

CSS3로 60FPS 애니메이션 달성

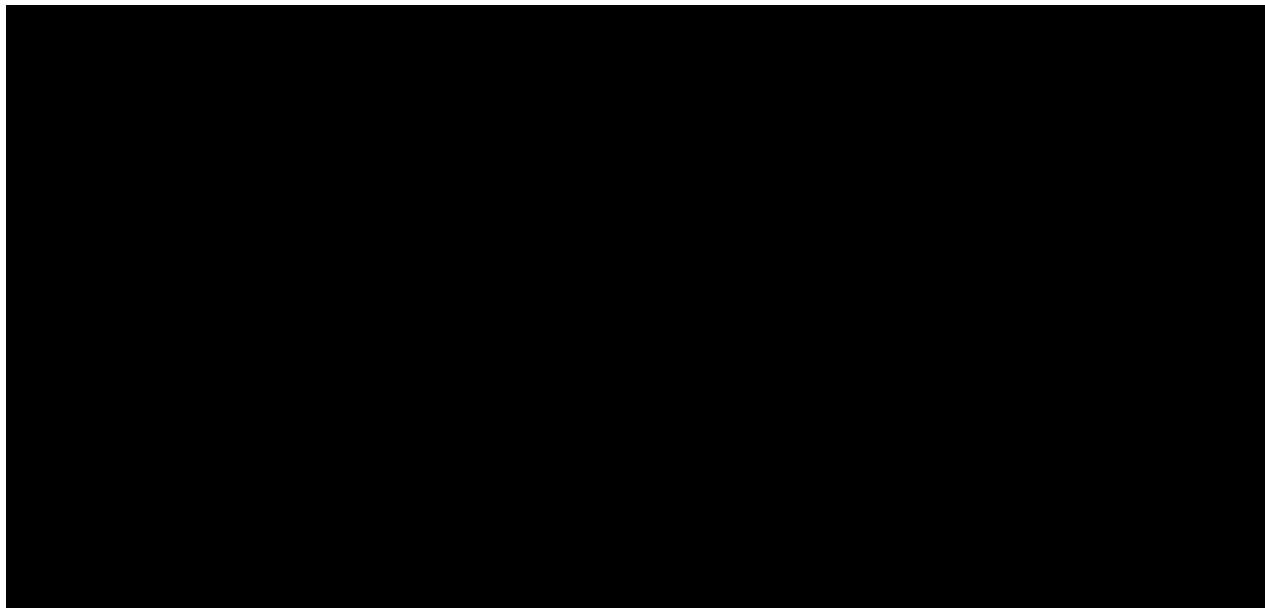
시작하기 전, 학습 이유

모두가 **모바일**에서 **CSS3** 애니메이션을 사용하고 있지만, 제대로 사용하지 못하고 있음

기기 사양이 매우 다양하기 때문에 코드를 최적화하지 않으면 사용자에게 낮은 수준의 사용자 경험을 하게 할 것

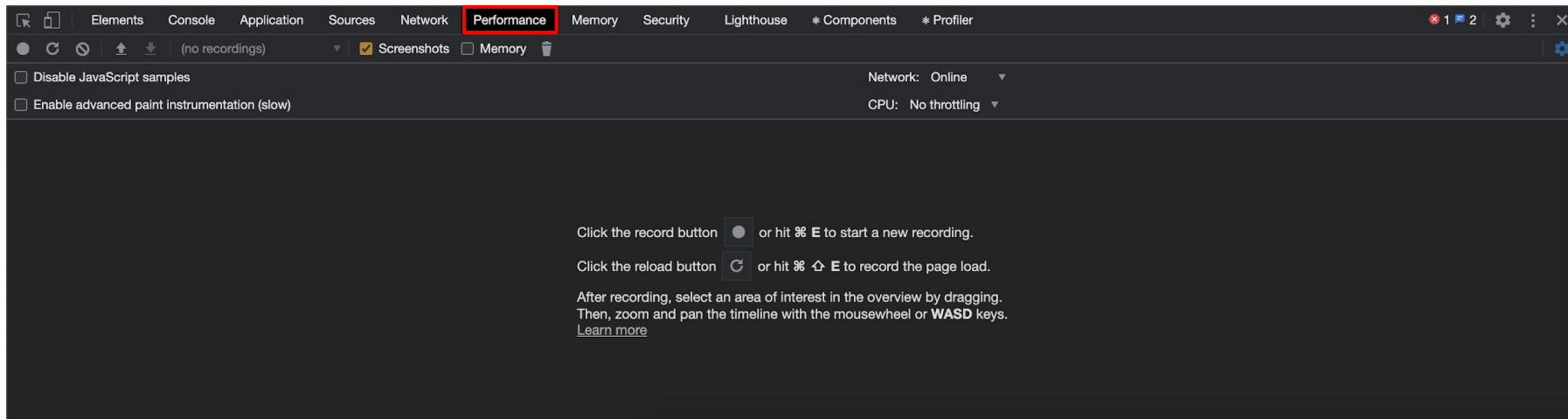
결론은 모바일 애니메이션을 위한 학습!

60FPS* Animation과 아닌 것의 차이



Performance 패널

런타임 성능*을 분석할 수 있도록 Chrome 개발자 도구에서 제공
해당 패널을 통해 페이지의 응답, 애니메이션 및 유틸* 단계 분석 가능



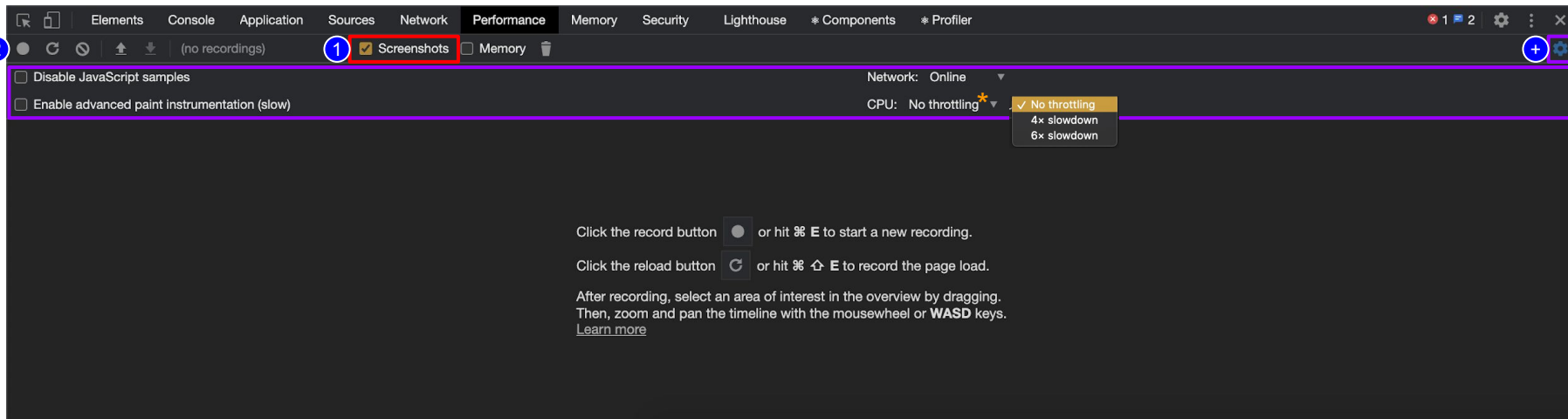
런타임 성능*: 로드할 때가 아닌, 페이지가 실행될 때의 성능을 의미함

유틸*: 쓰지 않고 놀림

Performance 패널을 사용해 성능 측정해보기

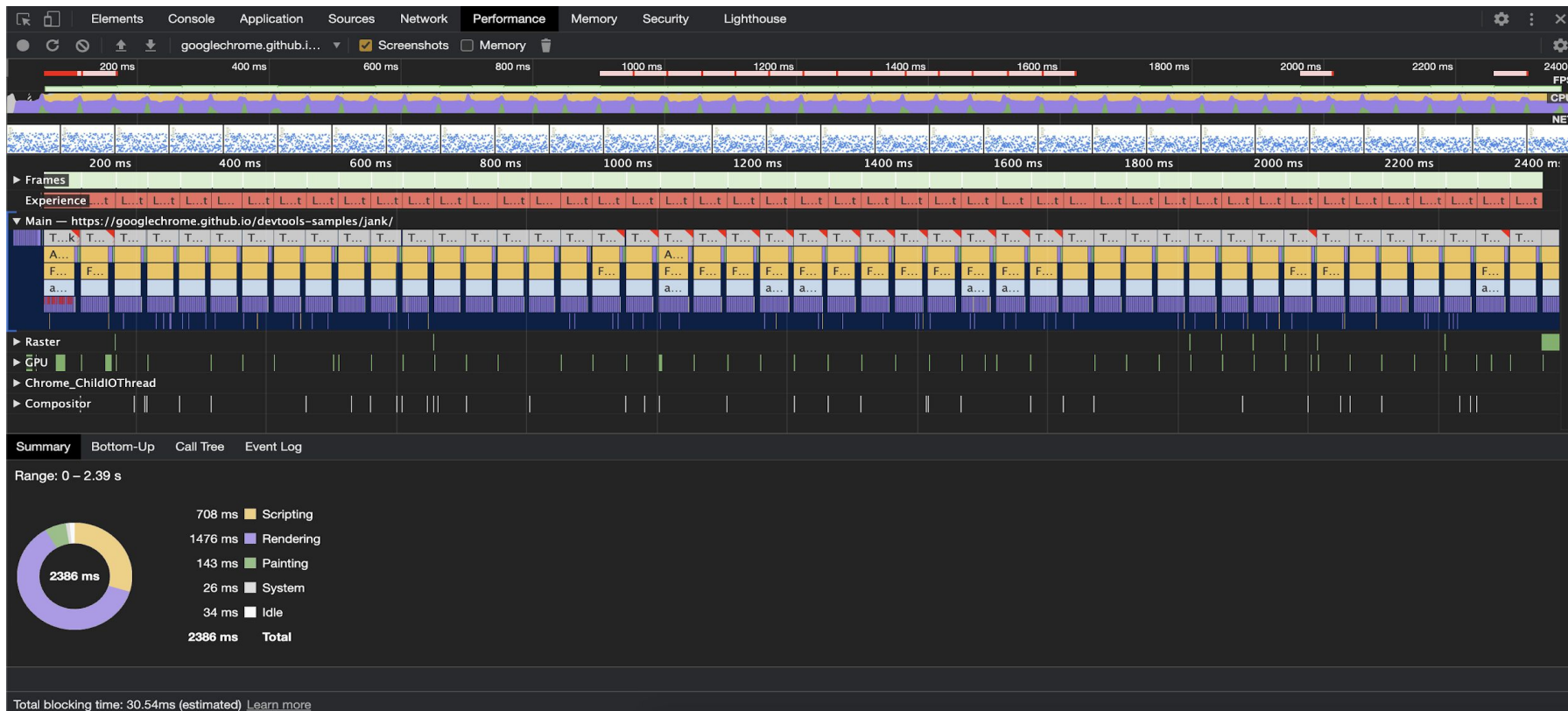
- 1) Screenshots 체크
- 2) Record 버튼 클릭

+ 추가 설정(Network, CPU 조절 등)을 원할 시 Capture Settings를 선택해 설정

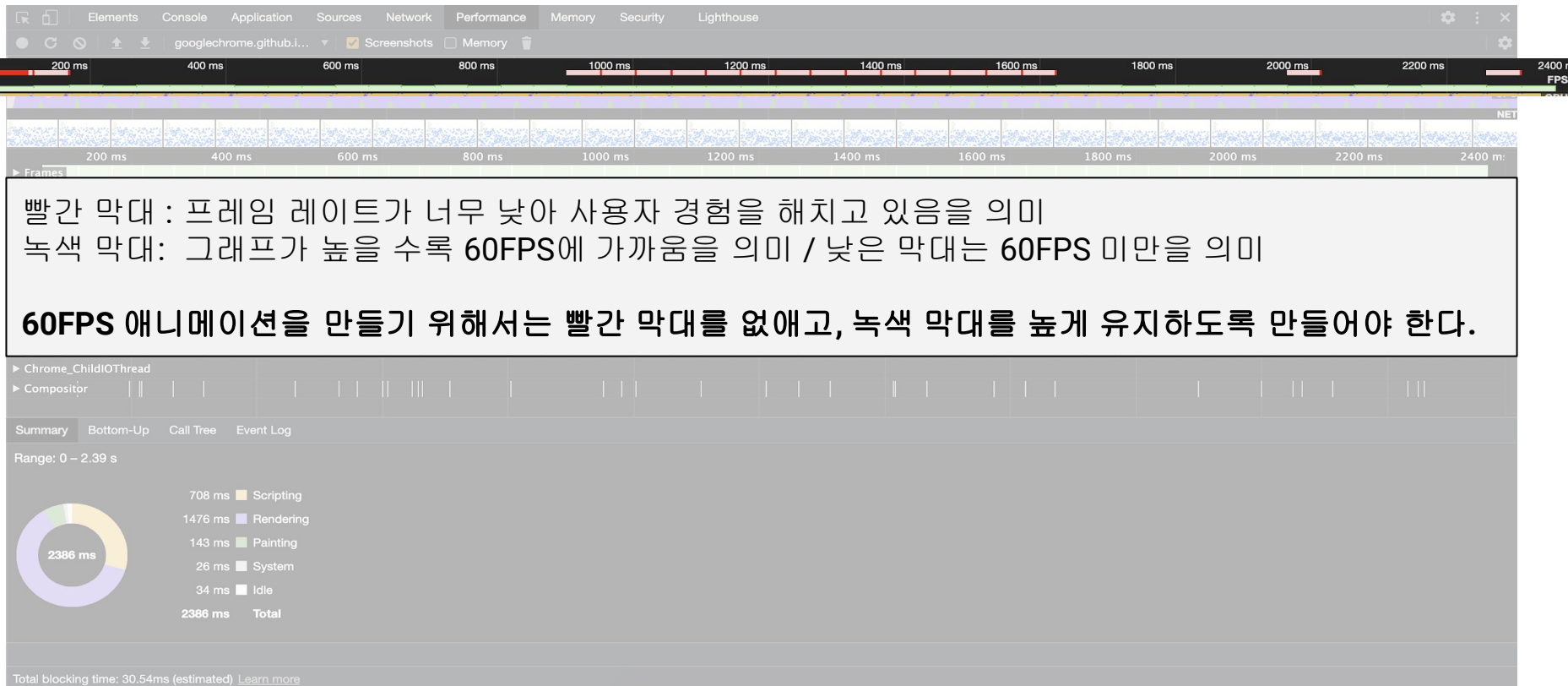


Throttling*: 기기의 CPU, GPU 등이 지나치게 과열될 때 기기의 손상을 막고자 클럭과 전압을 강제로 낮추거나 강제로 전원을 꺼서 발열을 줄이는 기능

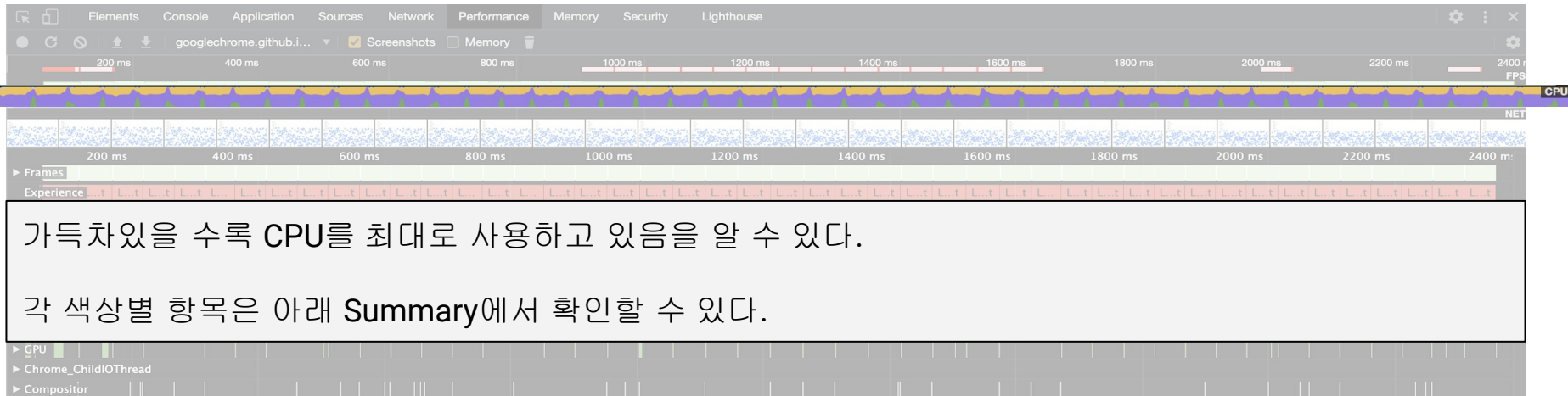
수집된 성능 지표 살펴보기



FPS 차트

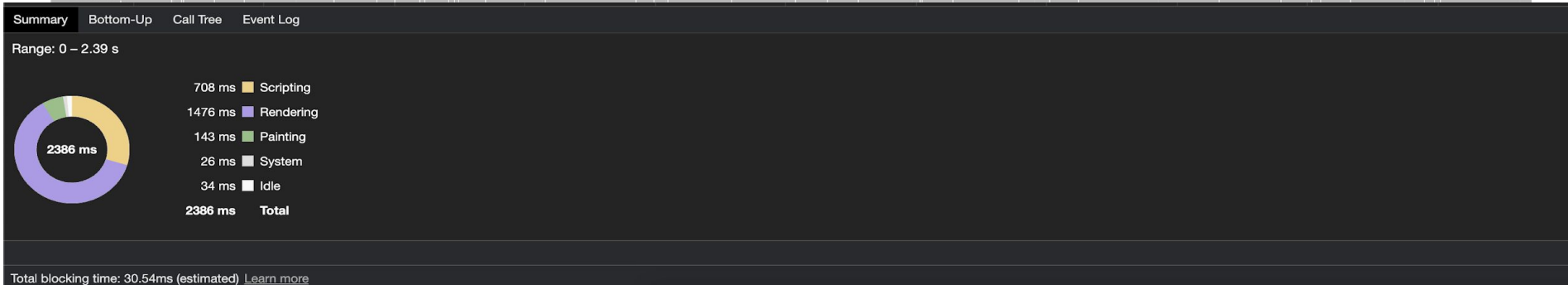


CPU 차트

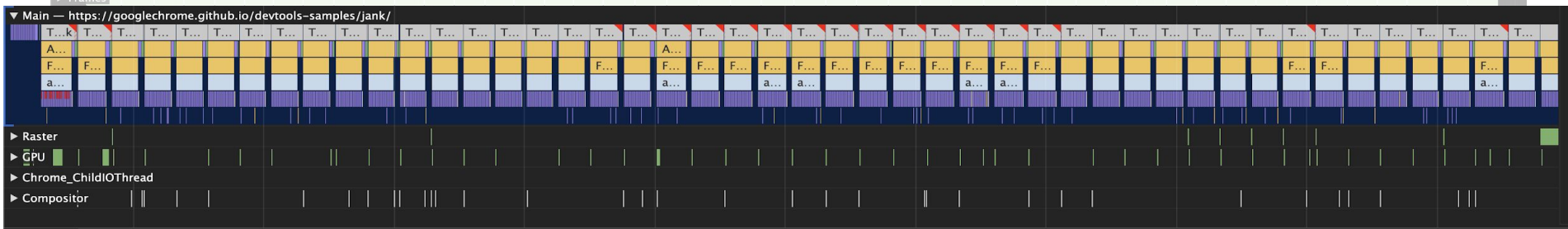
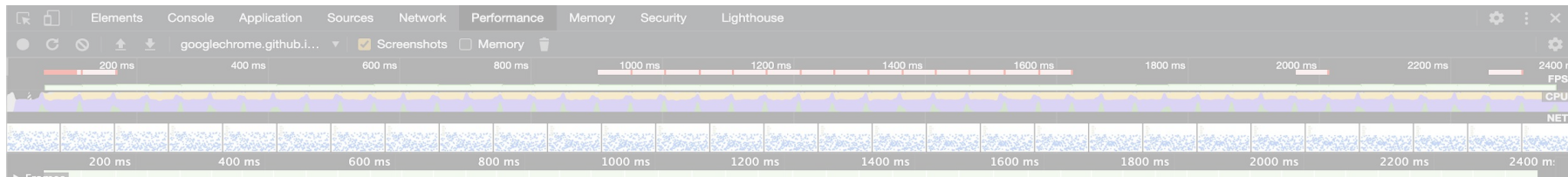


가득차있을 수록 **CPU**를 최대로 사용하고 있음을 알 수 있다.

각 색상별 항목은 아래 **Summary**에서 확인할 수 있다.



Main 섹션



x축은 시간에 따른 기록, y축은 콜 스택을 의미(위쪽의 이벤트가 아래쪽의 이벤트를 발생시킨 것)한다.

각각의 막대는 이벤트를 나타내며, 막대 내 빨간 삼각형은 이벤트에 오류가 있음을 나타낸다.



실습해보기!

실습링
크

CSS를 살펴보기 전 Timeline 학습



위와 같은 Timeline을 **Critical Rendering Path** 라고 부름

요소를 렌더링하고 재생하는 동안 브라우저는 무엇을 하는지 한 단계씩 알아보자.

1 단계 **Styles**



요소에 적용할 스타일 계산 시작

2단계 **Layout**



각 요소(레이아웃)에 대한 모양과 위치를 생성하기 시작

width 및 **height**, **margin** 또는 **left / top / right / bottom**과 같은 페이지 속성을 설정

3단계 **Paint**



각 요소의 픽셀을 레이어로 채우기 시작합니다.

사용되는 속성은 예를 들어 **box-shadow, border-radius, color, background-color** 등입니다.

4단계 Composite



화면의 모든 레이어를 그리기 시작

최신 브라우저는 **Position** 및 **Opacity** 속성을 사용하여 애니메이션 할 수 있다.

- **Position** – transform: translateX(n) translateY(n) translateZ(n);
- **Scale** – transform: scale(n);
- **Rotation** – transform: rotate(n deg);
- **Opacity** – opacity: n ;

compositing(합성)에만 영향을 주는 속성을 쓰는 것이 우리가 목표하는 것

이제 알아보시다!

HTML 작성

html

```
<div class="layout">  
  <div class="app-menu"></div>  
  <div class="header">  
    <div class="menu-icon"></div>  
  </div>  
</div>
```

CSS 작성

html

```
<div class="layout">
  <div class="app-menu"></div>
  <div class="header">
    <div class="menu-icon"></div>
  </div>
</div>
```

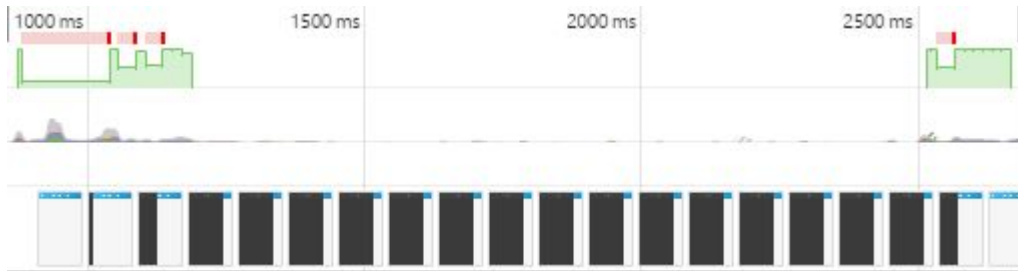
CSS

```
.app-menu {
  left: -300px;
  transition: left 300ms linear;
}

.app-menu-open .app-menu {
  left: 0px;
  transition: left 300ms linear;
}
```

지금까지 코드로 만약 애니메이션을 만들었다면?

https://www.youtube.com/watch?v=9v3333333333



1단계 CSS 작성

CSS

```
.app-menu {  
  left: -300px;  
  transition: left 300ms linear;  
}
```

```
.app-menu-open .app-menu {  
  left: 0px;  
  transition: left 300ms linear;  
}
```



단순하게 이렇게 생각하기 쉽지만,

transitions 속성과 left/ top/ right/ bottom 속성을 함께 사용하면 안된다.

왜냐하면,

브라우저가 매번 레이아웃을 생성해
모든 자식에게 영향을 미쳐
유동적인 애니메이션을 만들지 못한다.

(Critical Rendering Path의 2단계에 해당)

그럼 transition과 transform을 사용해보자

html

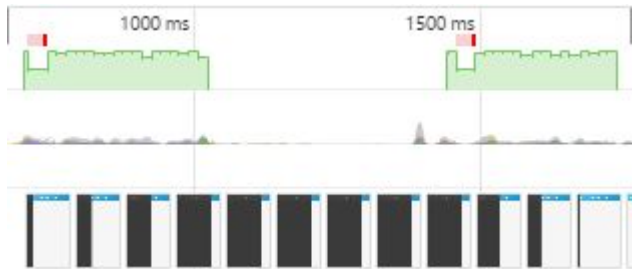
```
<div class="layout">
  <div class="app-menu"></div>
  <div class="header">
    <div class="menu-icon"></div>
  </div>
</div>
```

CSS

```
.app-menu {
  -webkit-transform: translateX(-100%);
  transform: translateX(-100%);
  transition: transform 300ms linear;
}

.app-menu-open .app-menu {
  -webkit-transform: none;
  transform: none;
  transition: transform 300ms linear;
}
```


지금까지 코드로 만약 애니메이션을 만들었다면?



2단계 CSS 작성

CSS

```
.app-menu {  
  -webkit-transform: translateX(-100%);  
  transform: translateX(-100%);  
  transition: transform 300ms linear;  
}  
  
.app-menu-open .app-menu {  
  -webkit-transform: none;  
  transform: none;  
  transition: transform 300ms linear;  
}
```



transform 속성은 composite에 영향을 미친다.

여기에서는 애니메이션이 시작되는 즉시 레이어가 그려지고 이동할 준비가 되므로 애니메이션을 렌더링 할 때 문제가 적다.

한 단계 더 나아가 **GPU***에서 애니메이션을 실행해보자!

CPU vs GPU	
CPU	GPU
Central Processing Unit	Graphics Processing Unit
Several cores	Many cores
Low latency	High throughput
Good for serial processing	Good for parallel processing
Can do a handful of operations at once	Can do thousands of operations at once

CPU와 GPU의 비교 / 출처 : [링크](#)

왜 **GPU**를 사용해야 하나요?

GPU는 복잡한 문제를 수천, 수백만 개의 개별 작업으로 나누고 한 번에 해결한다.

따라서, 이미지가 화면을 가로 질러 날아가도록 텍스처, 조명 및 모양 렌더링을 한번에 수행해야하는 그래픽 작업에 이상적이기 때문이다.

어떻게 GPU에서 애니메이션을 실행?

CSS

```
.app-menu {  
  -webkit-transform: translateX(-100%);  
  transform: translateX(-100%);  
  transition: transform 300ms linear;  
  will-change: transform;  
}  
  
.app-menu-open .app-menu {  
  -webkit-transform: none;  
  transform: none;  
  transition: transform 300ms linear;  
}
```

생소한 will-change 속성 알아보기!

will-change 속성은 요소가 어떻게 변경될 것으로 예상되는지 브라우저에 힌트를 주는 것이다. 그럼 브라우저는 요소가 실제로 변경되기 전에 최적화를 설정할 수 있다.

또한 css가 아닌 자바스크립트로 추가하는 것이 이상적이다. (stylesheet에 선언된 will-change는 제거할 수 없기 때문이다.)

단, 사용자가 계속해서 상호 작용할 가능성이 있고 사용자의 상호작용에 신속하게 반응해야하는 소수의 요소에 will-change를 설정하는 경우에는 css에 추가해도 된다. (제한된 수의 요소는 브라우저에서 수행한 최적화가 과도하게 사용되지 않으므로 그다지 피해를 주지 않기 때문이다.)

(단, will-change는 will이므로 애니메이션이 발생할 때 적용해서는 안됨)

하지만, will-change 속성을 mdn에서는 기존 성능 문제를 해결하기 위해 최후의 수단으로 사용하라고 말하고 있다. (남용 시 오히려 성능 문제 야기)

추가학습) will-change와 이전 속성과의 비교

will-change 속성 이전에는 **translateZ()** 또는 **translate3d()** 속성을 사용해 GPU에서 실행하도록 했다.

위 속성을 **css**에 추가하면 브라우저가 이 속성을 사용해 요소를 새 **composited layer**로 이동하도록 해 GPU가 사용되어 성능이 향상되는 **css trick**을 사용한 것이다.

그러나 새 레이어에 요소를 설정하는 것은 비교적 비용이 많이 드는 작업이고 변환 애니메이션의 시작이 눈에 띄게 깜빡이는 지연이 발생하는 오류가 있었다.

따라서 위와 같은 지연오류를 막고자 나온 것이 **will-change** 속성이다.

will-change는 해킹에 의존하지 않고 브라우저를 필요하거나 유용하지 않을 수도 있는 레이어 생성을 강제하지 않고도 속도를 높일 수 있는 더 나은 방법이다.

하지만 **will-change**는 현명하게 사용하지 않으면 오히려 실제로 페이지를 충돌시킬 수 있는 성능 저하가 발생한다. 또한 직접적으로 감지할 수 없는 부작용이 있기 때문에 아래의 몇 가지 사항들을 사용할 때 염두해두고 사용해야한다.

- 1) 너무 많은 속성 또는 요소에 대한 변경 사항을 선언하기 위해 **will-change**를 사용하지 않기
- 2) 브라우저에 충분한 작업 시간 제공하기
- 3) 변경이 완료된 후 **will-change** 제거하기

주의! 마지막으로 앞서 언급했듯이 **will-change**의 일부 변경으로 인해 새로운 **composited layer**가 생성된다.

그러나 GPU는 대부분의 브라우저에서 CPU가 수행하는 하위 픽셀 앤티 앨리어싱을 지원하지 않으며, 이로 인해 콘텐츠(특히 텍스트)가 흐릿해질 수 있다.

지금까지 코드로 만약 애니메이션을 만들었다면?



현재의 문제 찾기

html

```
<div class="layout">
  <div class="app-menu"></div>
  <div class="header">
    <div class="menu-icon"></div>
  </div>
</div>
```

js

```
function toggleClassMenu() {
  var layout = document.querySelector(".layout");
  if(!layout.classList.contains("app-menu-open")) {
    layout.classList.add("app-menu-open");
  } else {
    layout.classList.remove("app-menu-open");
  }
}

var oppMenu = document.querySelector(".menu-icon");
oppMenu.addEventListener("click", toggleClassMenu,
false);
```

HTML 구조 다시 손보기

html

```
<div class="menu">
  <div class="app-menu"></div>
</div>
<div class="layout">
  <div class="header">
    <div class="menu-icon"></div>
  </div>
</div>
```

위 코드의 문제는

레이아웃 **div** 내에 해당 클래스를 추가하여

브라우저가 스타일을 한 번 더 재계산하도록 해
렌더링 성능에 영향을 미친다는 것이다.

그렇다면, 영향을 미치지 않기 위해

menu를 격리된 영역으로 **html** 구조를 왼쪽과 같이
수정해봅시다.

JS도 다시 손보기

JS

```
function toggleClassMenu() {  
    myMenu.classList.add("menu--animatable");  
  
    if(!myMenu.classList.contains("menu--visible")) {  
        myMenu.classList.add("menu--visible");  
    } else {  
        myMenu.classList.remove('menu--visible');  
    }  
}  
  
function OnTransitionEnd() {  
    myMenu.classList.remove("menu--animatable");  
}  
  
var myMenu = document.querySelector(".menu");  
var oppMenu = document.querySelector(".menu-icon");  
myMenu.addEventListener("transitionend", OnTransitionEnd, false);  
oppMenu.addEventListener("click", toggleClassMenu, false);  
myMenu.addEventListener("click", toggleClassMenu, false);
```

JavaScript의 **transitionend** 함수를 사용하여
전환 시간이 끝나면 제거되는 클래스의 애니메이션을
조작해보자.

추가설명

transitionend event는 css 전환이 완료되면 transitionend 이벤트가
시작된다.

단, transition이 완료전에 제거되거나, transtion-property가 제거되거나
display:none이면 실행되지 않는다.

transitionend 이벤트는 전환 상태와 전환되지 않은 상태로 완전히
되돌아 갈 때 양방향으로 발생한다.

정말 마지막으로 이 코드를 실행한다면!

html

```
<div class="menu">
  <div
class="app-menu"></div>
</div>
<div class="layout">
  <div class="header">
    <div
class="menu-icon"></div>
  </div>
</div>
```

CSS

한 눈에 보이지 않음

```
.menu--visible {
  pointer-events: auto;
}

.app-menu {
  background-color: #fff;
  color: #fff;
  position: relative;
  max-width: 400px;
  width: 90%;
  height: 100%;
  box-shadow: 0 2px 6px rgba(0, 0, 0, 0.5);
  -webkit-transform: translateX(-103%);
  transform: translateX(-103%);
  display: flex;
  flex-direction: column;
  will-change: transform;
  z-index: 160;
  pointer-events: auto;
}

.menu--visible .app-menu {
  -webkit-transform: none;
  transform: none;
}

.menu--animatable .app-menu {
  transition: all 130ms ease-in;
```

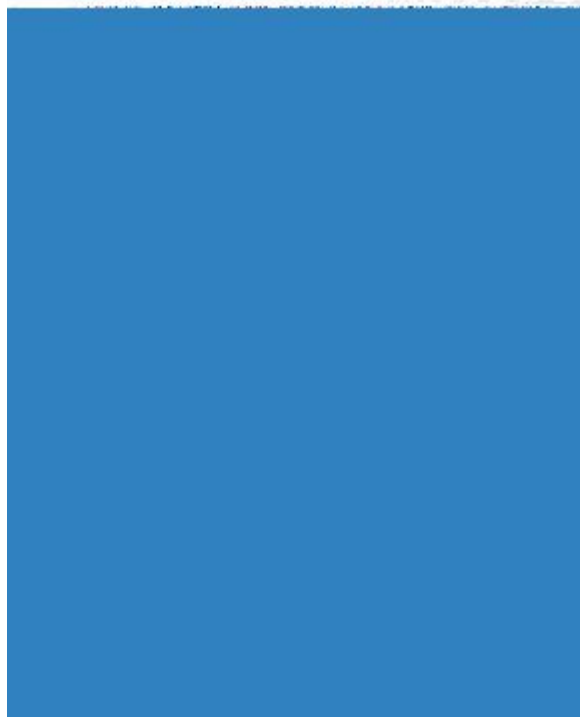
js

```
function toggleClassMenu() {
  myMenu.classList.add("menu--animatable");
  if (!myMenu.classList.contains("menu--visible")) {
    myMenu.classList.add("menu--visible");
  } else {
    myMenu.classList.remove('menu--visible'
);
  }
}

function OnTransitionEnd() {
  myMenu.classList.remove("menu--animatable");
}

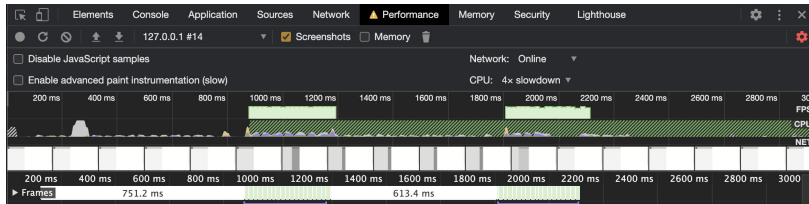
var myMenu = document.querySelector(".menu");
var oppMenu = document.querySelector(".menu-icon");
myMenu.addEventListener("transitionend", OnTransitionEnd,
false);
oppMenu.addEventListener("click", toggleClassMenu, false);
myMenu.addEventListener("click", toggleClassMenu, false);
```

완성!

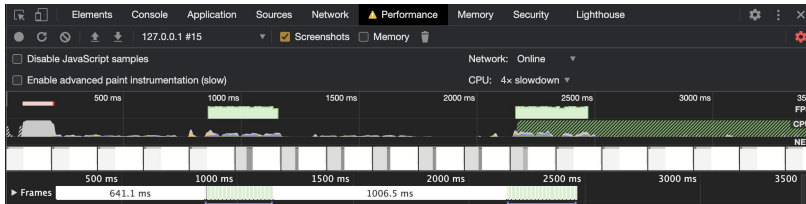


END

Q. 왜 transitionend를 써서 class를 없애야 하나요?



willchange를 사용한 경우 performance



willchange를 사용하지 않은 경우
performance

A. 우리가 **will-change**를 사용하는 이유는 **하드웨어 가속***을 사용하기 위해서이다. CSS 작업이 하드웨어 가속일 때 일반적으로 페이지 렌더링이 빨라져 속도가 향상되기 때문이다.

(GPU는 그래픽 렌더링에 필요한 복잡한 수학적 및 기하학적 계산을 수행하기 위해 특별히 설계되었다. 그러나 CSS 애니메이션, **transforms** 및 **transitions**은 자동으로 GPU 가속화되지 않고 대신 브라우저의 느린 소프트웨어 렌더링 엔진에서 실행된다. 그러나 일부 브라우저는 더 나은 렌더링 성능을 제공하기 위해 특정 속성을 통해 하드웨어 가속을 제공한다.)

레이어 생성 기술은 페이지 속도를 높일 수 있지만 비용이 발생한다. 시스템 RAM과 GPU에서 메모리를 차지하고 (특히 모바일 장치에서 제한됨) 많은 메모리를 사용하면 (특히 모바일 장치에서) 나쁜 영향을 미칠 수 있다. 따라서 **will-change**는 현명하게 사용되어야 하며 하드웨어 가속 작업이 페이지의 성능에 실제로 도움이 되는지 확인하고 페이지의 다른 작업으로 인해 성능 병목 현상이 발생하지 않는지 확인해야 한다.

브라우저가 곧 발생할 변경 사항에 대해 수행하는 최적화는 일반적으로 비용이 많이 들고 앞서 언급했듯이 시스템 리소스를 많이 차지할 수 있다. 최적화를 위한 일반적인 브라우저 동작은 이러한 최적화를 제거하고 가능한 한 빨리 정상 동작으로 되돌리는 것이다. 그러나 **will-change**는 브라우저가 그렇지 않은 경우보다 훨씬 오랫동안 최적화를 유지하는 이 동작을 재정의한다. 요소가 변경된 후 **will-change**를 제거해야 한다. 그러면 브라우저가 최적화가 요구하는 모든 리소스를 복구할 수 있다.

하드웨어 가속*: 모든 것을 CPU에 투입하는 대신 GPU가 몇가지 무거운 작업을 수행해 페이지를 렌더링하는데 브라우저 지원한다는 것을 의미 / GPU 가속이라고도 부름

참고 자료

- <https://developers.google.com/web/tools/chrome-devtools/evaluate-performance>
- <https://medium.com/outsystems-experts/how-to-achieve-60-fps-animations-with-css3-db7b98610108>
- https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement/transitionend_event
- <https://developer.mozilla.org/en-US/docs/Web/CSS/will-change>
- <https://stackoverflow.com/questions/26907265/how-to-use-and-how-works-css-will-change-property>
- <https://www.digitalocean.com/community/tutorials/css-will-change>
- <https://dev.opera.com/articles/css-will-change-property/>
- <https://blogs.nvidia.com/blog/2009/12/16/whats-the-difference-between-a-cpu-and-a-gpu/>