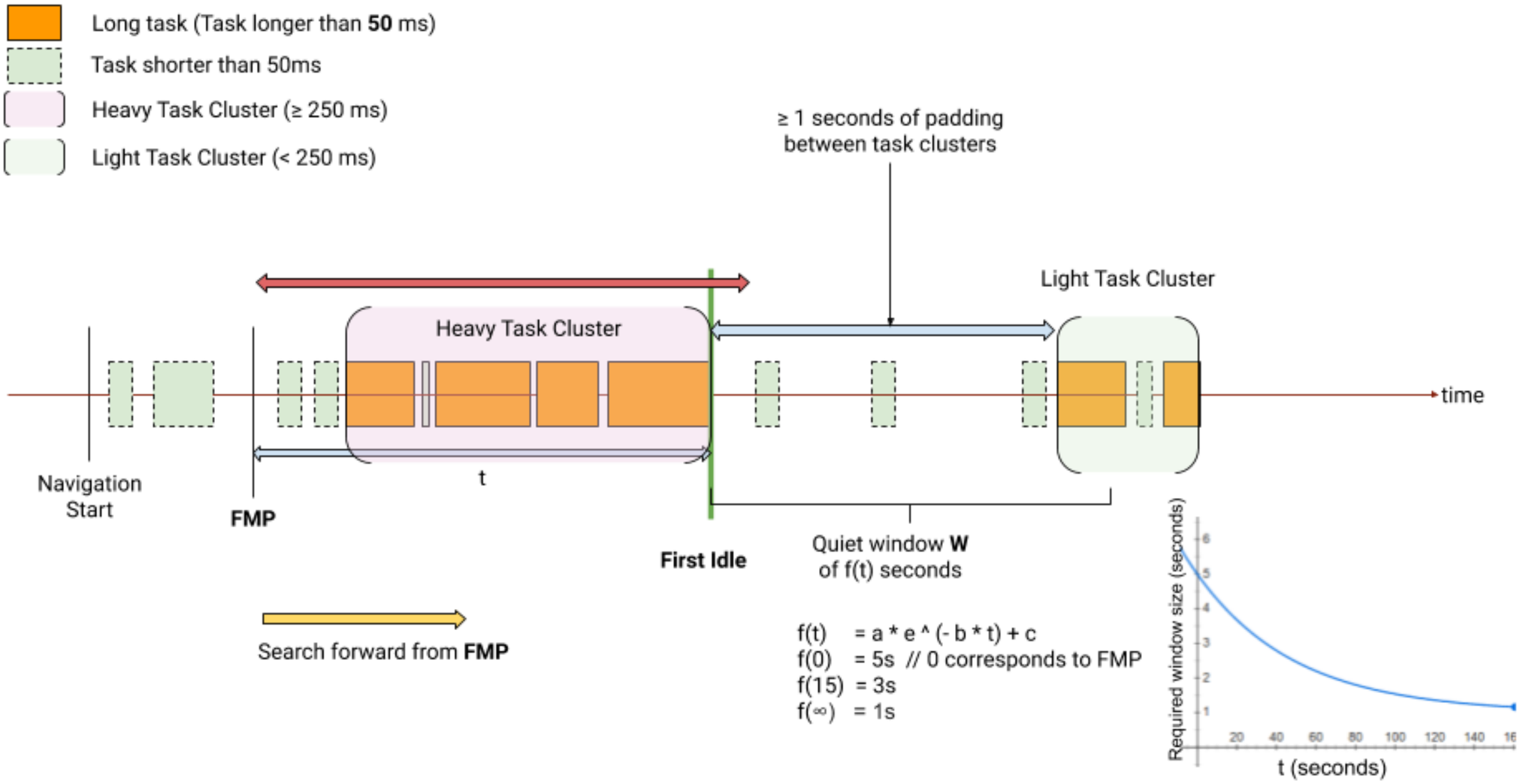


Figure 2: Visual progress of the page load

페이지의 layout이 가장 급격히 변하는 순간

3.3.5 Time to Interactive (1st CPU Idle)

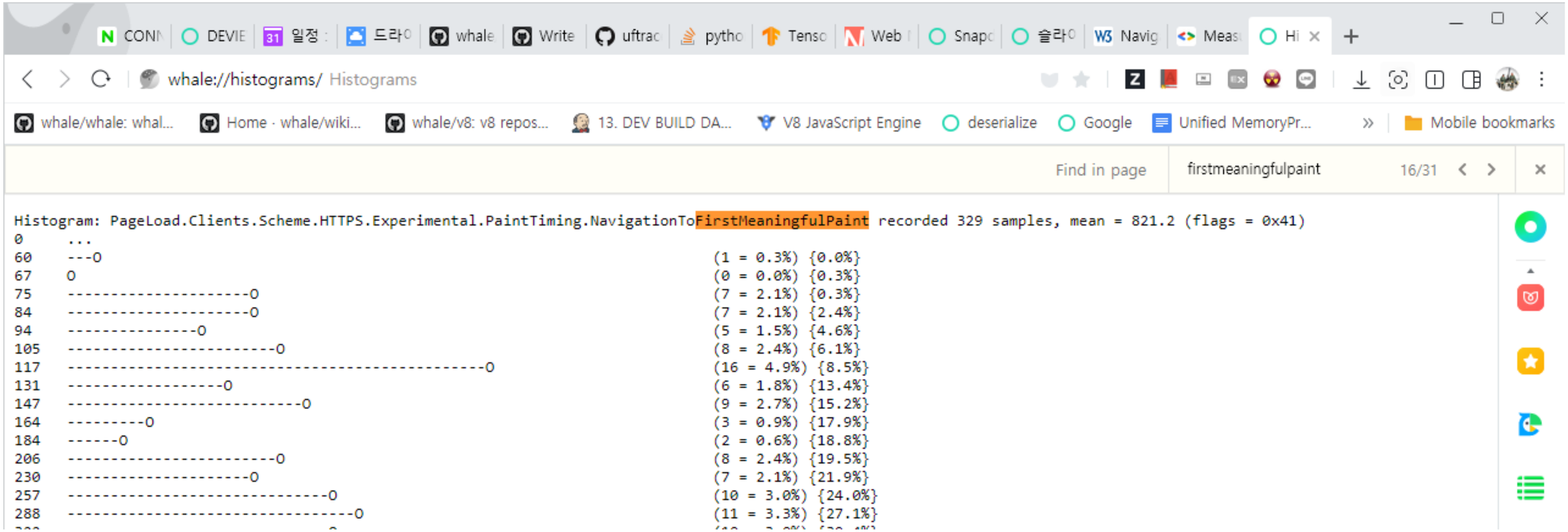
DEVIEW 2019



3.3.6 Histogram

DEVIEW 2019

Chromium 브라우저의 실행 정보가 누적된 in-memory DB



3.4 HTTP Cache의 활용도 및 영향력

URL query to disk cache

가변적인 캐시 용량 (20~200MB)

Eviction heuristics의 맹점

- sort(LRU time x entry sizes)

3.4.1 HTTP Cache Profiling

Chromium 브라우저에서 네이버 메인의 각 주제판 방문

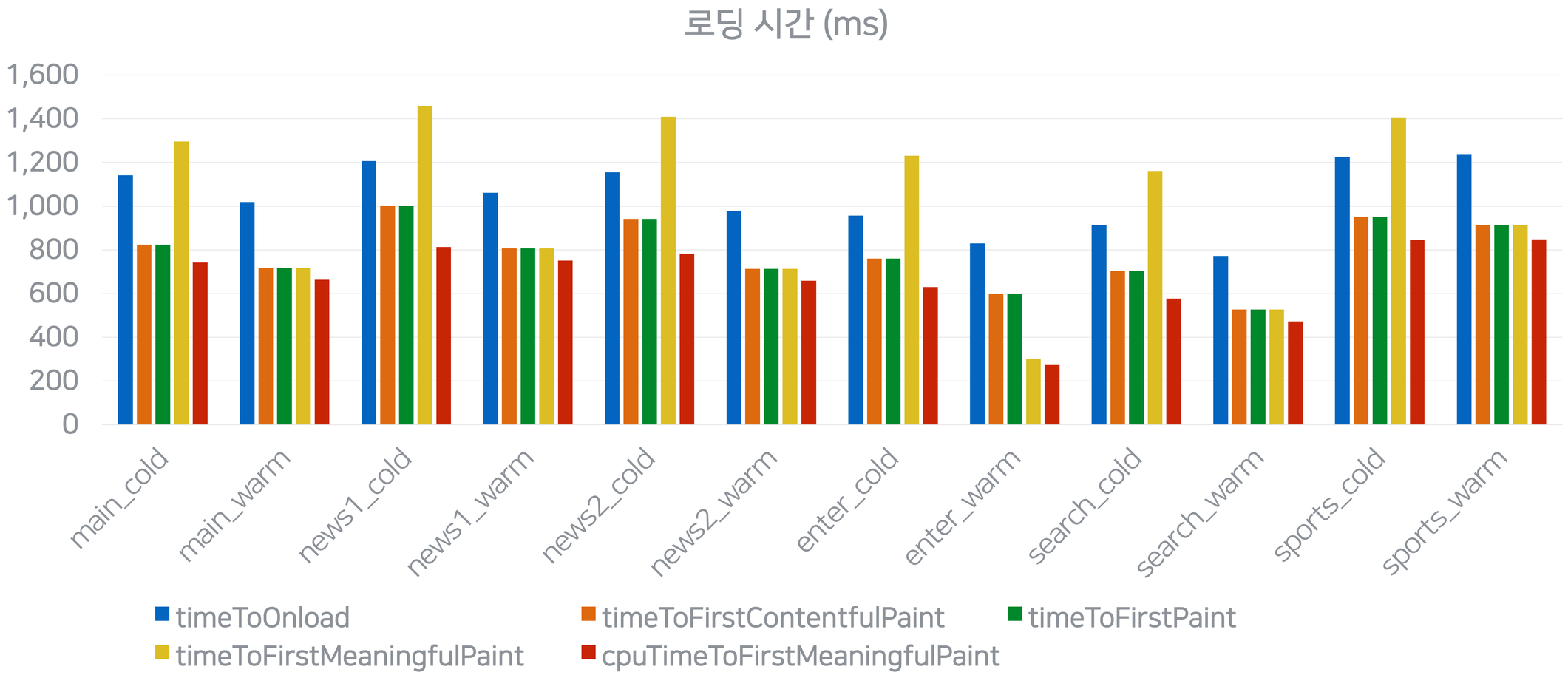
- warm-up => 1차 => ~3 hours => 2차 => 3차

Histogram에서 cache hit ratio (OpenIndexState) 확인

- 1차 iteration : 0.64
- 2차 iteration : 0.45
- 3차 iteration : 0.63
- max cache size : 200MB
- 3차 iteration 후 cache size : 92MB

3.4.2 HTTP Cache 로딩 성능

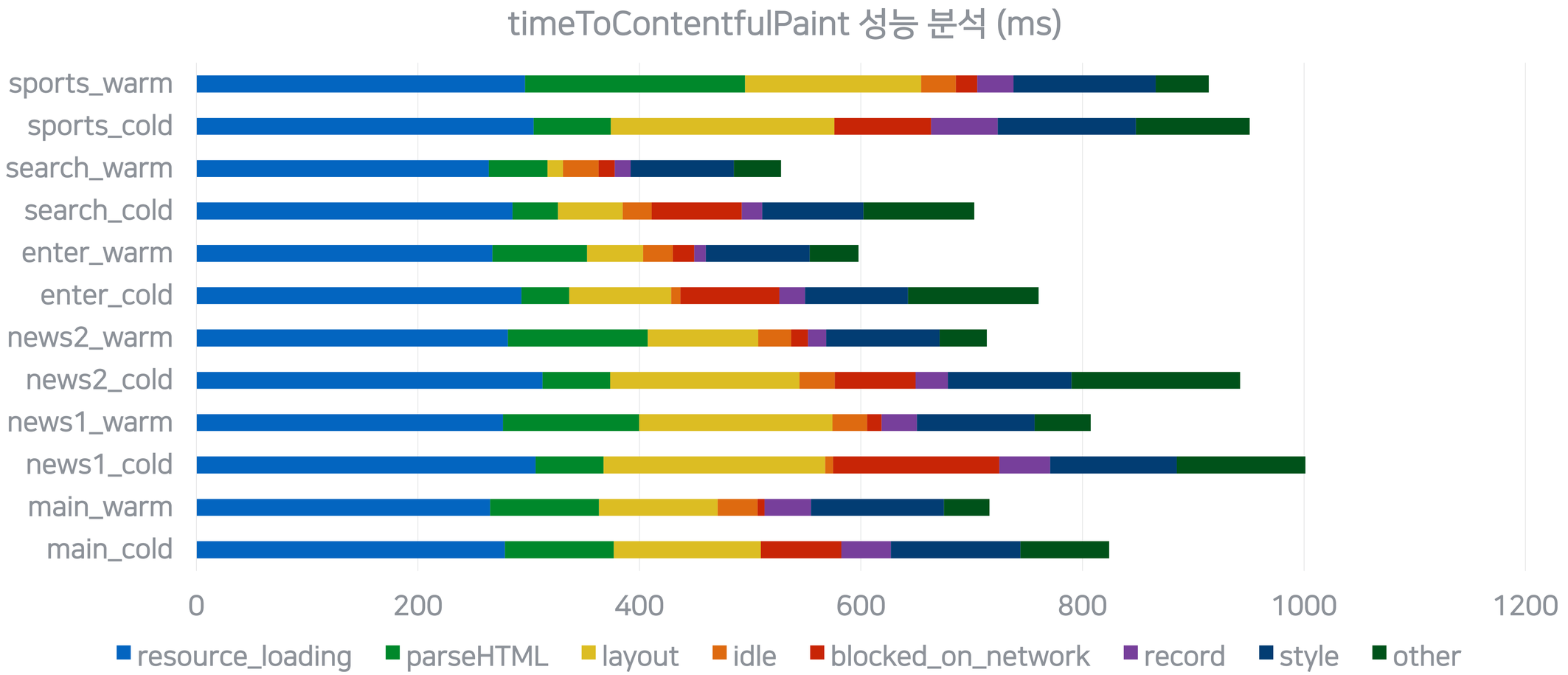
Telemetry로 record한 웹페이지를 replay



3.4.3 HTTP Cache 성능 분석

DEVIEW 2019

네트워크 지연 감소 확인

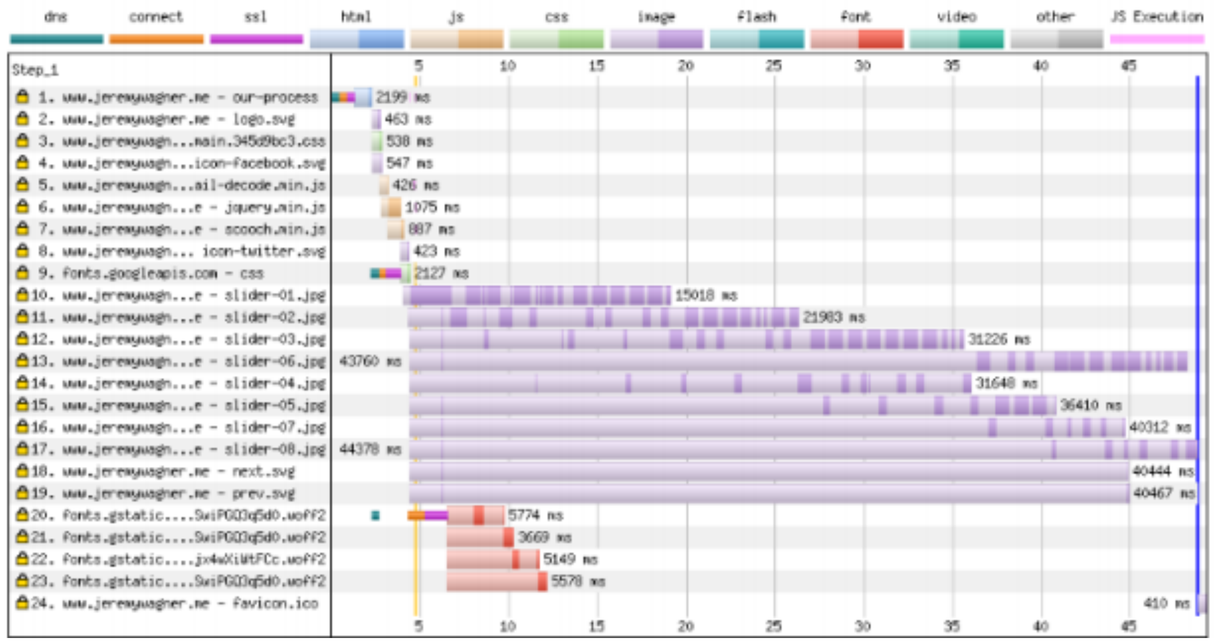


3.5 이미지 로딩의 부하

저사양 단말에서의 불필요한 고해상도 이미지 로딩

네트워크 전송 및 이미지 decoding에 따른 부담

단말 최적화 이미지 전송 (proxy vs. client hint)



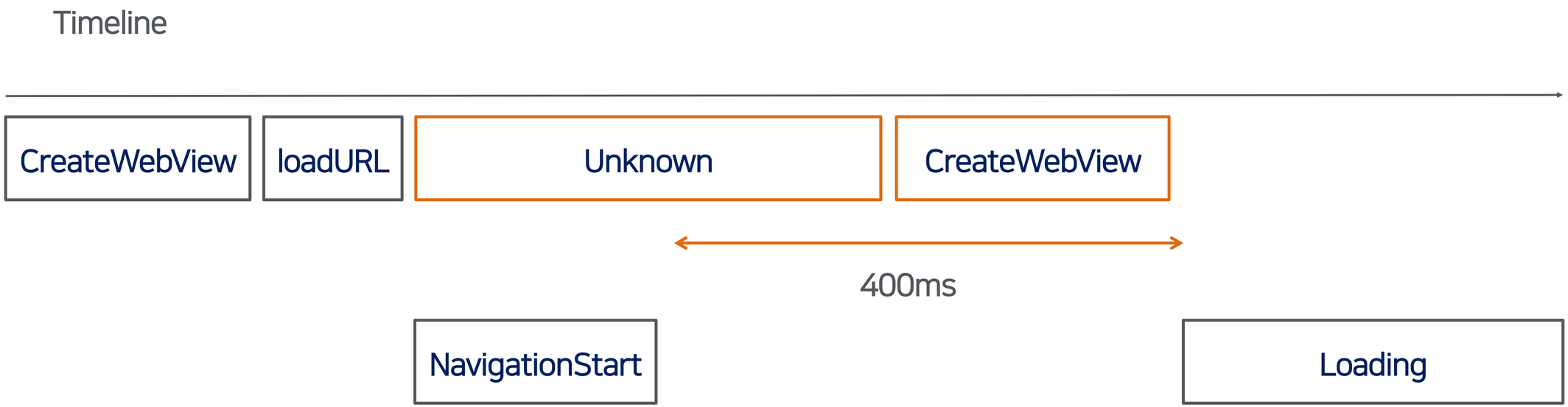
4.성능 개선 방안 개발 및 결과

4.1.1 Background 웹뷰의 지연 생성

Renderer 프로세스에서 로딩을 더 일찍 시작하도록 초기화 순서 조정

Background 웹뷰의 생성을 늦췄지만,

로딩에 필요한 리소스의 초기화가 늦어 종료 시점은 동일

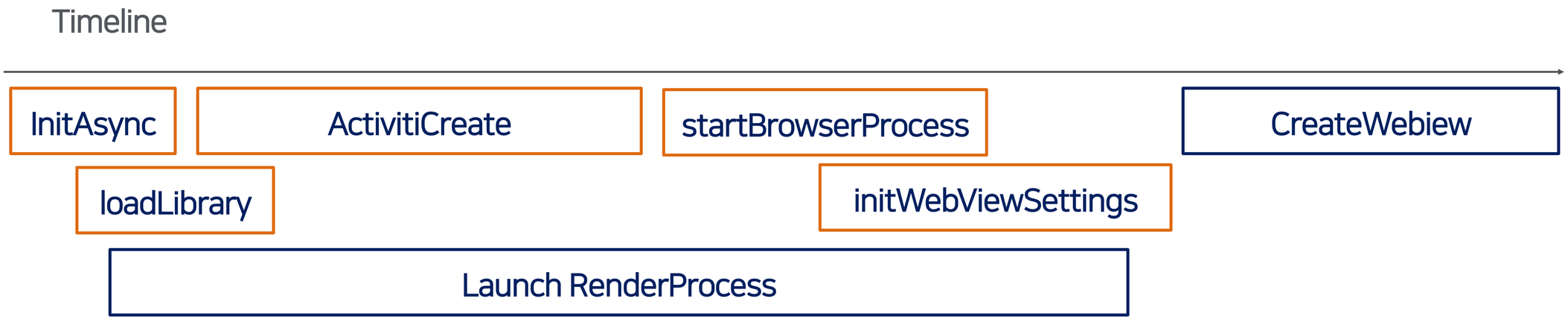


4.1.2 브라우저 엔진 비동기 초기화

비동기 초기화 task의 시작 지연

별도 thread 생성 부담

callback 의존성



4.2.1 Privileged HTTP Cache의 구현

지정된 URL 패턴의 resource에 대하여 일정 보존 기간 보장

- https://privileged.http.cache/ *

지정된 referer로 요청된 이미지에 대하여 일정 보존 기간 보장

- third party로 부터 API를 통해 유입되는 image
- Resource URL의 확장자로만 image 여부를 판별할 수 있을까?

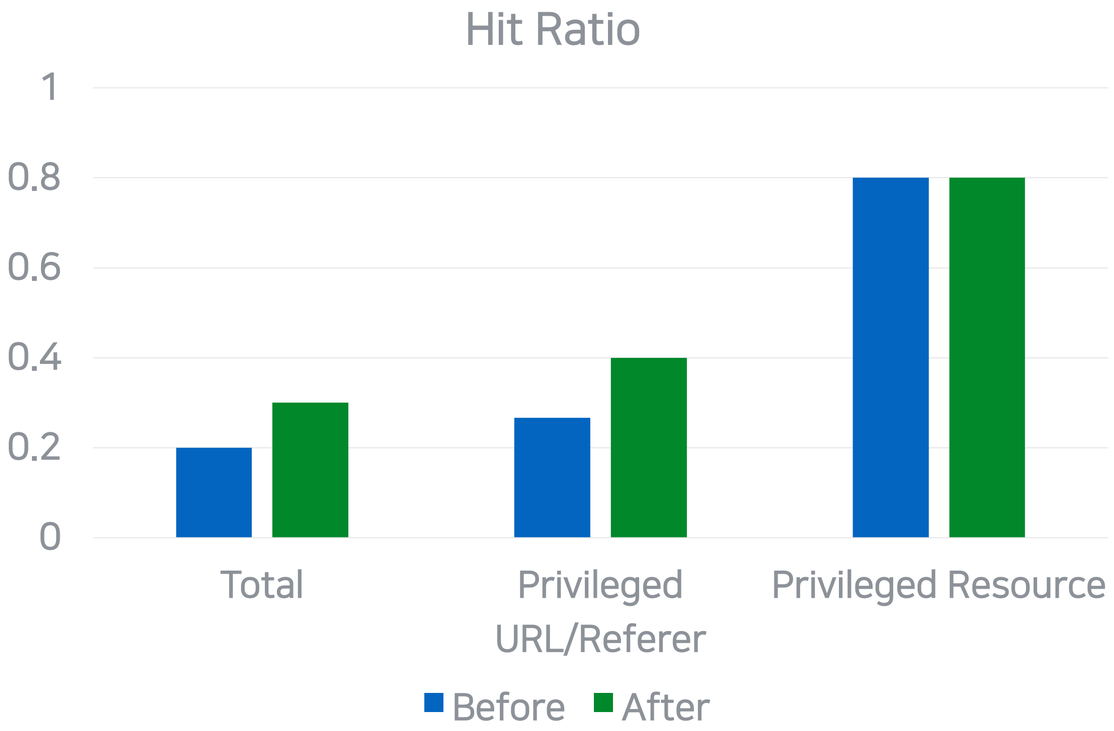
지정된 resource type(js/css)에 대하여 추가 보존 기간 보장

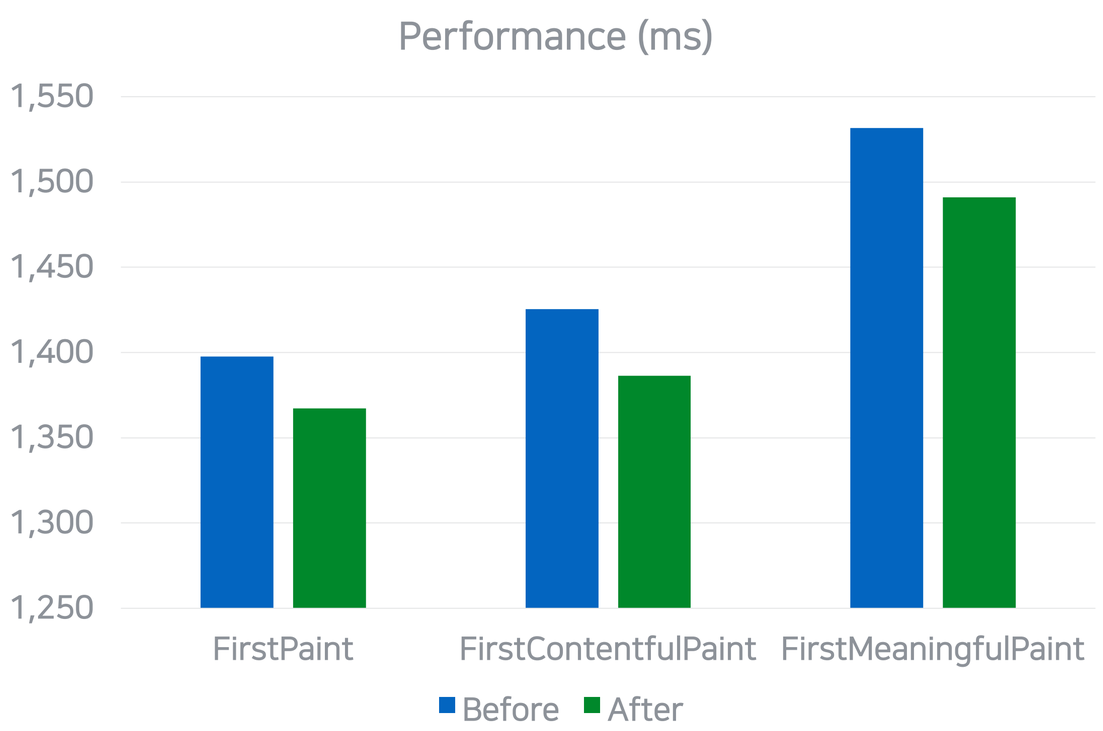
4.2.2 Privileged HTTP Cache 실험 결과

DEVIEW 2019

이미지 캐시 hit ratio 14% 향상 (L)

Paint 성능 30~40ms 향상 (L)



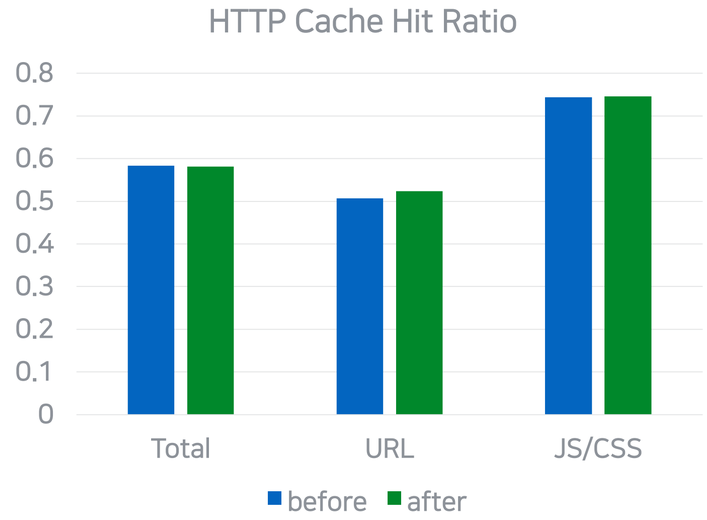


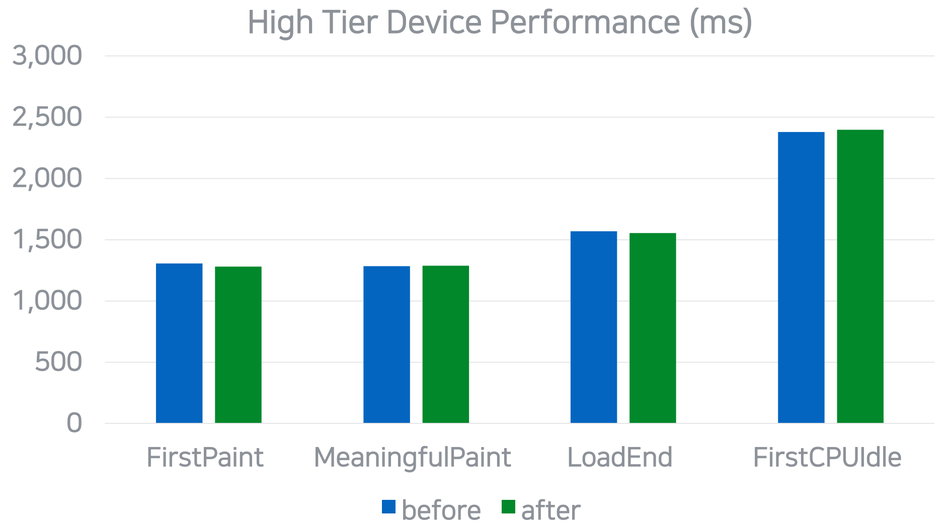
4.2.3. Privileged HTTP Cache 적용

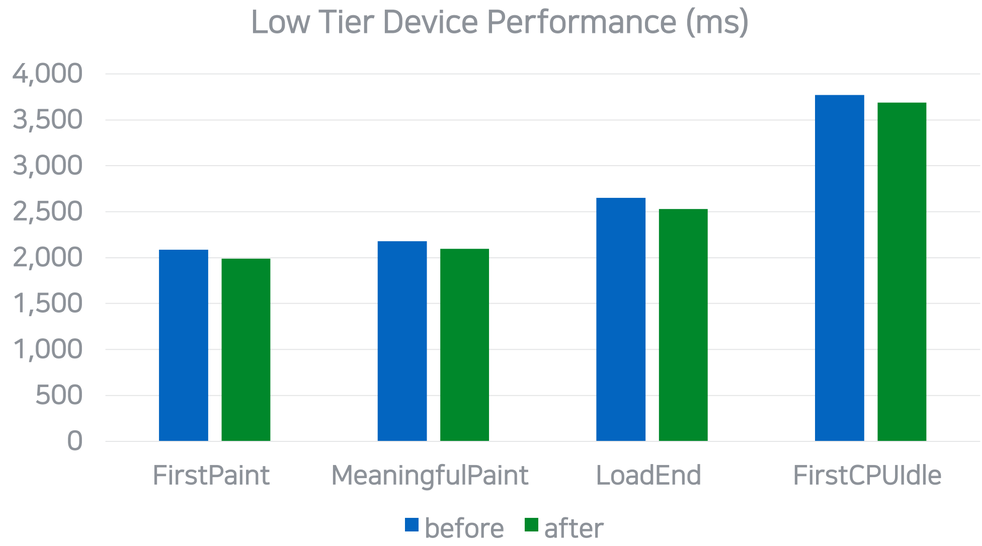
DEVIEW 2019

Low tier device의 성능 지표

80~120ms 개선







4.3.1 Cross WebView CSS cache 구현

Render 프로세스 안에서 모든 WebView가 접근 가능하도록

- Cross thread map으로 global cache 구현

CSS rule set은 document와 life cycle을 함께 하므로

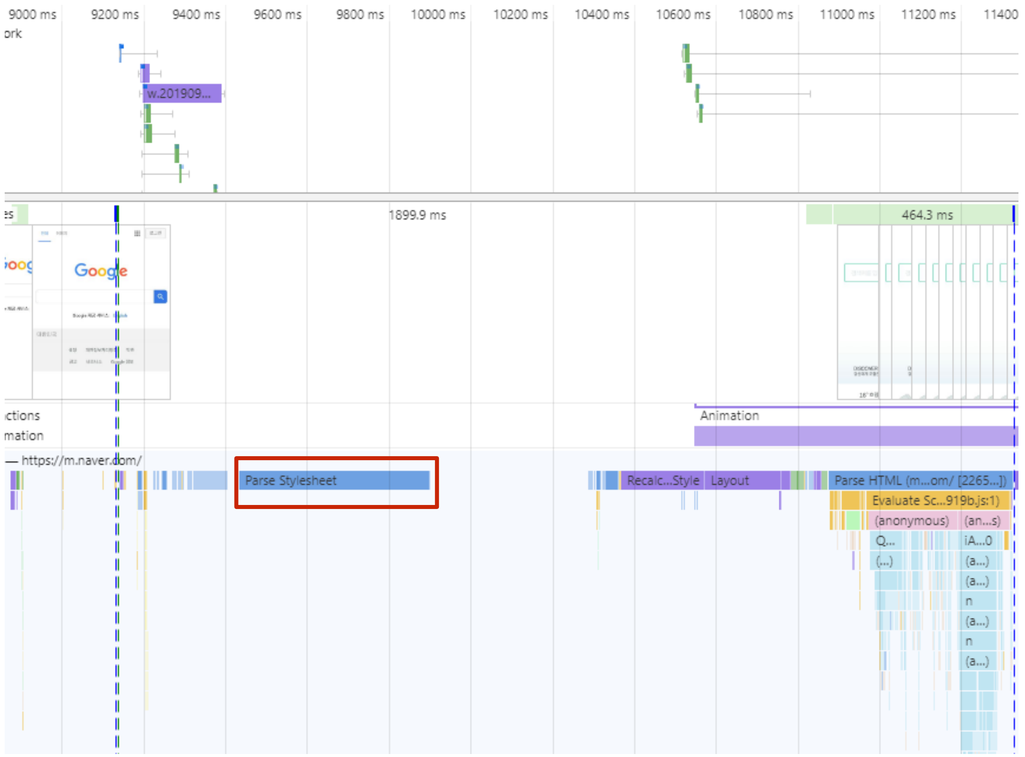
- Parsing된 rule set을 cache로 복사하여 garbage collection 대상에서 제외
- 메모리 최적화를 위하여 memory pressure와 연동 및 지정된 URL로 제한

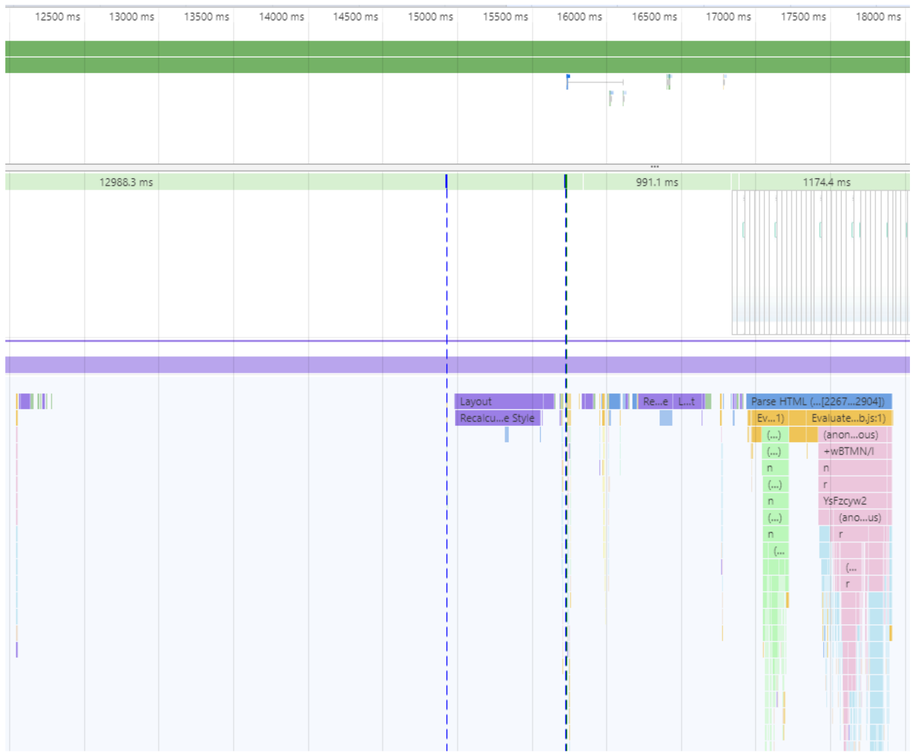
4.3.2 Cross WebView CSS cache 효과

DEVIEW 2019

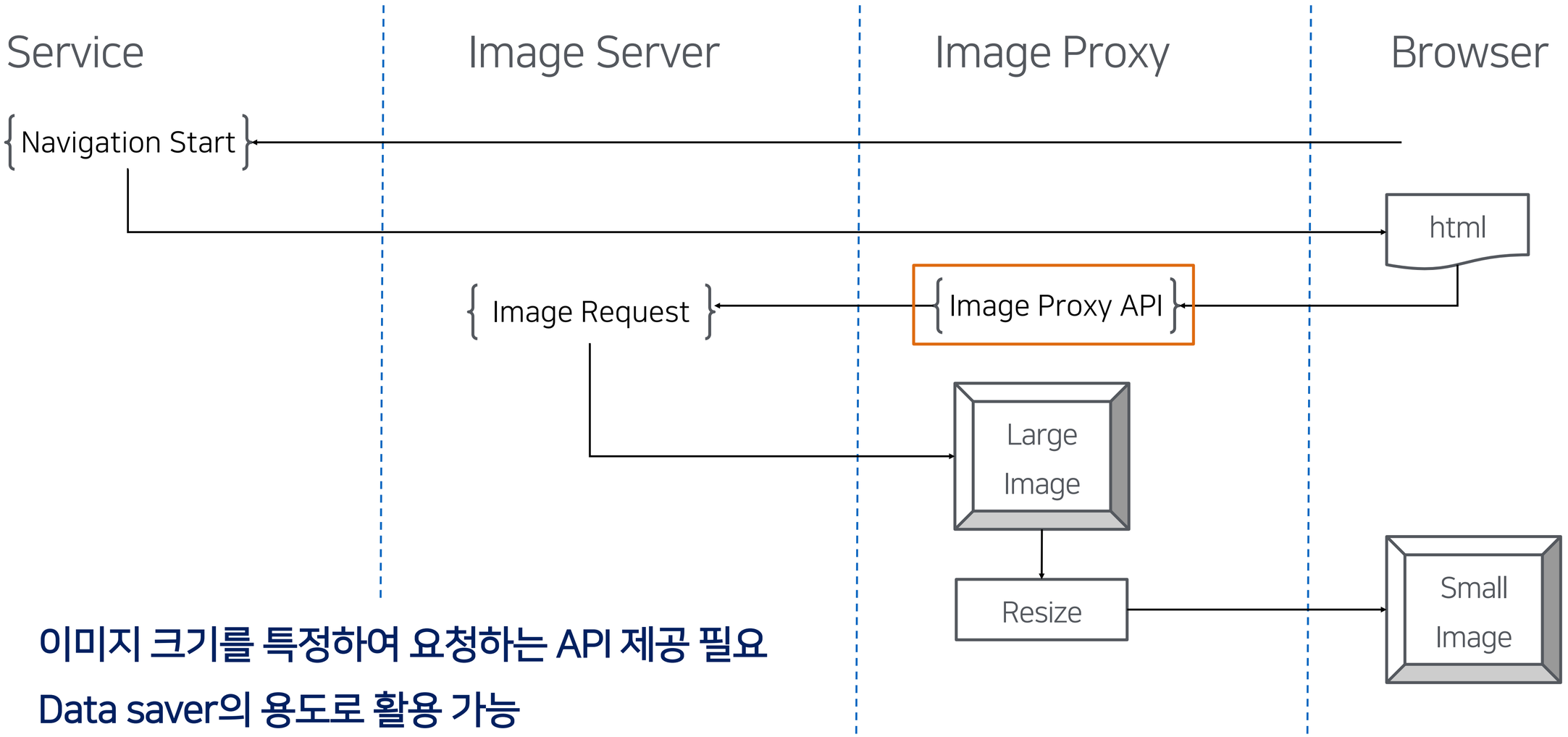
467ms의 CSS parsing이 최초 한번만 발생 (L)

네이버 주제판 당 로딩시간 감소 (L: 22%, M: 17%, H: 18%)





4.4.1 Proxy 기반 이미지 최적화



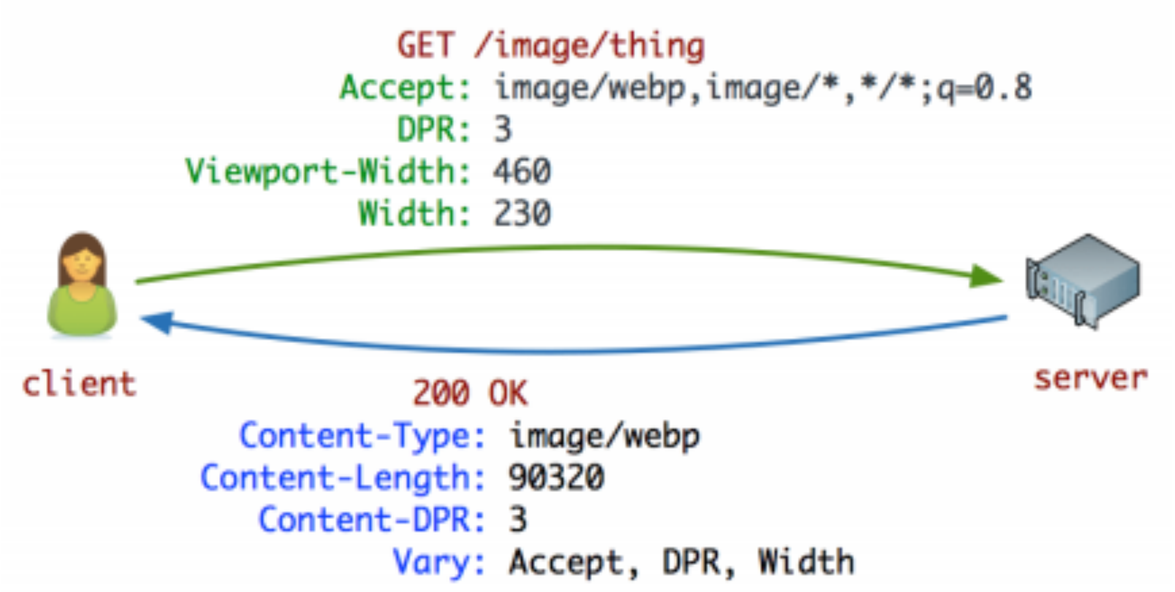
이미지 크기를 특정하여 요청하는 API 제공 필요

Data saver의 용도로 활용 가능

4.4.2 Client hint 기반 이미지 최적화

DEVIEW 2019

단말의 화면 해상도 정보를 header에 실어 서버에 요청



적합한 크기 및 품질의 이미지를 담은 응답을 처리

https://httpwg.org/http-extensions/client-hints.html

4.4.3 이미지 품질과 성능의 접점

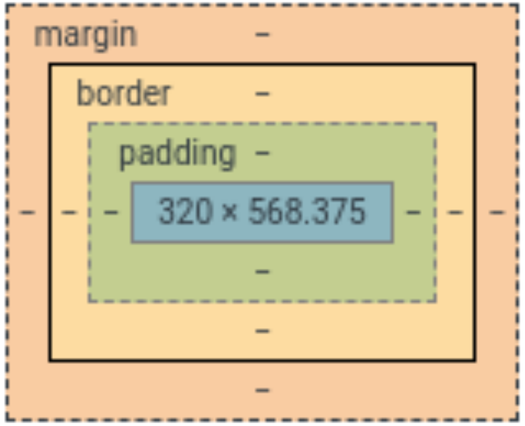
DEVIEW 2019

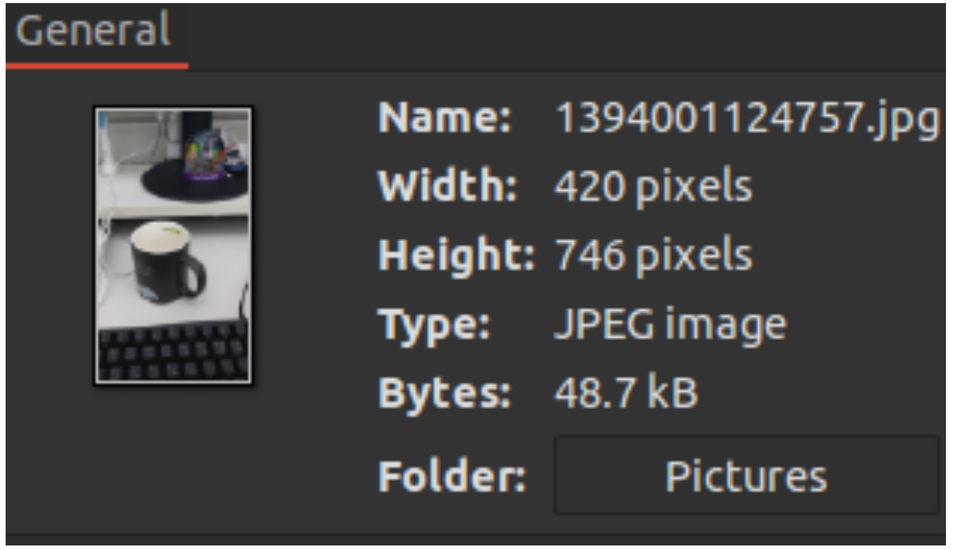
단말에서 원본보다 큰 이미지로 조정

Layout image size x Device Pixel Ratio (DPR)

320 x 3 = 960

실제 pixel 수 보다 큰 이미지의 유효성?





DEVIEW
2019

Q & A

DEVIEW
2019

Thank You