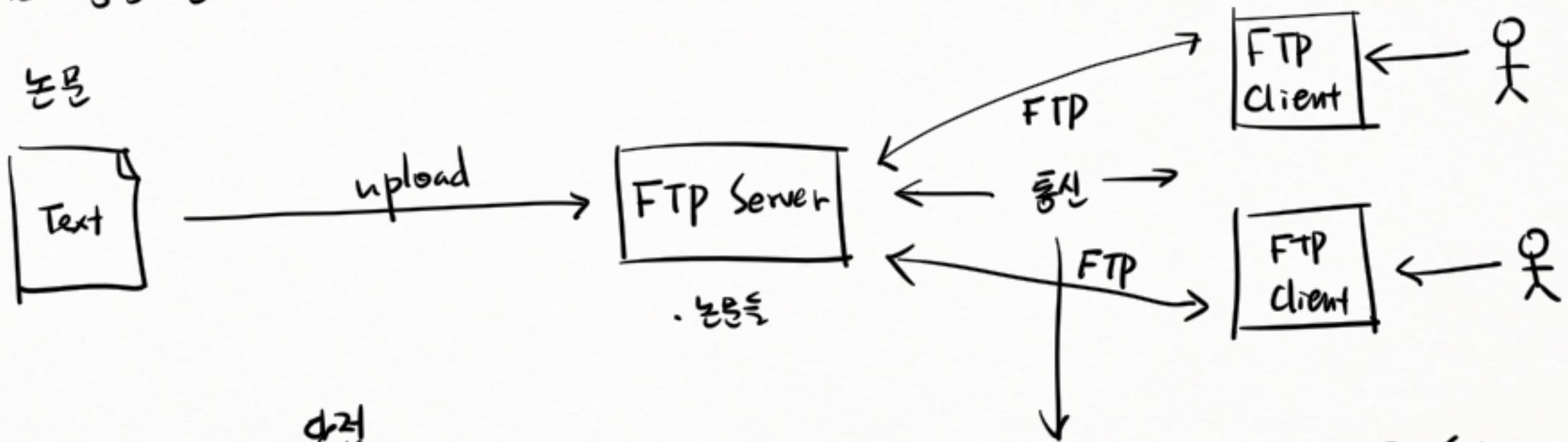


## \* HTML, HTTP

### ① HTTP, HTML 등장 전



단점

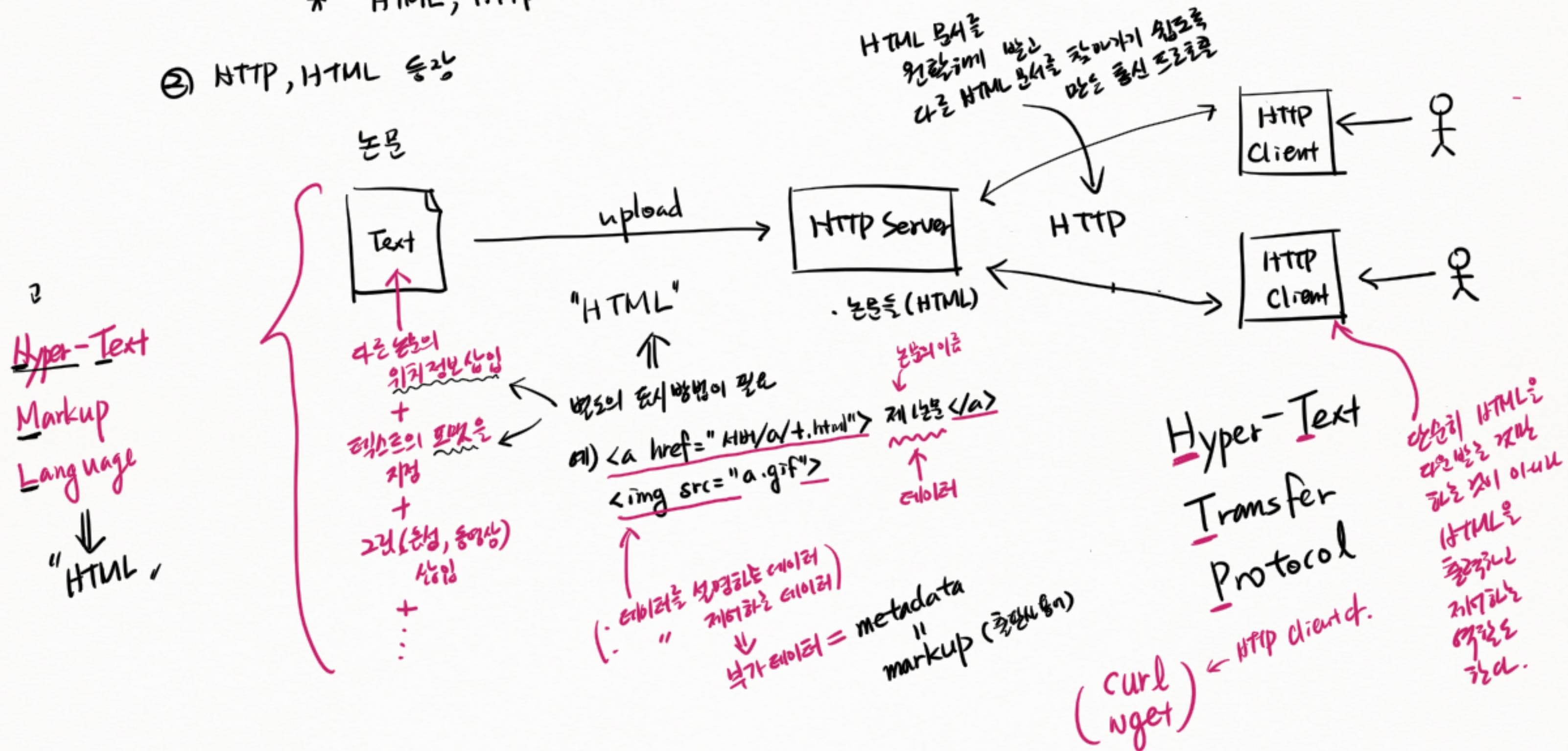
- 논문이 첨부하는 다른 논문을  
다운로드하기 번거롭다
- 논문 안에 다른 논문이 업로드된  
서버 주소가 많다.

통신 규칙이 아니라 Data를 송수신  
"protocol"

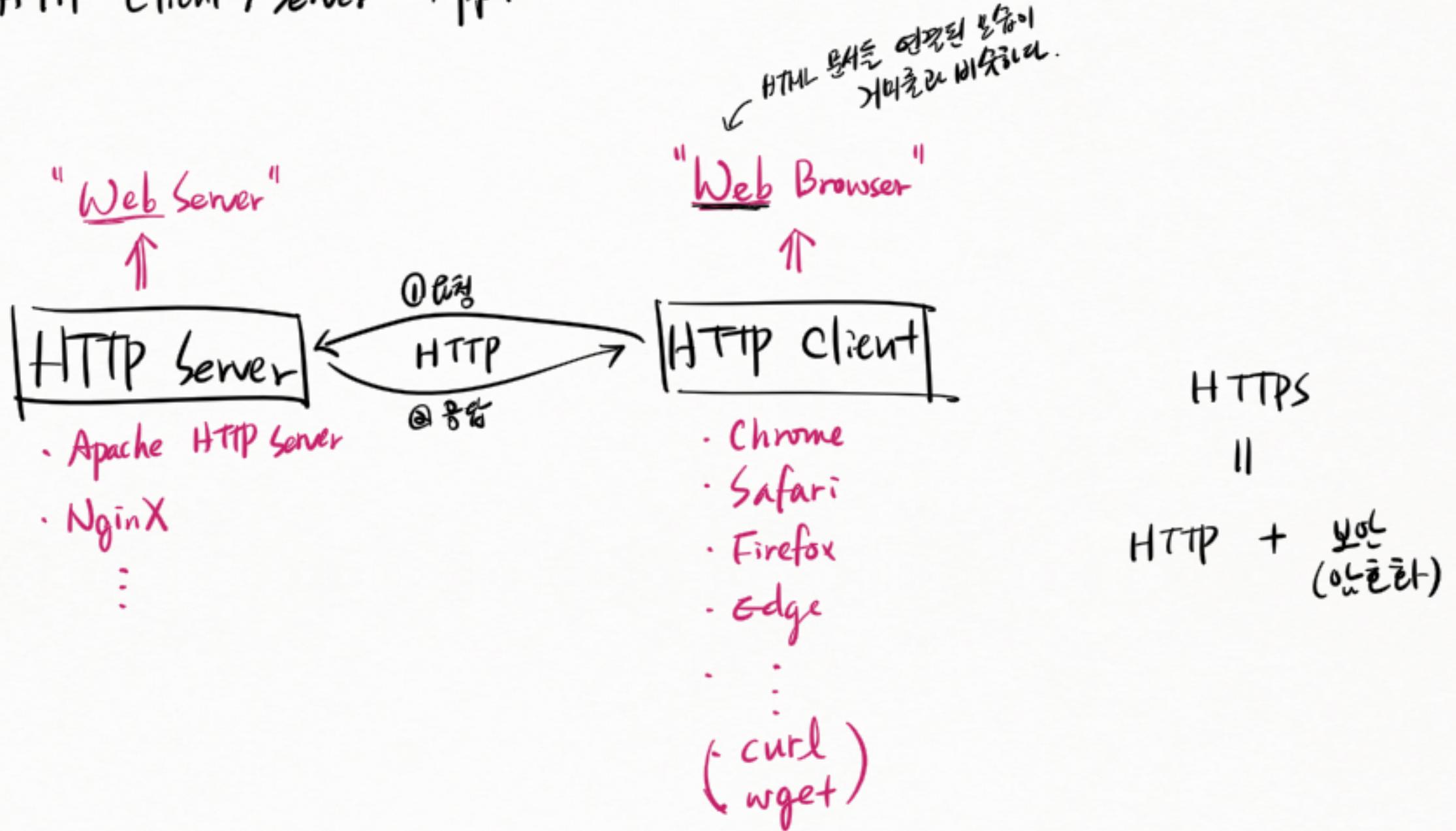
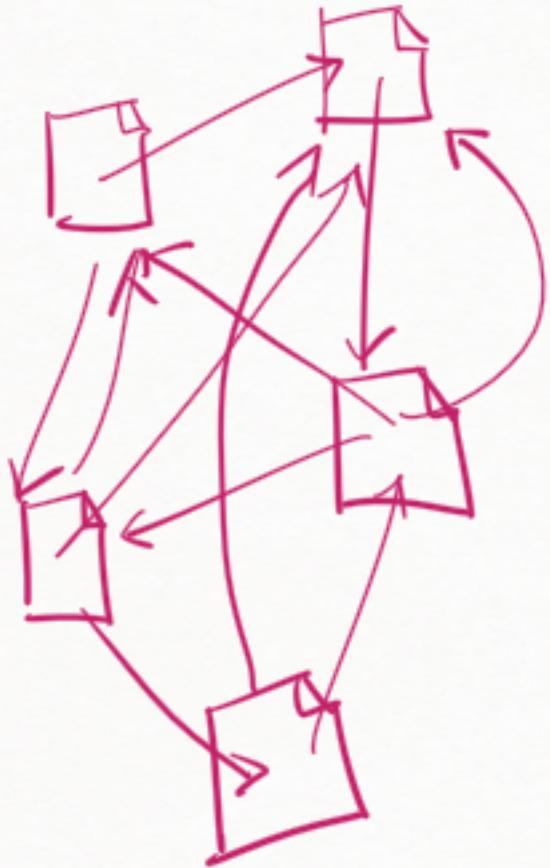
File Transfer Protocol

## \* HTML, HTTP

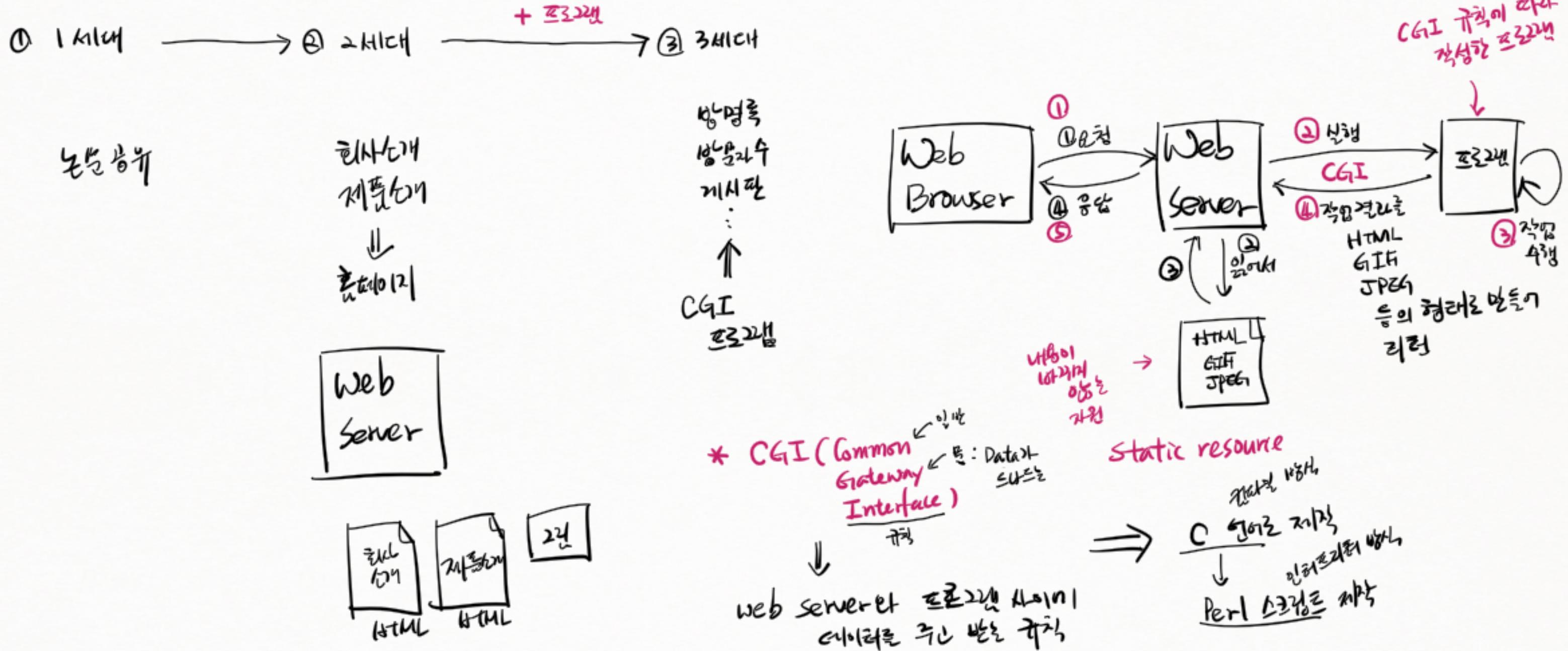
② HTTP, HTML  $\frac{6}{8}^{2/2}$



## \* HTTP Client / Server App.



## \* Web 기술의 활용

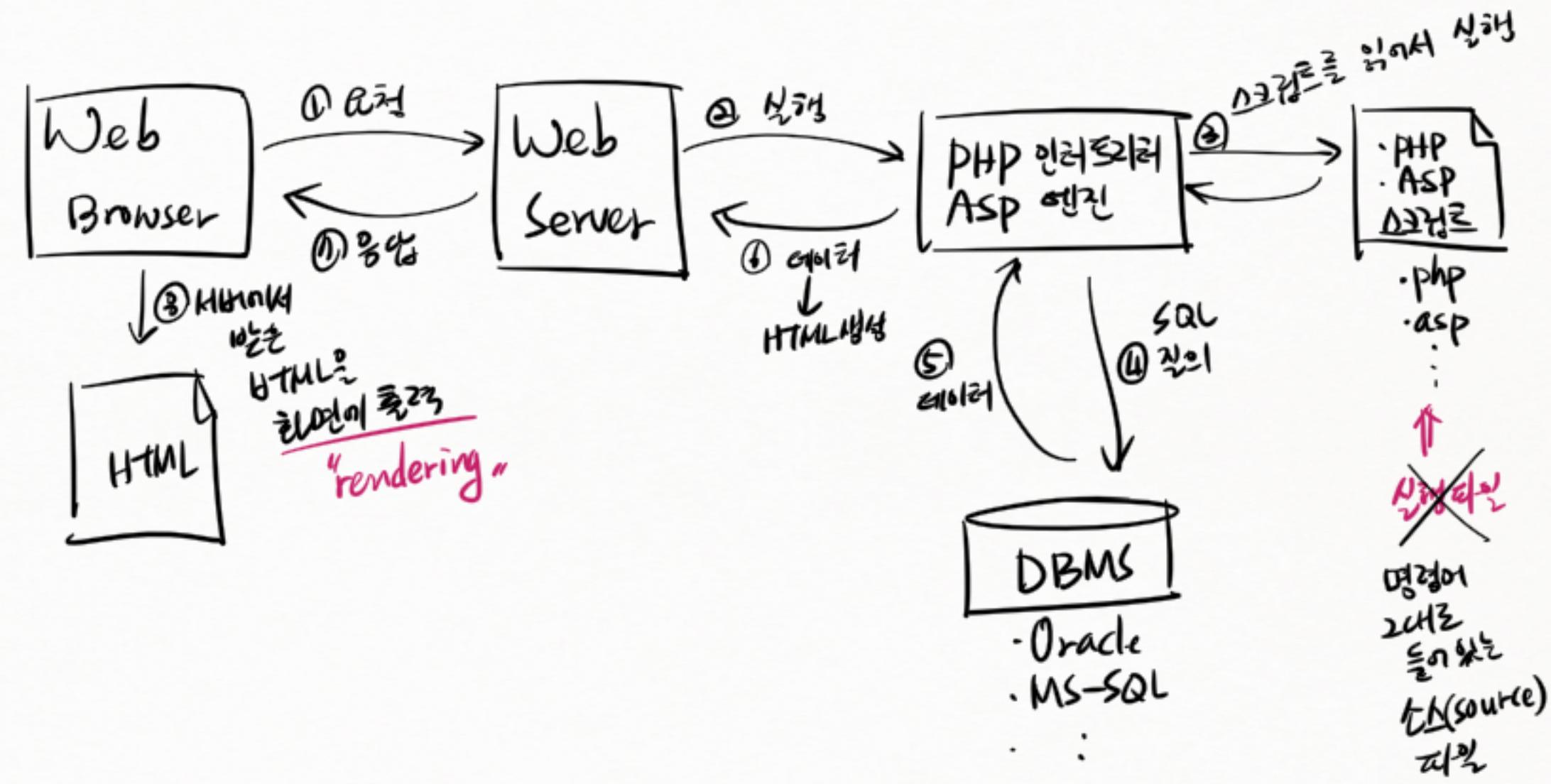


## \* Web 기술의 활용 II

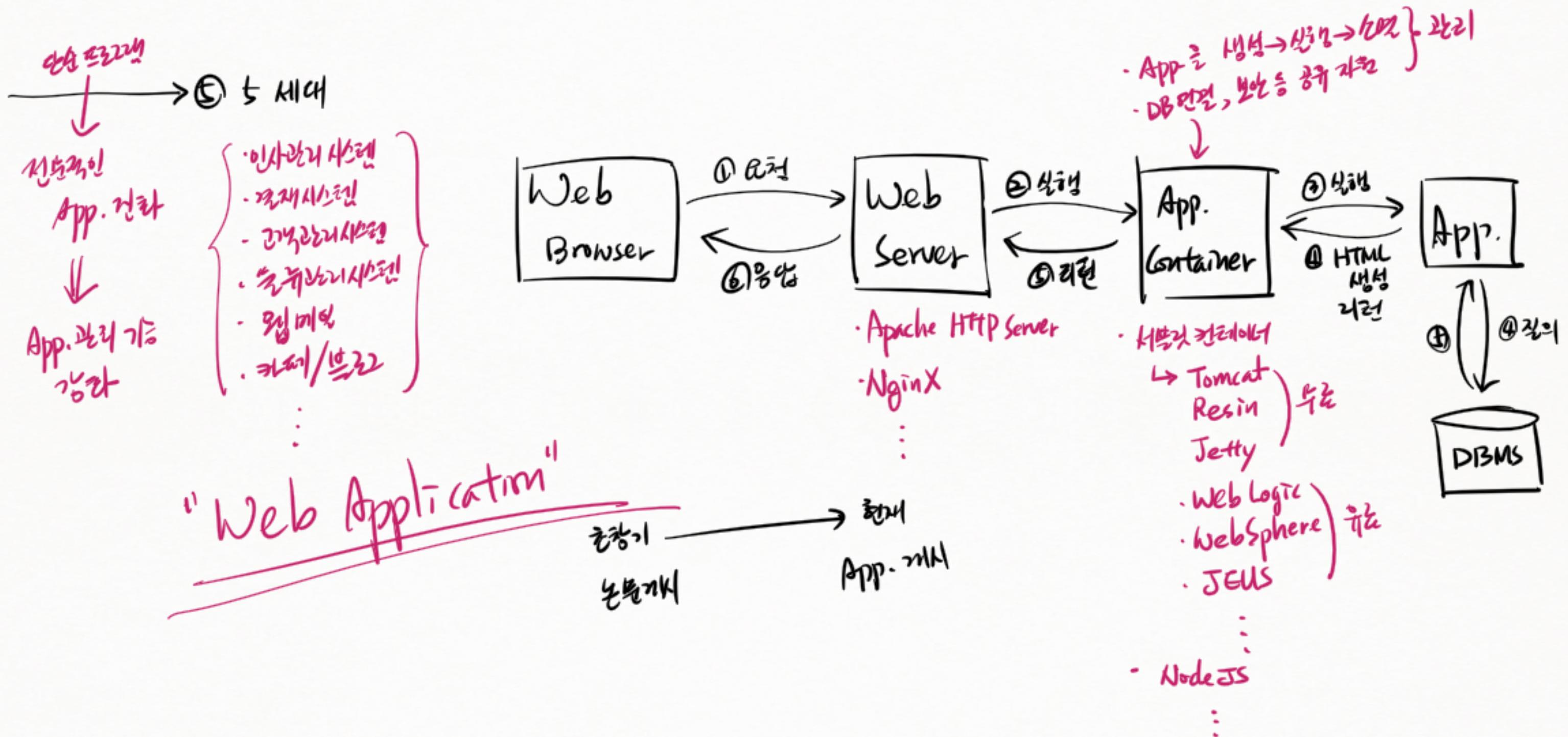
+ DBMS  
+ 스크립트  
→ ④ 웹서버

수정물  
+  
MS ASP 프로그래밍  
PHP  
:  
+  
DBMS

(01) 웹프로그래밍



## \* Web 기술의 활용 III



## \* Web Application 2가지 기술

[전면 만드는 기술]

CSS — 웹컨트의 디자인을  
제작하는 기술

+

HTML ← 웹컨트를 구성하는  
내용하는 기술

+

JavaScript ←  
• 사용자의 입력에 응답  
• 웹컨트를 제어

:

“Front-end 2가지”

← Full Stack  
2가지

SI / SM

[ 서버를 만드는 기술]

✓ Java, JavaScript, PHP, ...

✓ SQL

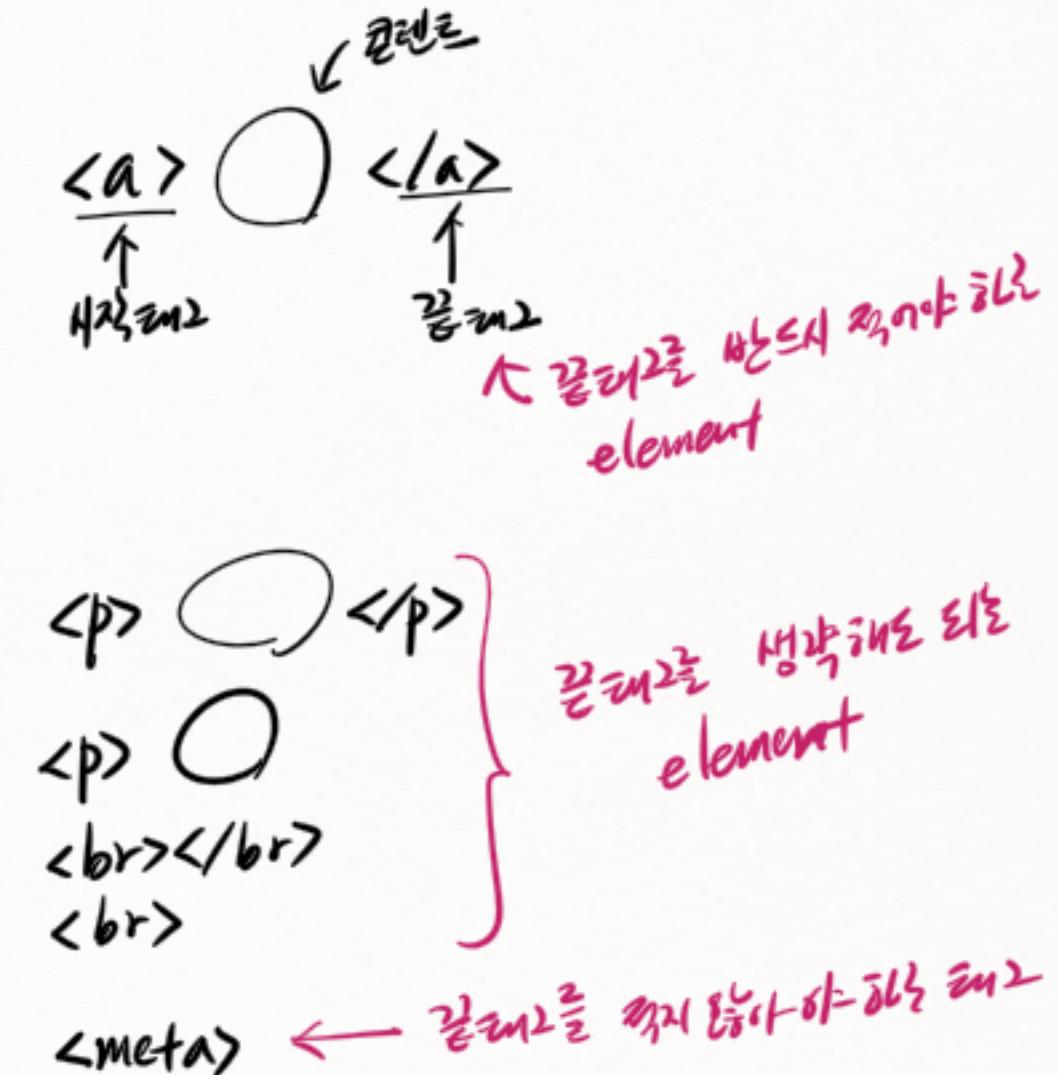
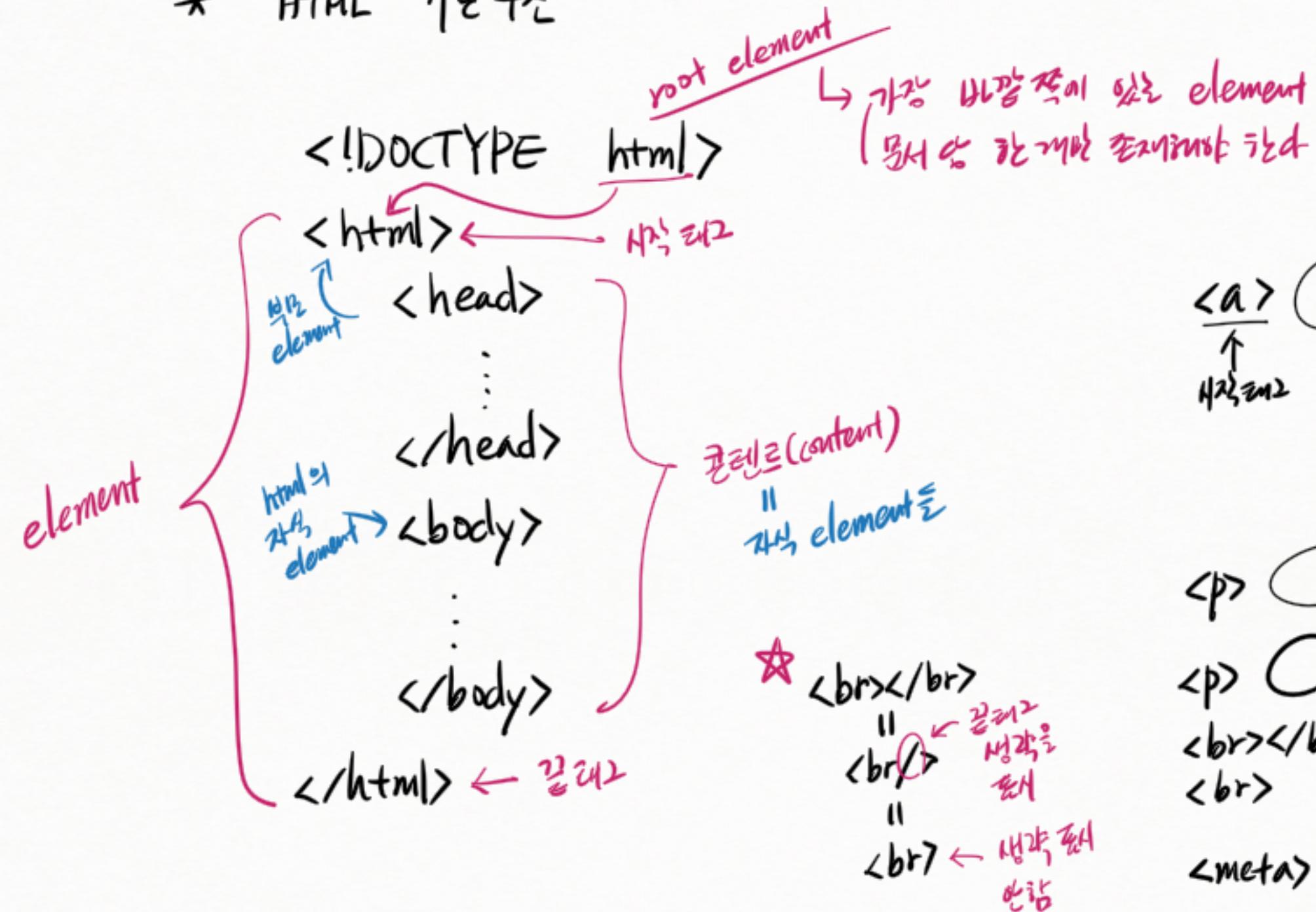
:

“Back-end 2가지”



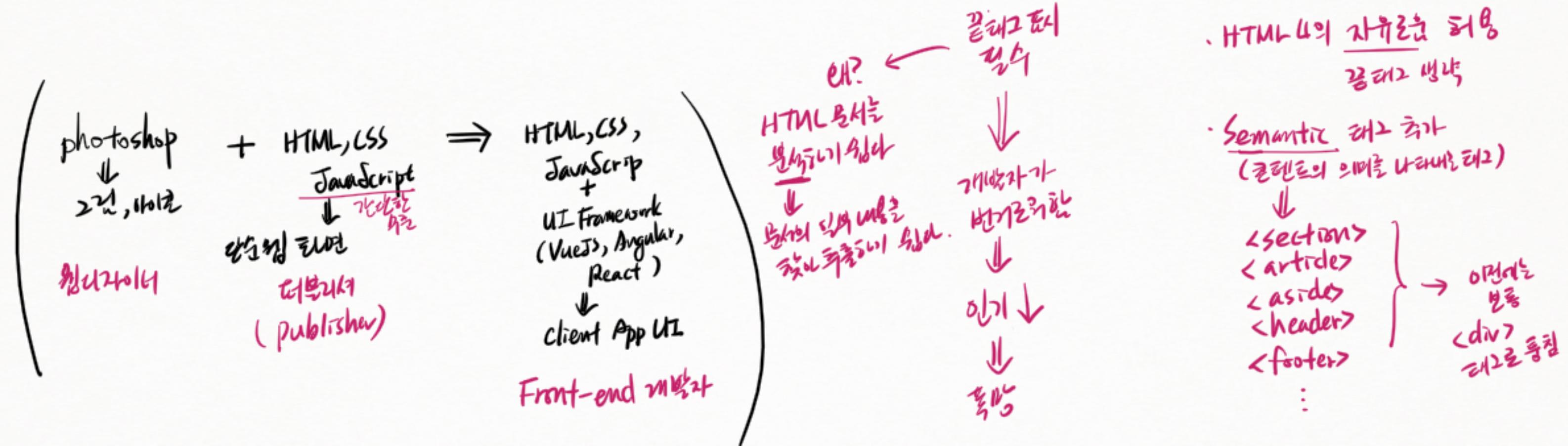
HTML

## \* HTML 기본 구조

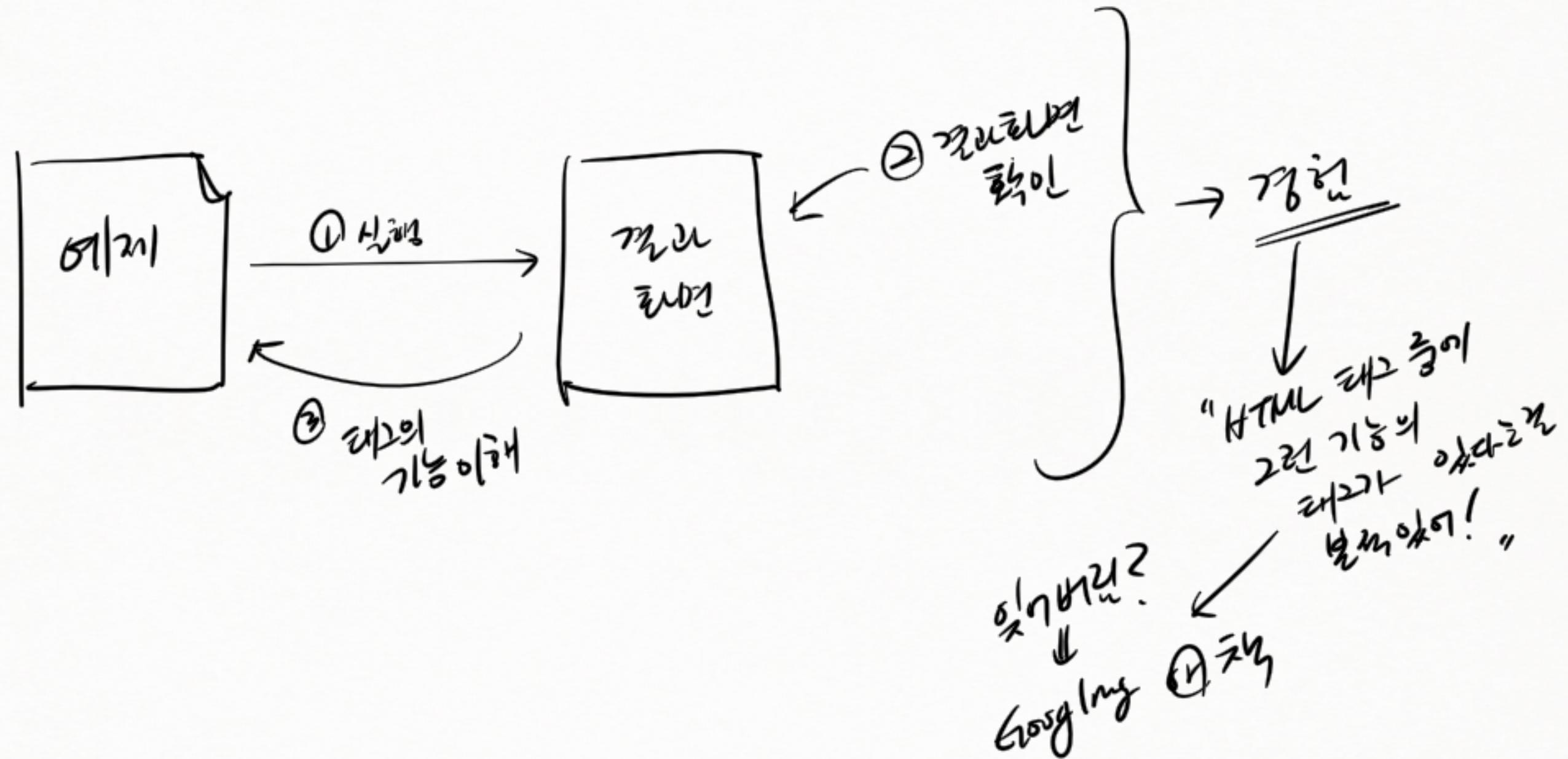


## \* HTML

HTML 3.x → HTML 4.x → XHTML → HTML5



## \* HTML 풍부함



## \* HTML4 의 DOCTYPE

HTML의 DOCTYPE

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://\_\_\_\_\_">

*문서 구조의 시작자*

*문서 구조의 공개언어*

*문서 구조의 규칙을 만드는 언어의 이름*

*+ 흥인 - HTML*

*문서 구조의 이름*

*문서 구조의 언어*

*문서 구조의 언어*

*문서 구조의 언어*

*문서 구조를 정의하는 DTD 파일의 URL*

*Document Type Definition*

I18N Internationalization

L10N Localization

태그를 사용하는 방법

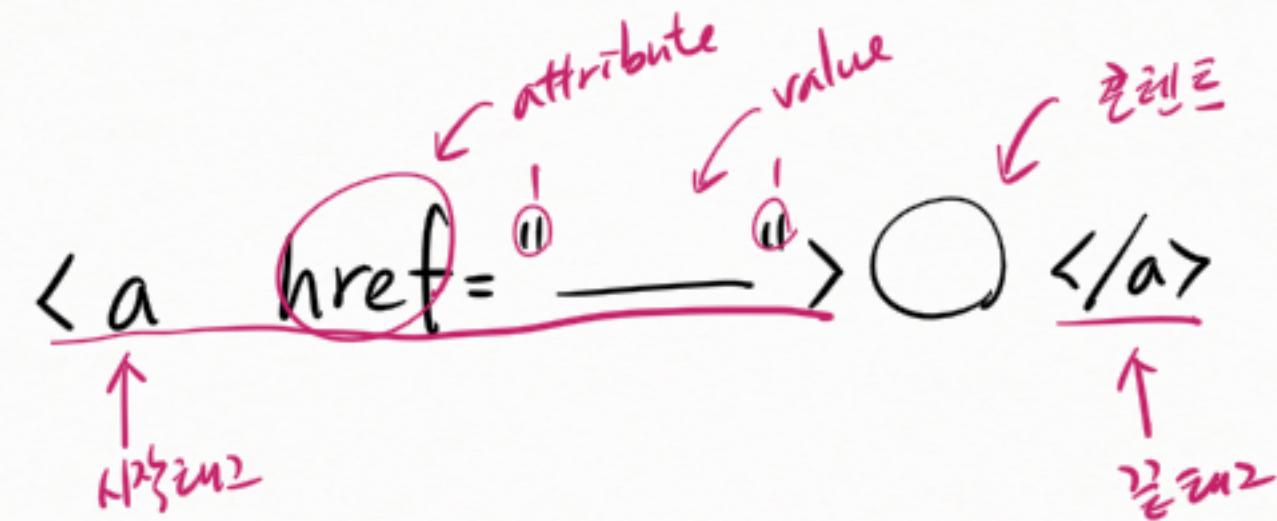
- 이전 태그가 필요해
- 이제는 HTML만 가능해
- 제한된 HTML만 가능해

\* DTD ← old

↓

Schema ← new

## \* tag et attribute



controls  
selected  
checked  
readonly  
:  
}      키보드에  
       마우스에  
       터치 패드에  
       현재 위치로  
       기능을  
       제공하는  
       특수한 속성

## \* CSS

① <style> 태그

```
<style>
  =
</style>
```

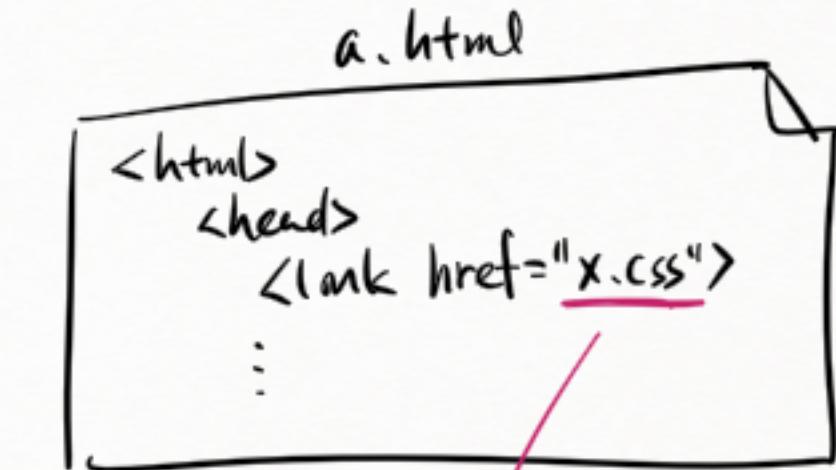
② inline 디자인

```
<tfoot style="—————" > 0 </tfoot>
```

inline style

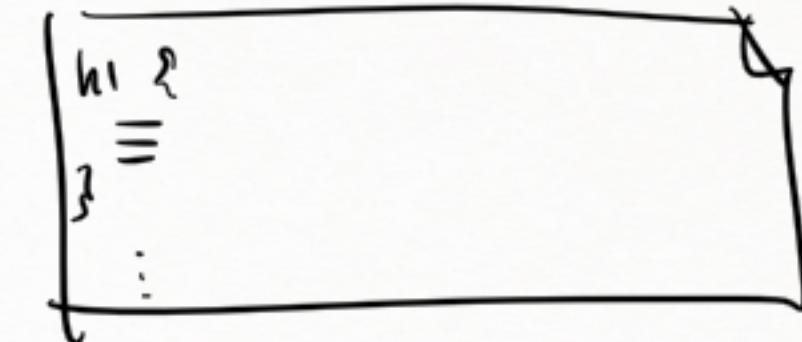
이것!

③ 외부 css 적용



외부 HTML 파일에서  
css를どのように 사용하는가?

x.css



## \* MIME Type

Multi-purpose  
Internet  
Mail  
Extensions

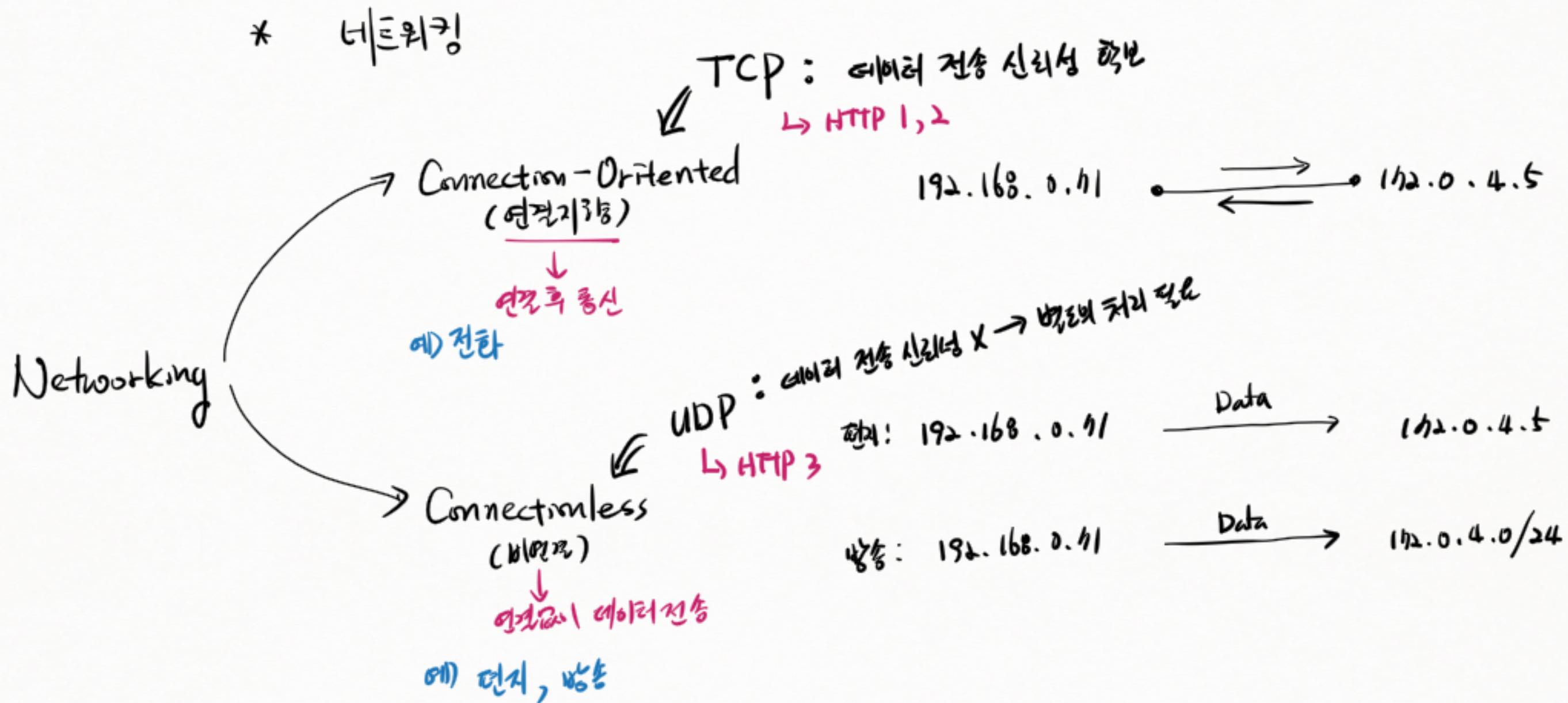
}  
메일이 전부는 초 텐트가 어떤 형식인지  
인터넷이나 메일과 같은 형식



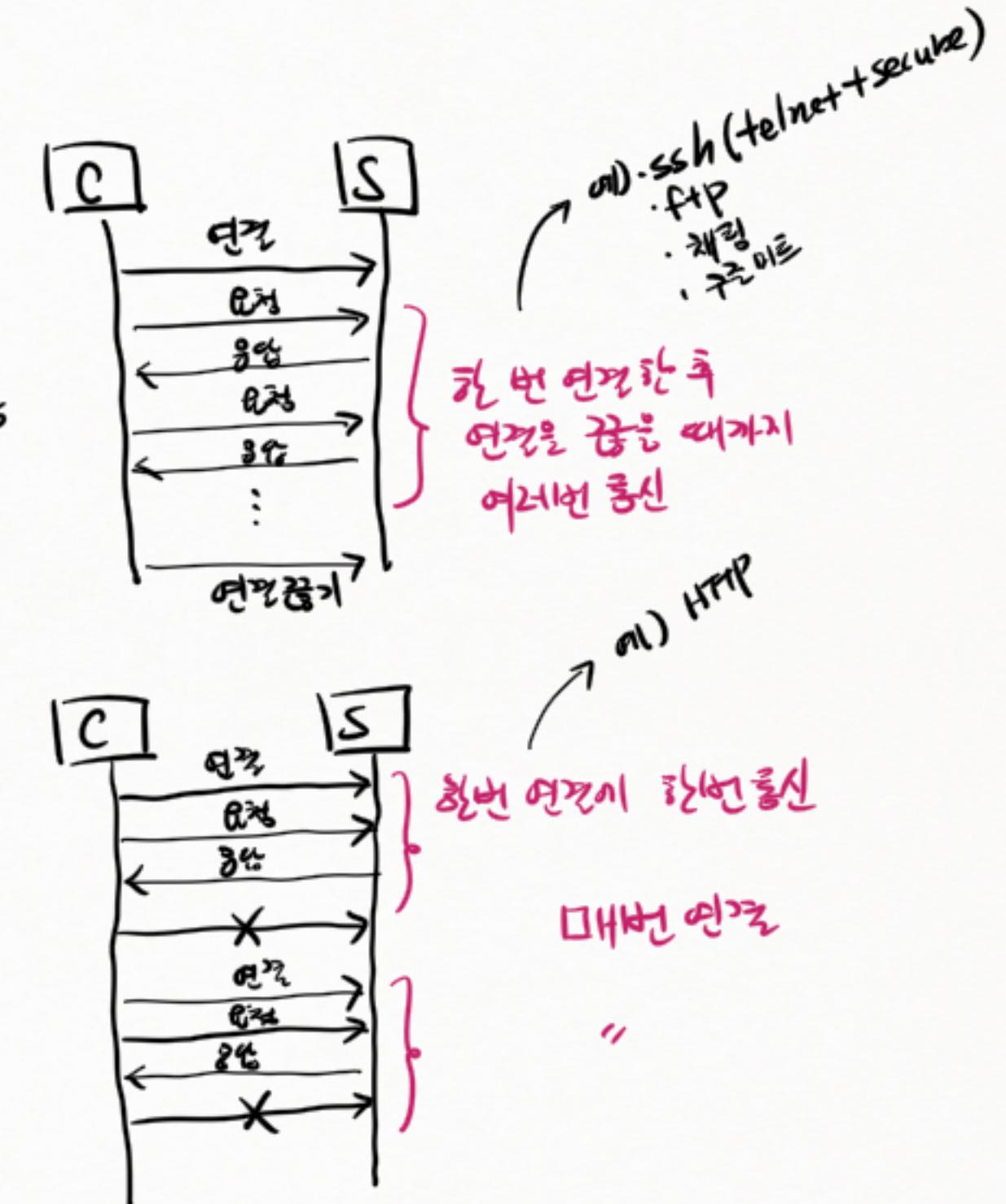
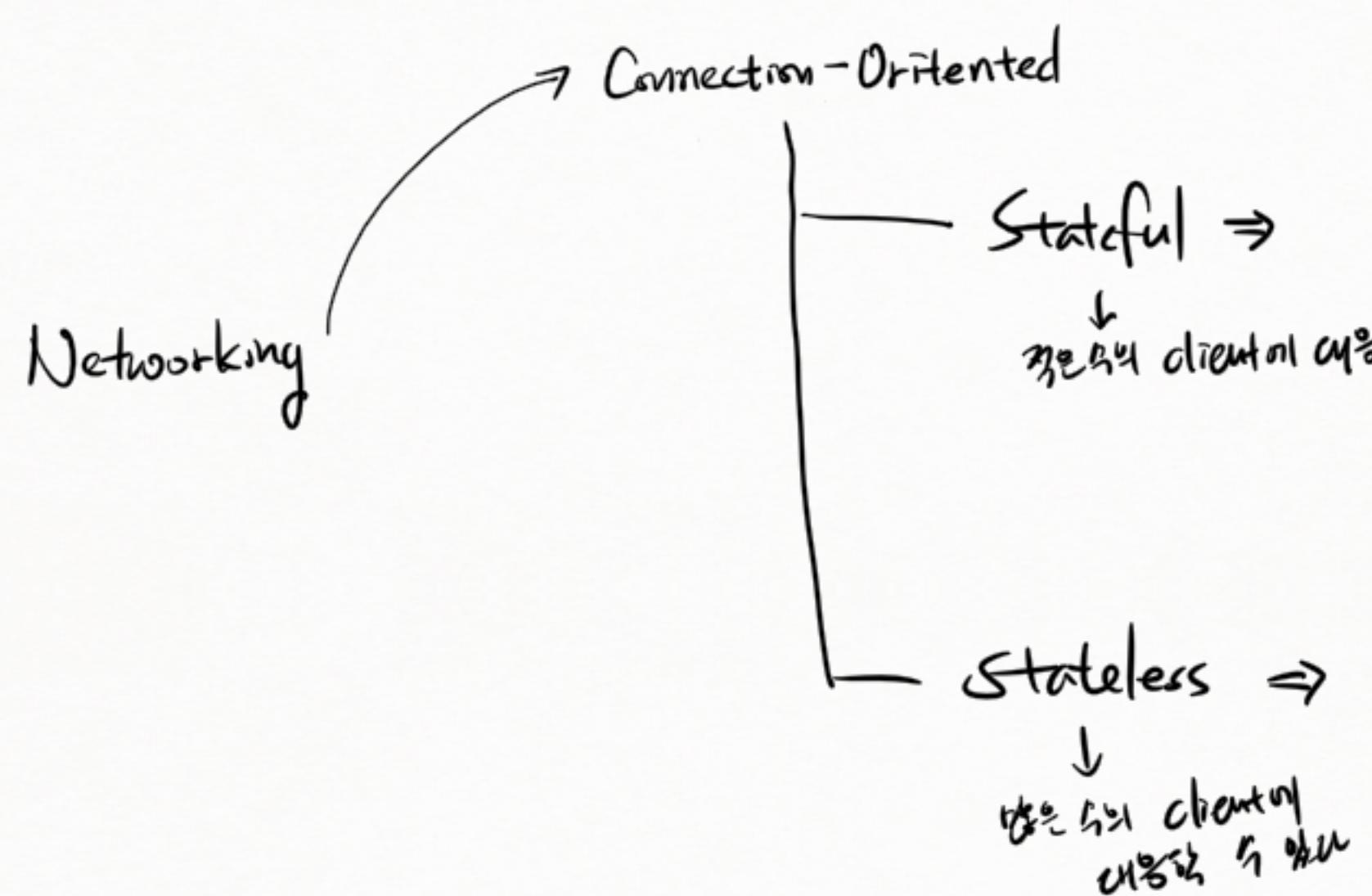
파일

파일 확장자는 Web 등 여러 가지 확장자로

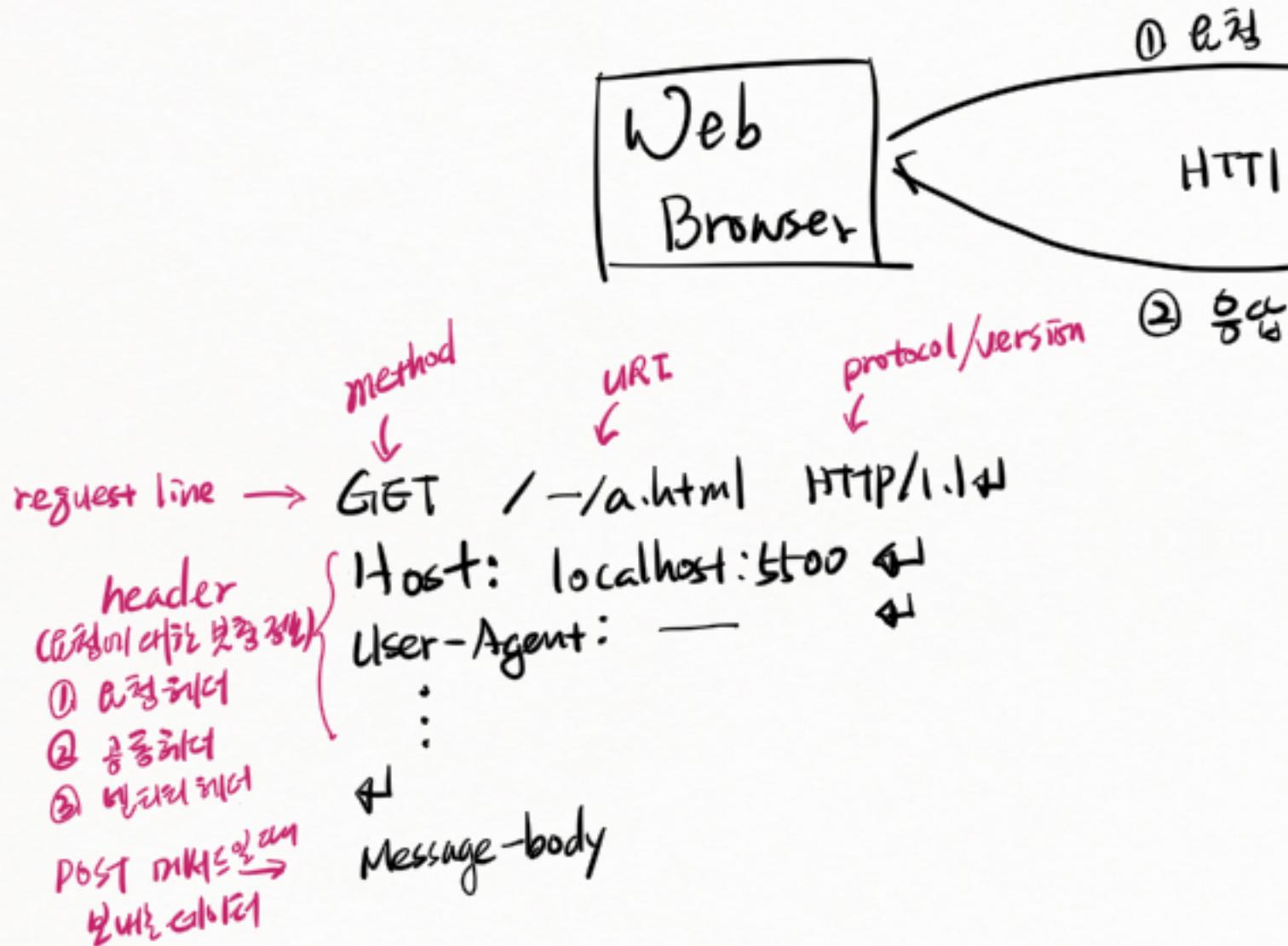
HTTP



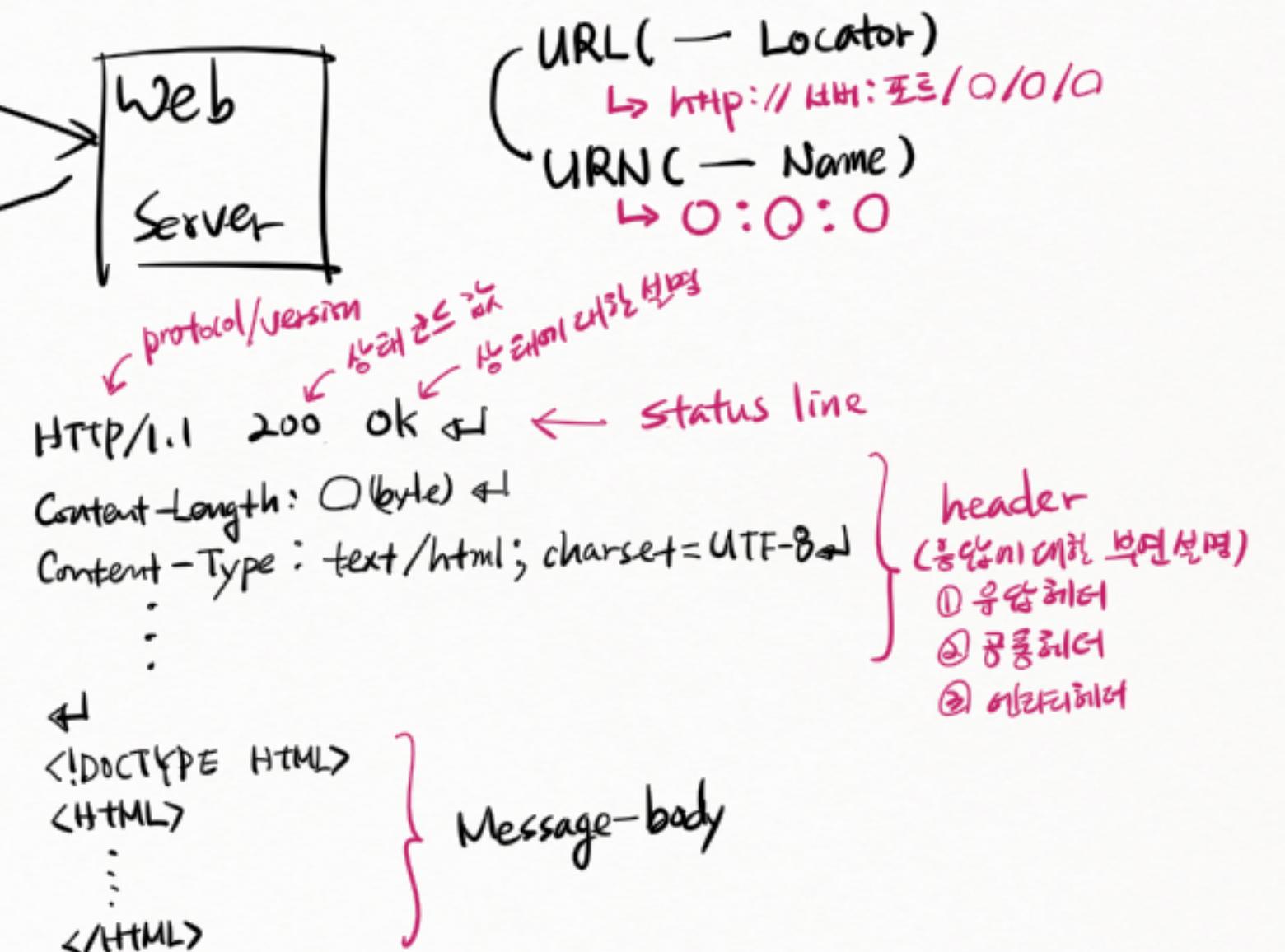
## \* 네트워킹



## \* HTTP 요청 / 응답

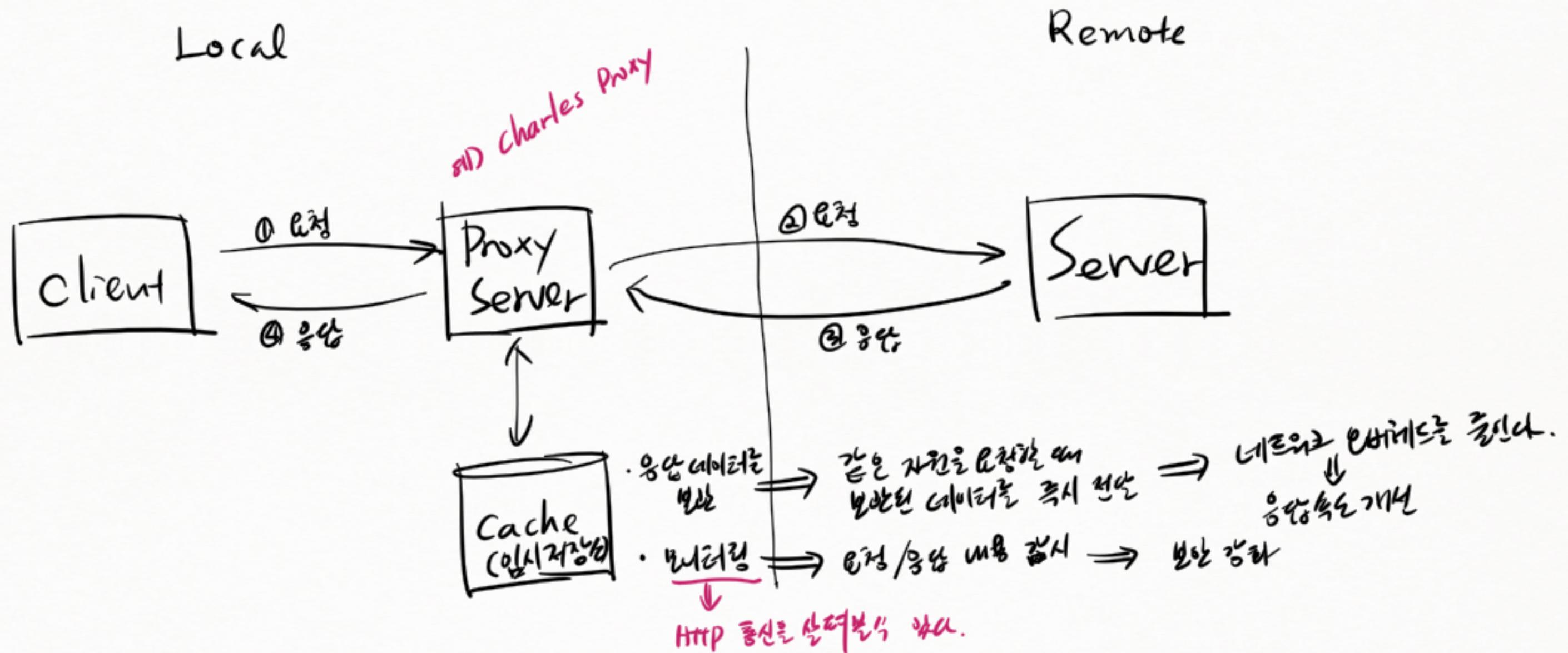


## \* URI (Uniform Resource Identifier)

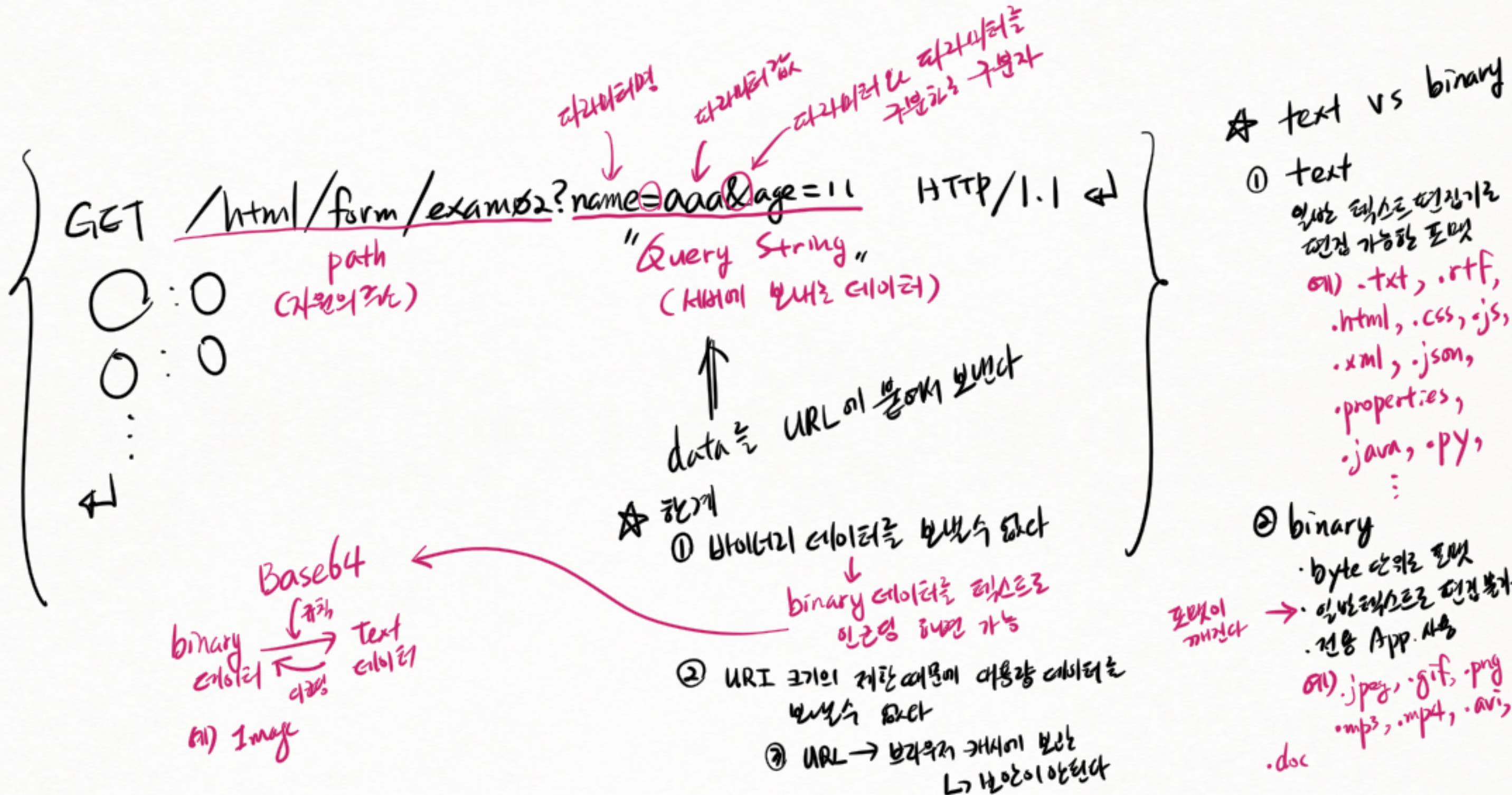


\* Proxy HTTP

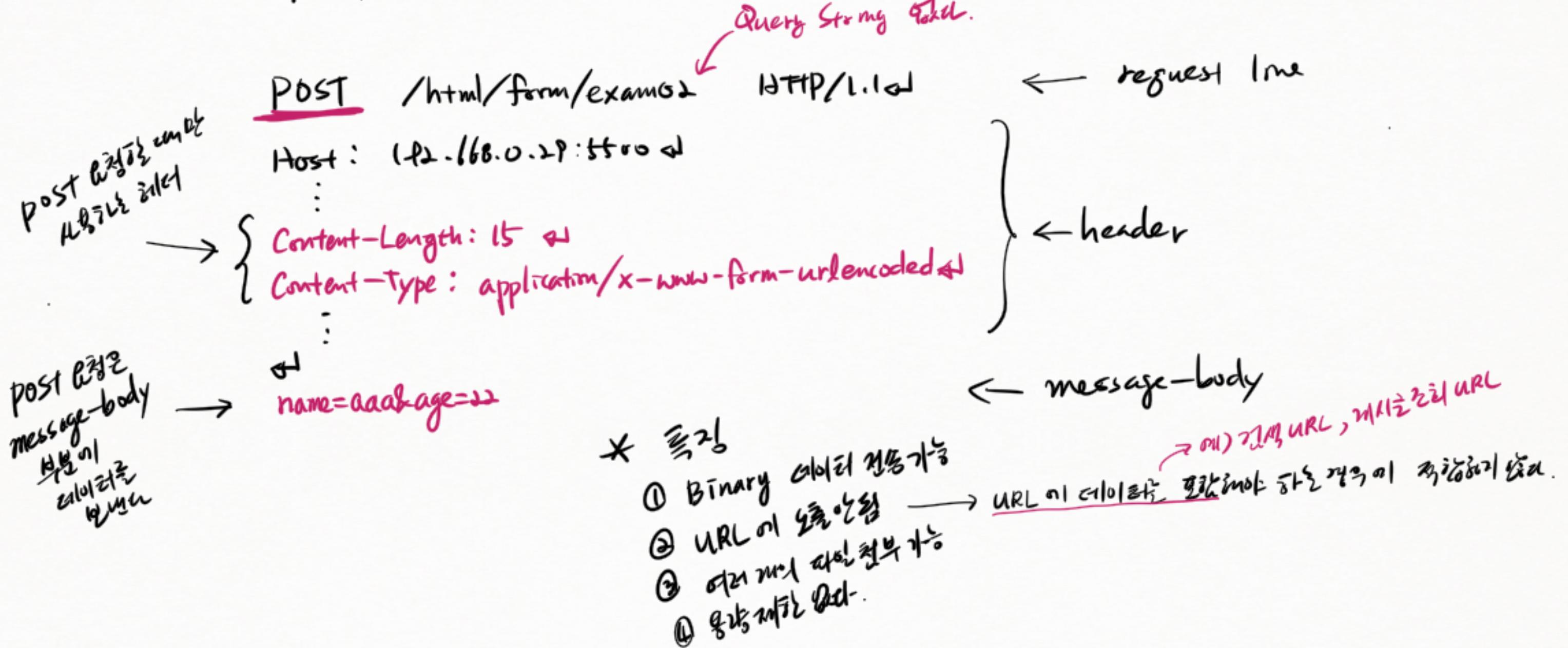
↑ client et Server 간에서 통신을 증가



## \* GET 요청



## \* POST 펴보기



\* name 속성

```
<input type="text" name="title">
```



? title = ABC

자바스크립트

자바스크립트

자바스크립트

\* name 키워드 II

```
<input type="text" name="name">  
<input type="tel" name="phone">
```

abc

010-1111-2222



? name=abc & phone=010-1111-2222

css

\* CSS : HTML element의 모양을 정의

↳ Cascading Style Sheet

selector

스마일 적용할 대상을  
가리킨다.

.content header > span.title:hover {

background-color: red;

color: white;

}

↑  
style name

↑ style value

pseudo selector  
(선택자의 형태)

⇒ 셀렉터 { 스마일: 값; 스마일: 값; ... }

☆ ① selector 히어러

☆ ② specificity (스마일 적용순서)

③ box 다루는 방법  
(레두리, 여백, 박스 레이아웃)

④ 폰트 다루는 방법

⑤ 색상 지정하는 방법

⑥ 배경 다루는 방법

⑦ block or inline 다루는 방법

⑧ 위치 조정하는 방법



## \* Selector

\* {} — }

\* {} — }

태그명 {} — }

body {} — }

태그명, 태그명, 태그명 {} — }

img, ul {} — }

.2중명 {} — }  
class

.c1, .c2 {} — }

#태그아이디 {} — }

#content {} — }

-[속성명=값] {} — }

-:상태명 {} — }  
pseudo selector

li:hover {} — }

진상 자손 {} — }

#content li {} — }

부모 > 자식 {} — }

ul > li {} — }

태그 + 다음태그 {} — }

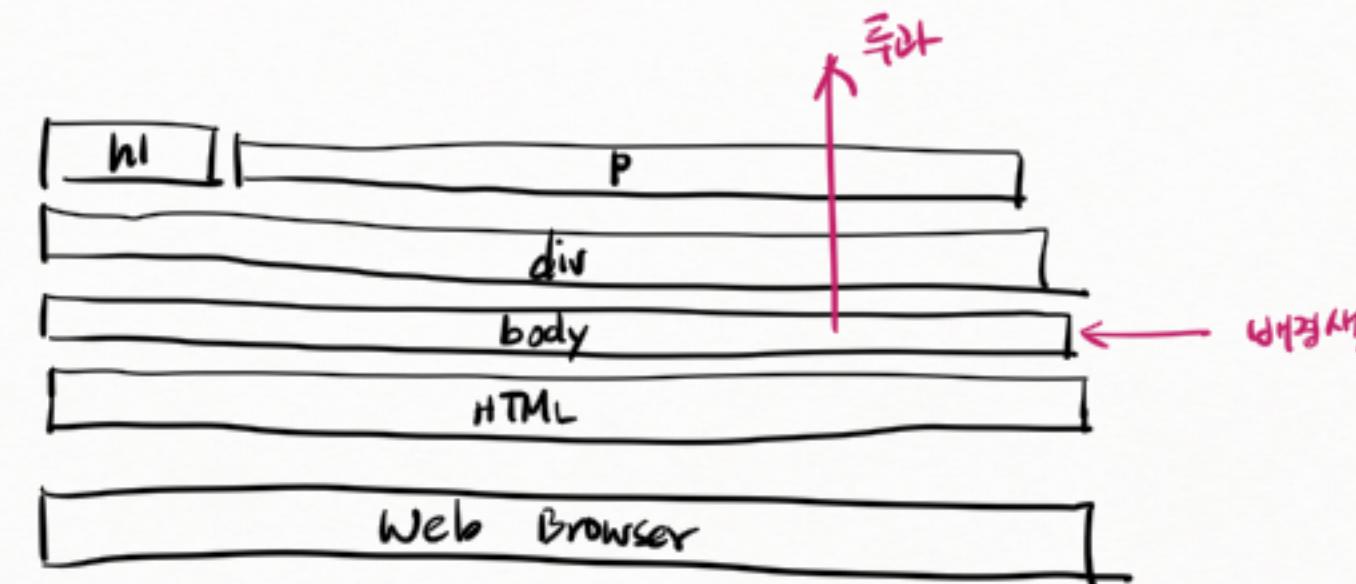
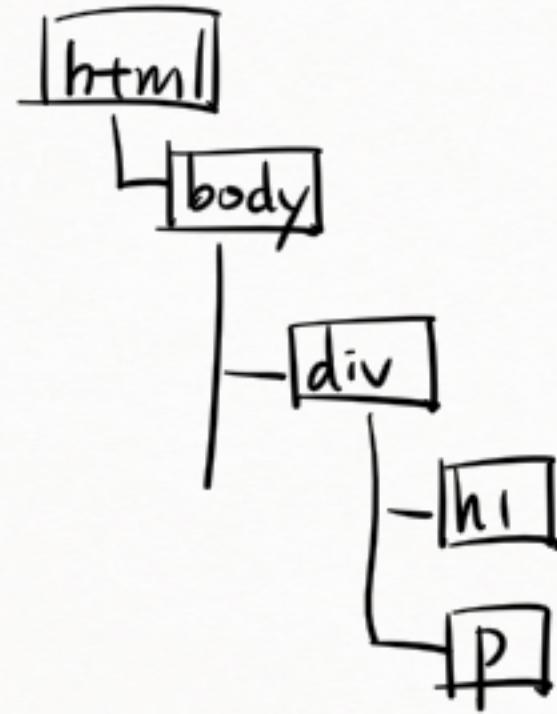
img + ul {} — }

대상자 조건 값 ... {} — }

div#menu li:c2 {} — }

input[type="text"] {} — }

\* 부록 태그와 자식 태그의 위치 찾기



\* CSS specificity : 스타일 적용 순서 낮 → 높

\* → 0

id, pseudo selector → 1

class, tag → 10

attribute → 100

inline style → 1000

\* {}

h1 {} a:hover {}

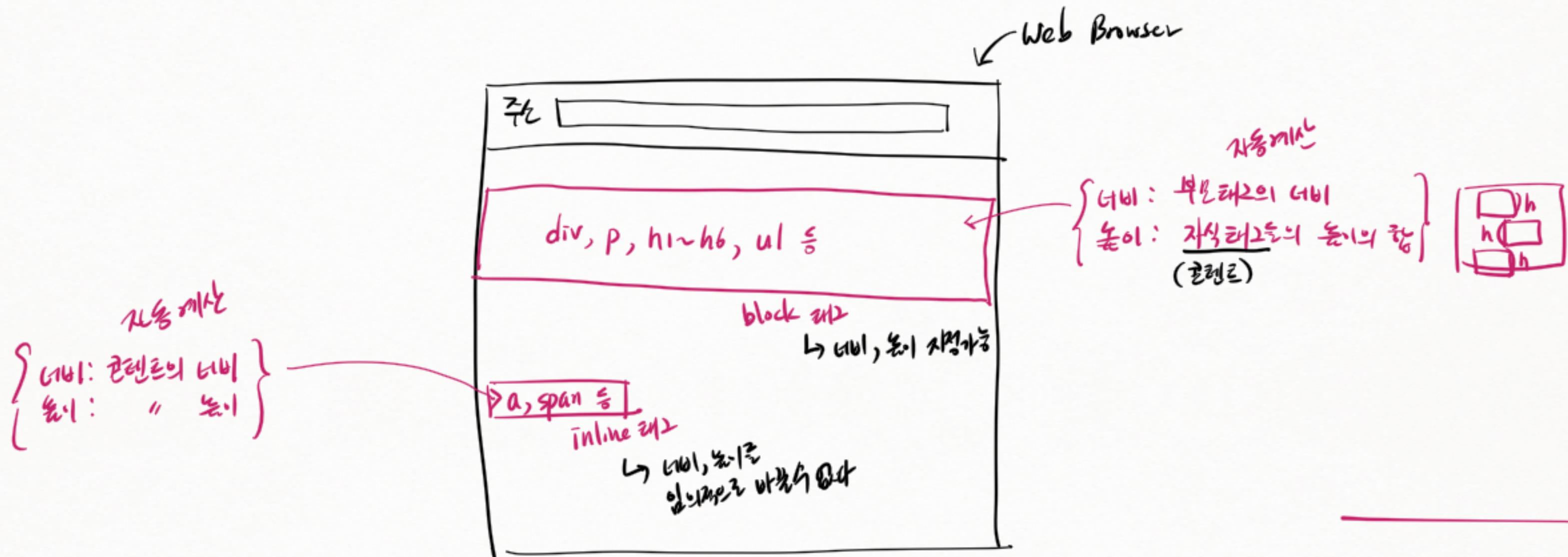
.cl {} input[readonly] {}

#menu {}

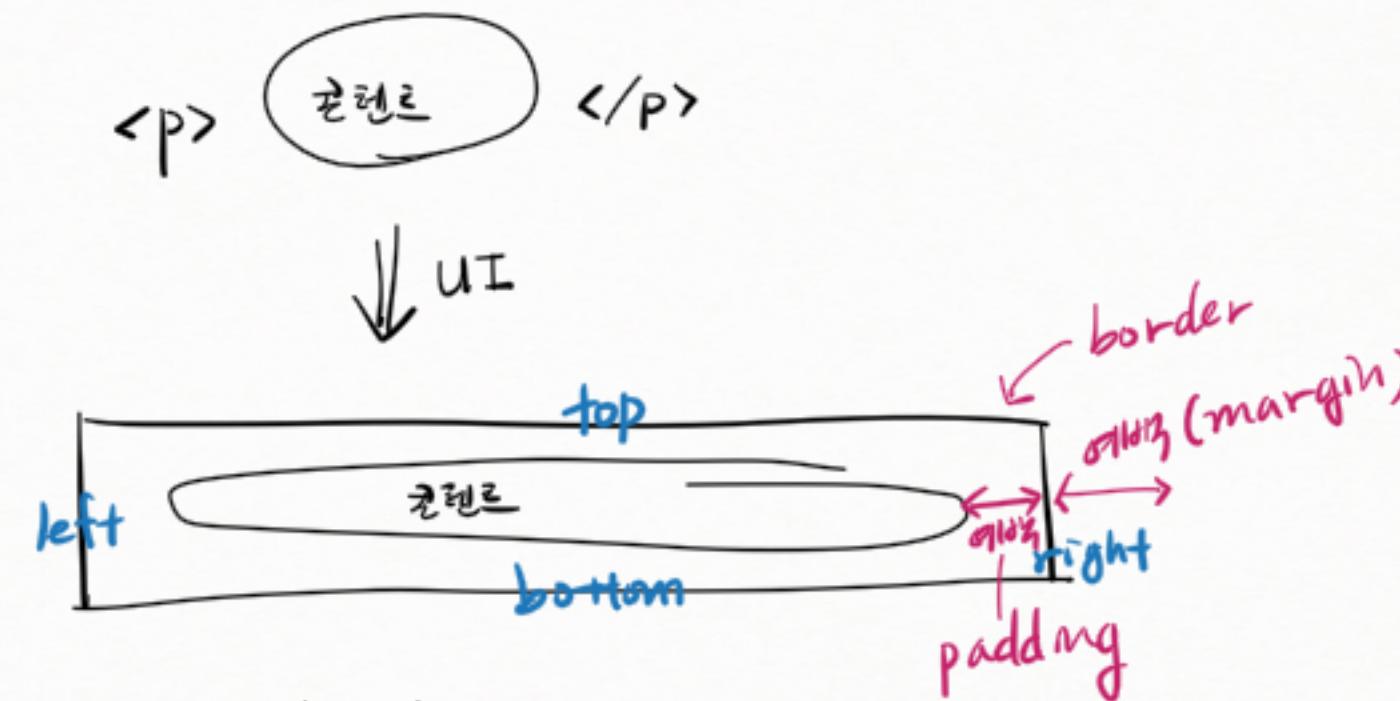
<a href="#" style=" " > () </a>

인라인 스타일

## \* 태그의 너비, 높이



## \* 레이아웃 (border)



right  
bottom color dotted  
border-top-style: solid;  
left width dashed  
:

짧은 명령  
 $\Rightarrow$  border-top: color style width;  
bottom  
right  
left

border-style: solid;  
↓  
모든 방향의 레이아웃 스타일

더 짧은 명령  
 $\Rightarrow$  border: color style width;  
↓  
레이아웃 방향

## \* Raster 폰트 Vector 폰트

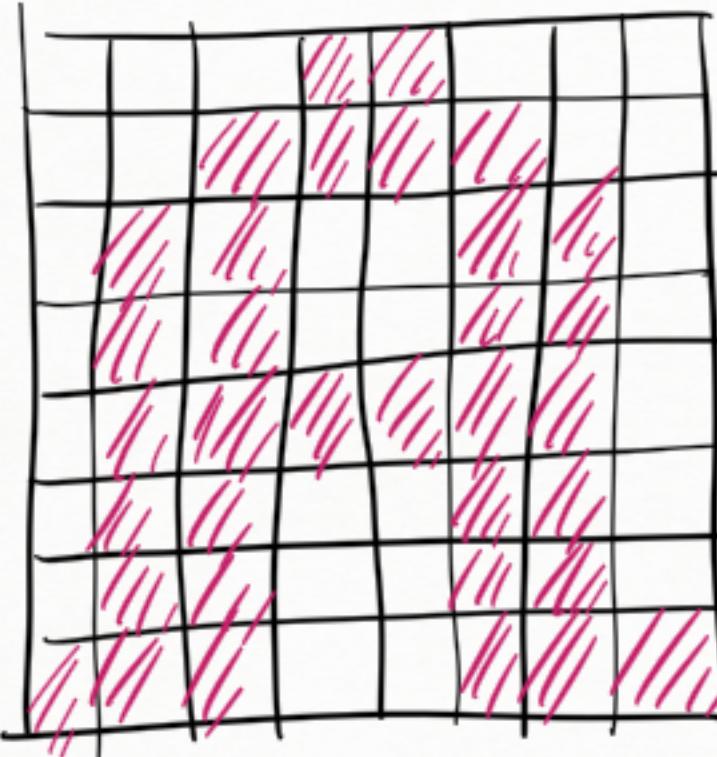
↳ 비트맵 이미지 예) .bmp, .gif, .png, .jpg ...

↳ 벡터이미지 예) 층집마크, 캐드

Raster 폰트

↳ 퍼센트 단위로 글자를 만든다

- 한글 속도가 빠르다
- 폰트크기에서 따라  
각 문자를 만든다
- 정해진 크기보다  
더 크게 한글 속도를  
얻을 때 각 글자의  
크기를 늘리기 때문이다  
제작 현상 발생
- 이미지의 불확정화  
상관없이  
파일크기는 같다.



(1) COURIER

Vector 폰트

→ 글자를 그리는 명령어를 작성한다

• 글자를 그리는 명령을 수행

한글 속도가 느린다  
(같은 CPU가 빠르기 때문에  
상관하지 않는다)

• 글자 크기를 늘리더라도  
명령을 통해 그리기 때문에  
제작 현상이 발생하지  
않는다.

• 이미지 크기 → 파일크기 ↓  
" 불확정 → " ↑

\* 폰트크기 = 높이

4~5 line

3~6 line

(3,3) (1,8) line

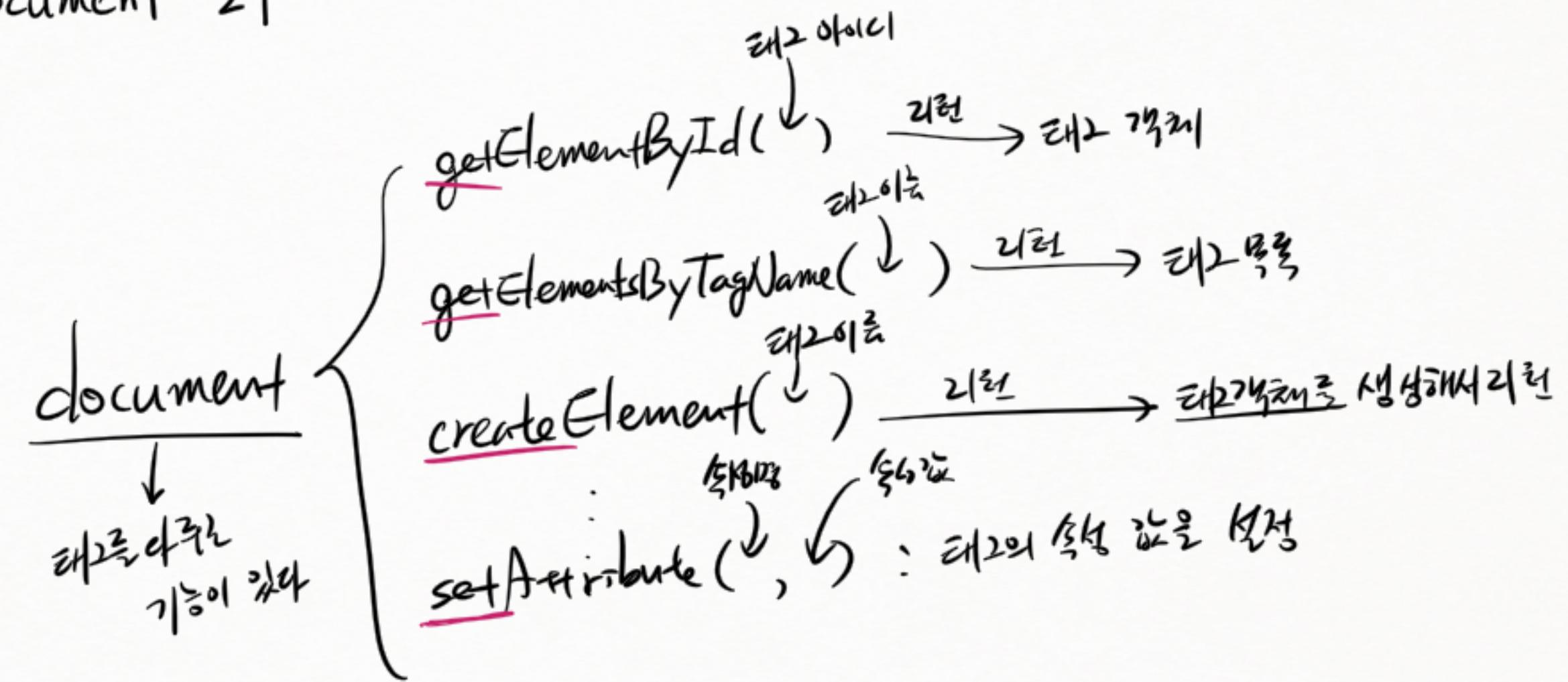
⋮

A j

↑ 글자 높이  
↓ 폰트크기

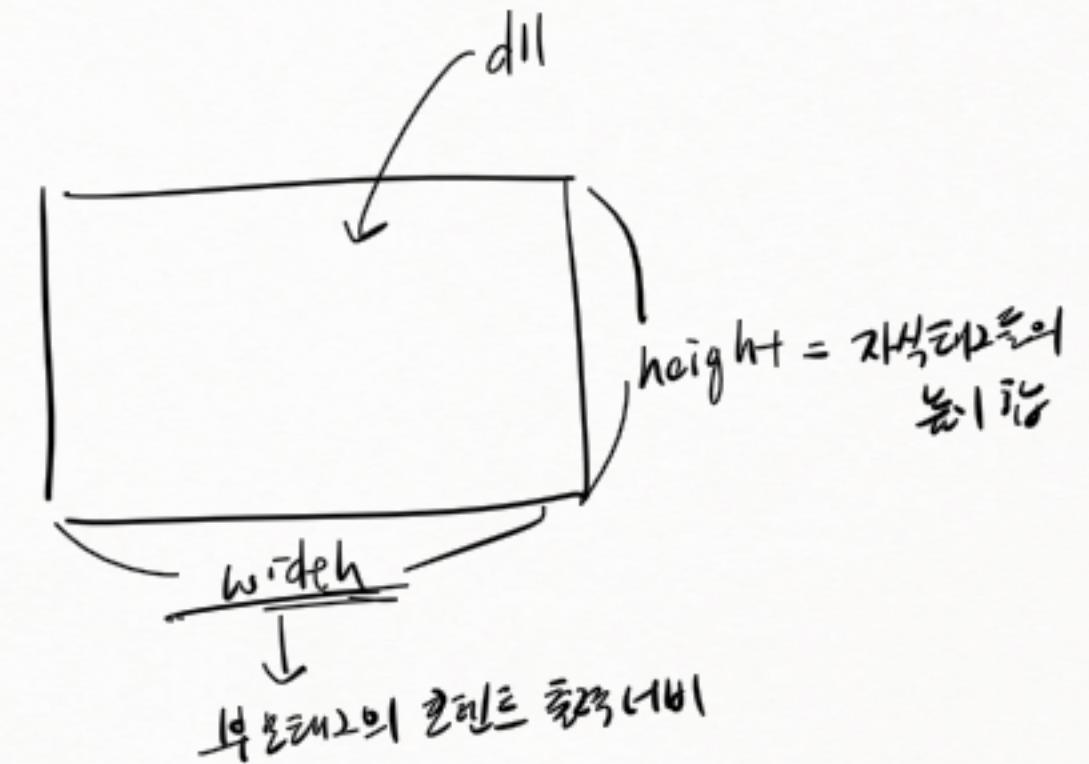
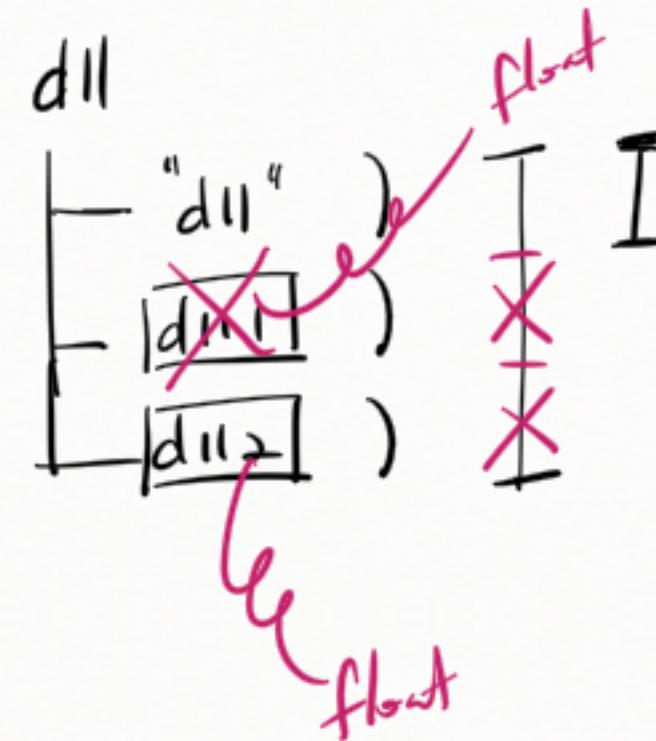
(1) True-Type 폰트  
courier new

## \* document 속성

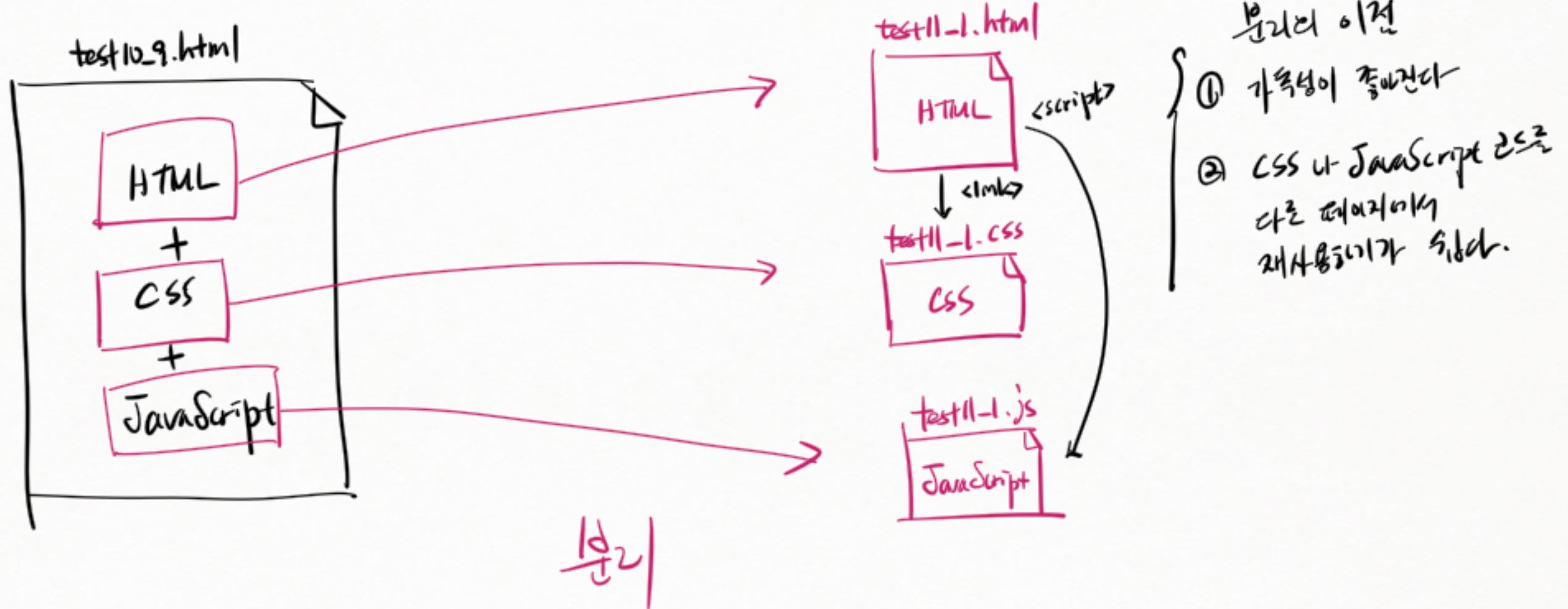


## \* float 2f absolute

↳ 부모태그의 크기를 계산할 때 float는 absolute 태그를 제외한다.

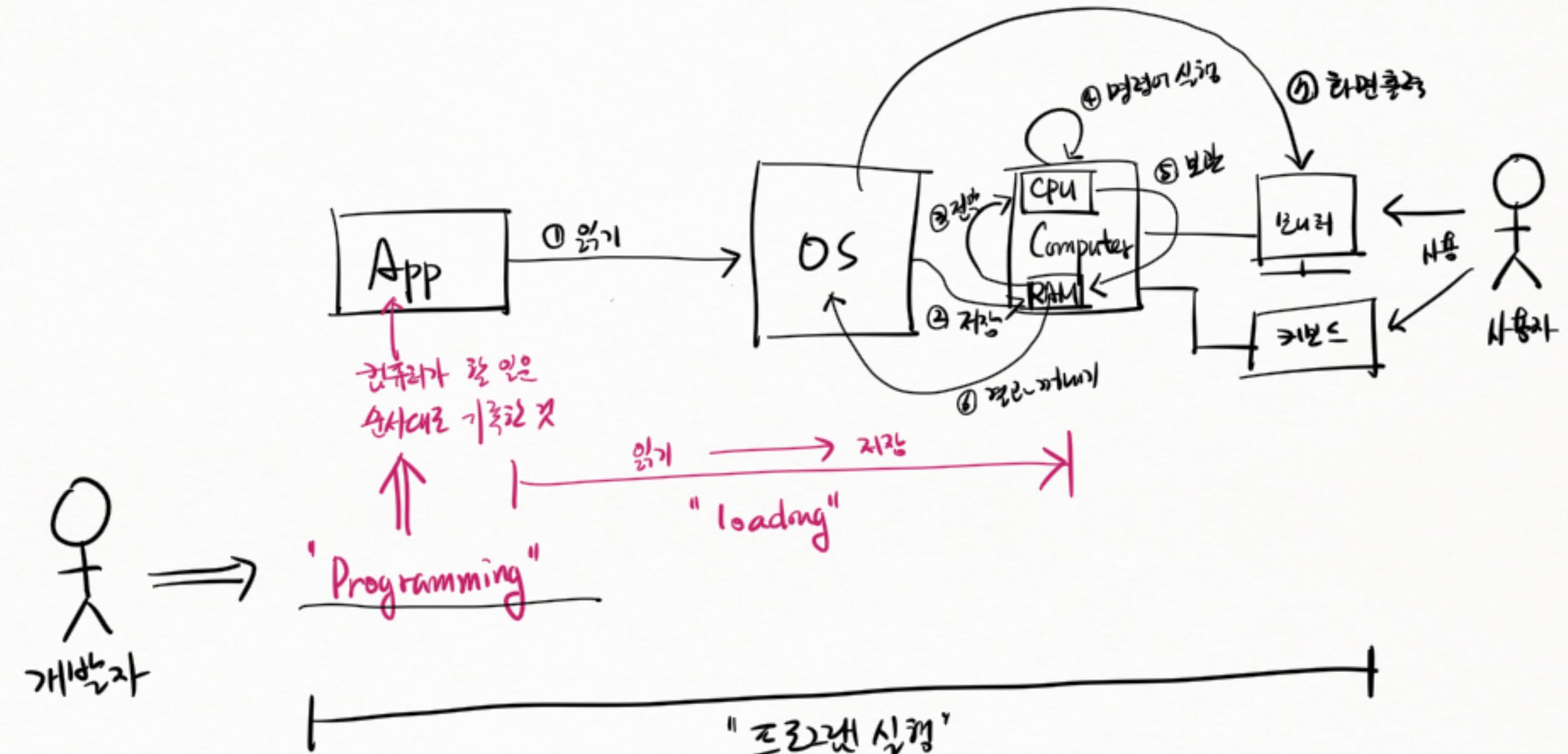


## \* CSS와 JavaScript 코드의 분리

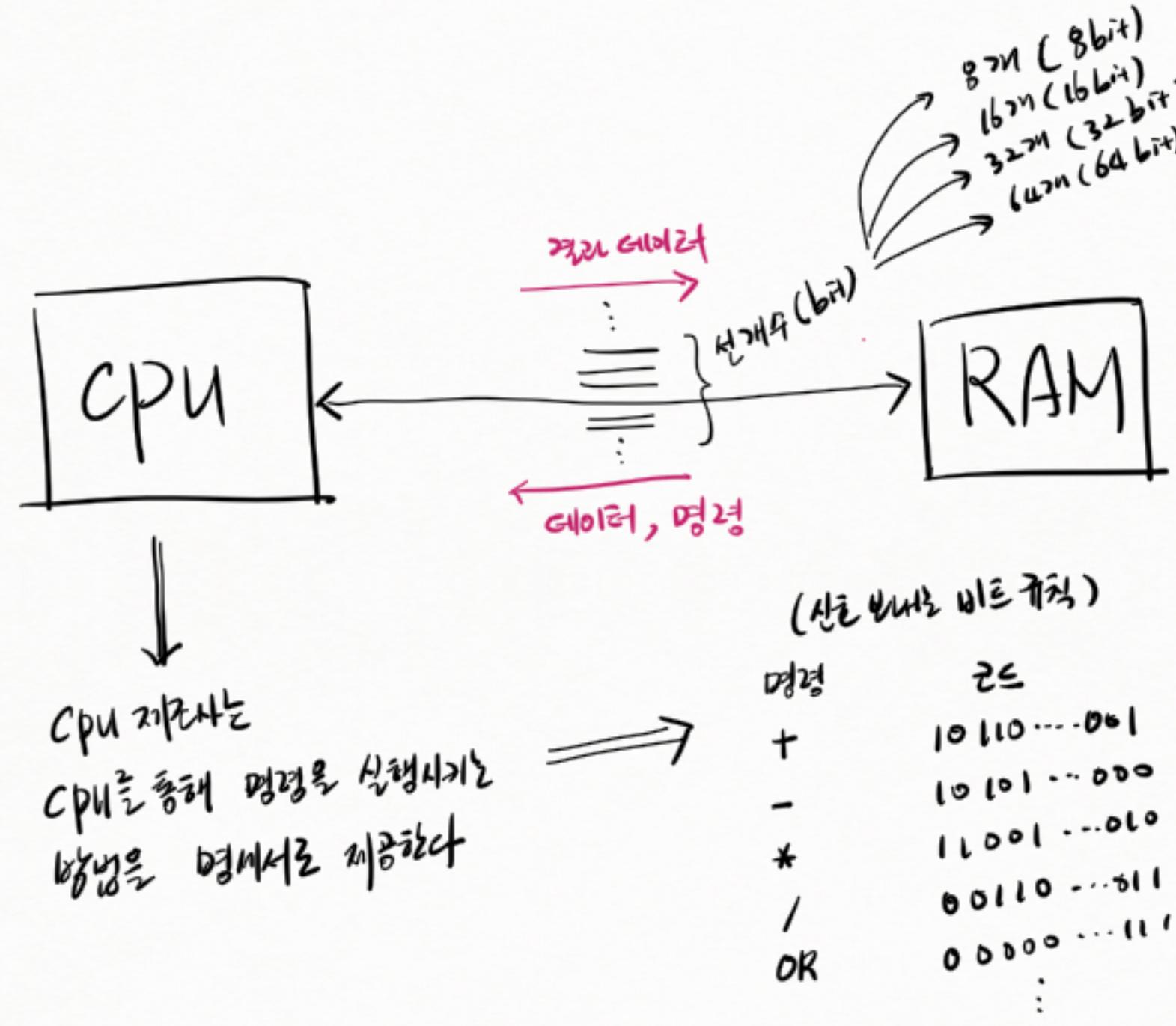


JavaScript

## \* 프로그램 실행과 프로그래밍



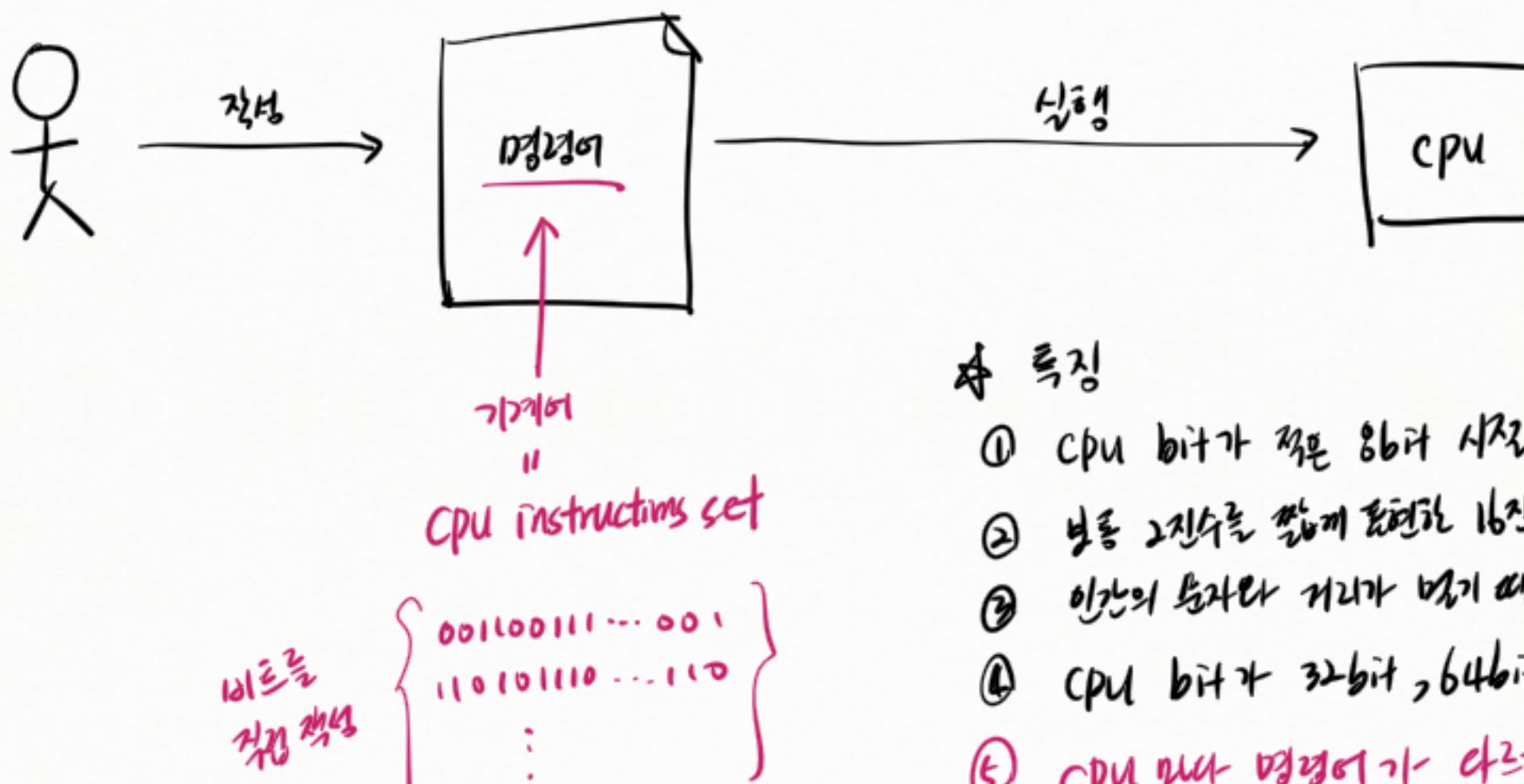
## \* CPU 와 RAM , bit



$$\begin{array}{r}
 1 \ 1 \ 1 \ 1 \ 0 \ 1 \\
 0 \ 0 \ 0 \ 0 \ 1 \ 0 \quad + \\
 1 \ 1 \ 0 \ 0 \ 1 \ 1 \quad - \\
 1 \ 0 \ 0 \ 0 \ 1 \ 0 \quad *
 \end{array}$$

\* 명령어 작성 : 기계어

## ① CPU instructionset 명세서를 보고 직접 명령어 작성

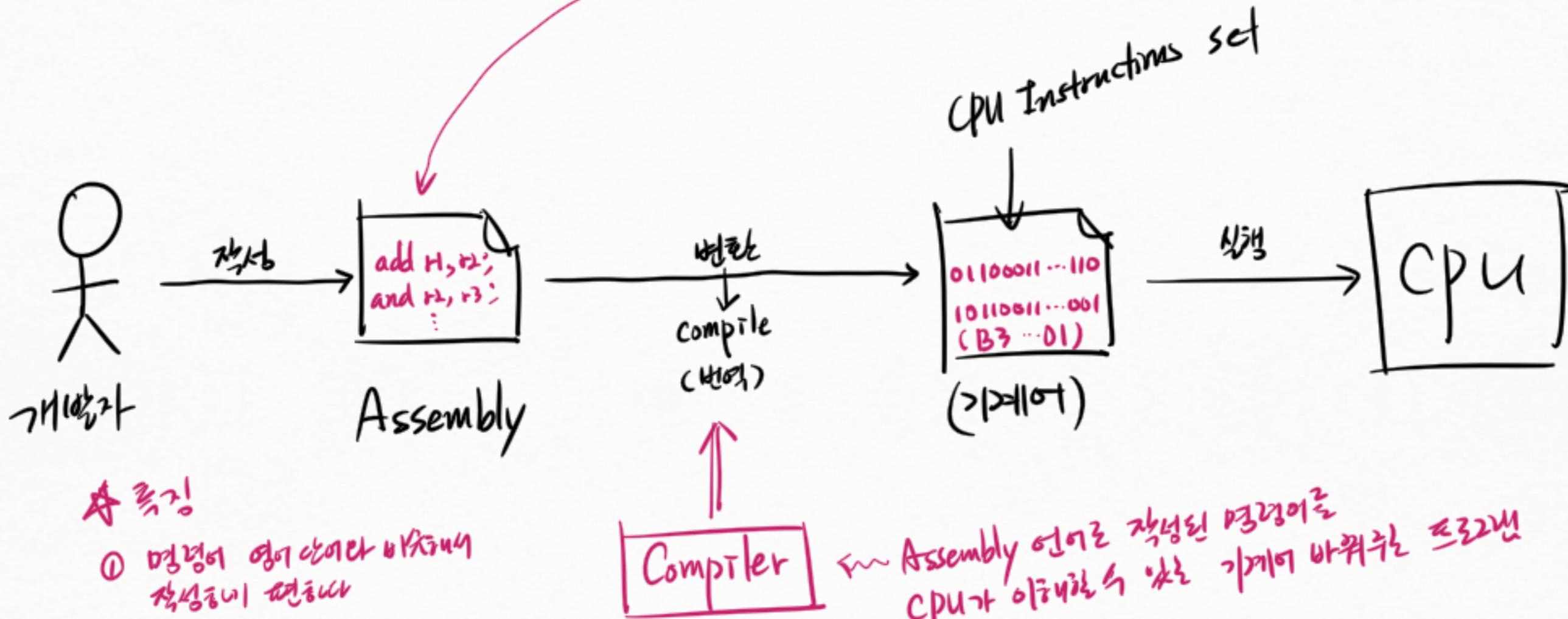


### \* 특징

- ① CPU bit가 적은 8bit 시장에는 개발자가 직접 작성하기도 한다
- ② 보통 2진수를 끌어와 표현한 16진수를 사용해서 작성한다
- ③ 이진의 둘자와 거리가 멀기 때문에 작성하기 매우 불편하고 힘들다.
- ④ CPU bit가 32bit, 64bit로 늘어나면서 더 어렵게 되었다
- ⑤ CPU마다 명령어가 다르기 때문에 다양한 CPU에서 실행할 수 있는 명령어는 작성하기 매우 힘들다.  
전기적 신호  
비트 구조

## \* 명령어 작성 : Assembly

② 직장 → 기계어로 작성하는 대신 간접적 영어 문장으로 이루어진 명령어 사용



★ 특징

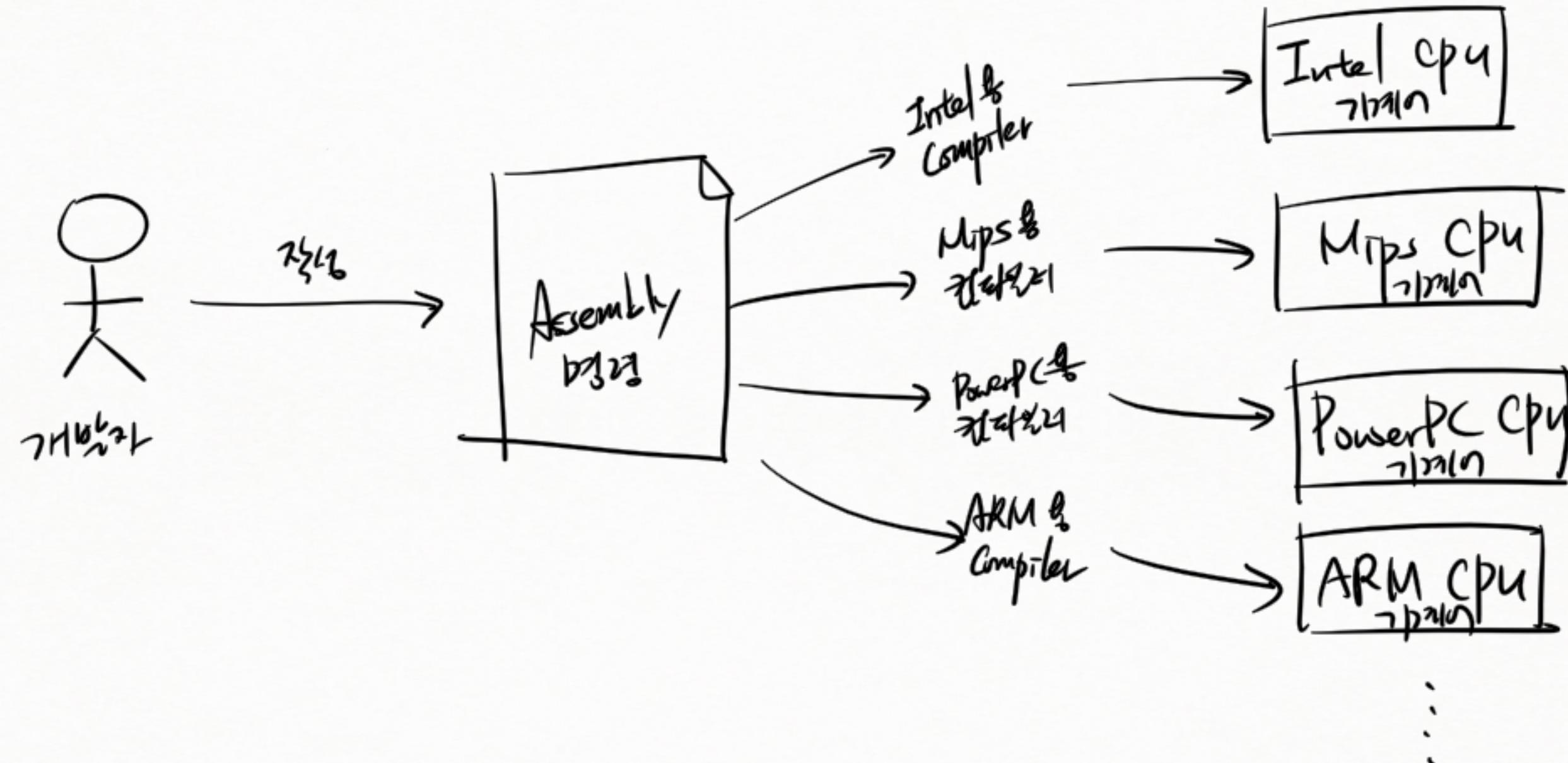
- ① 명령어 영어 문장과 비슷하여 작성하는데 편하다
- ② CPU마다 다르게 작성되어 헷갈리다

왜?

컴파일러가 CPU에 맞춰서  
기계어로 번역해 준다.

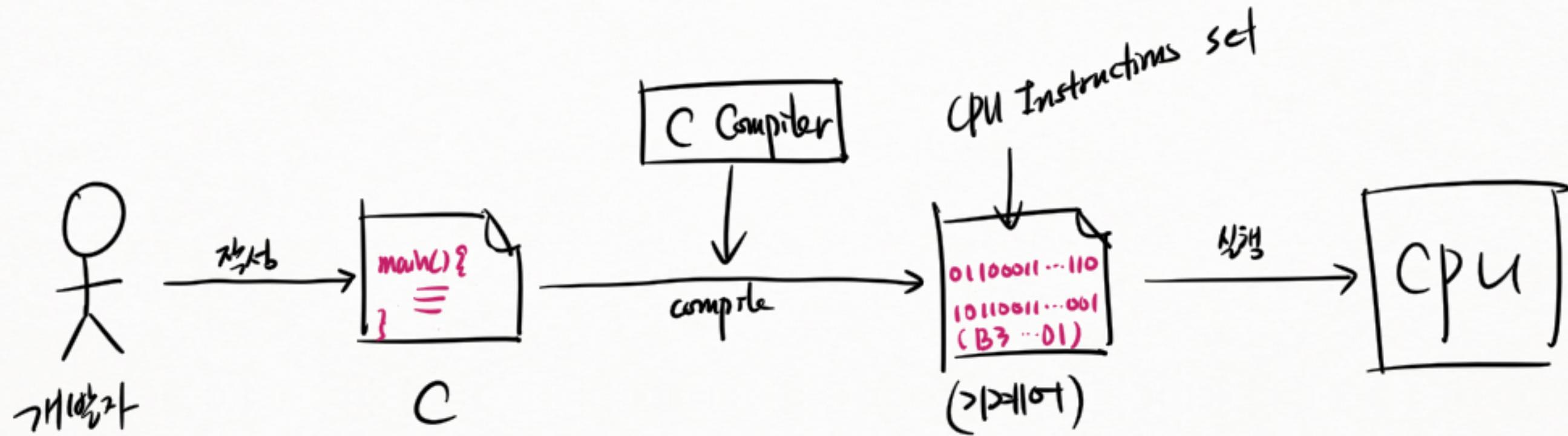
~ Assembly 언어로 작성된 명령어를  
CPU가 이해할 수 있도록 기계어 바꿔주는 프로그램

\* 명령어 작성 : Assembly 및 컴파일러

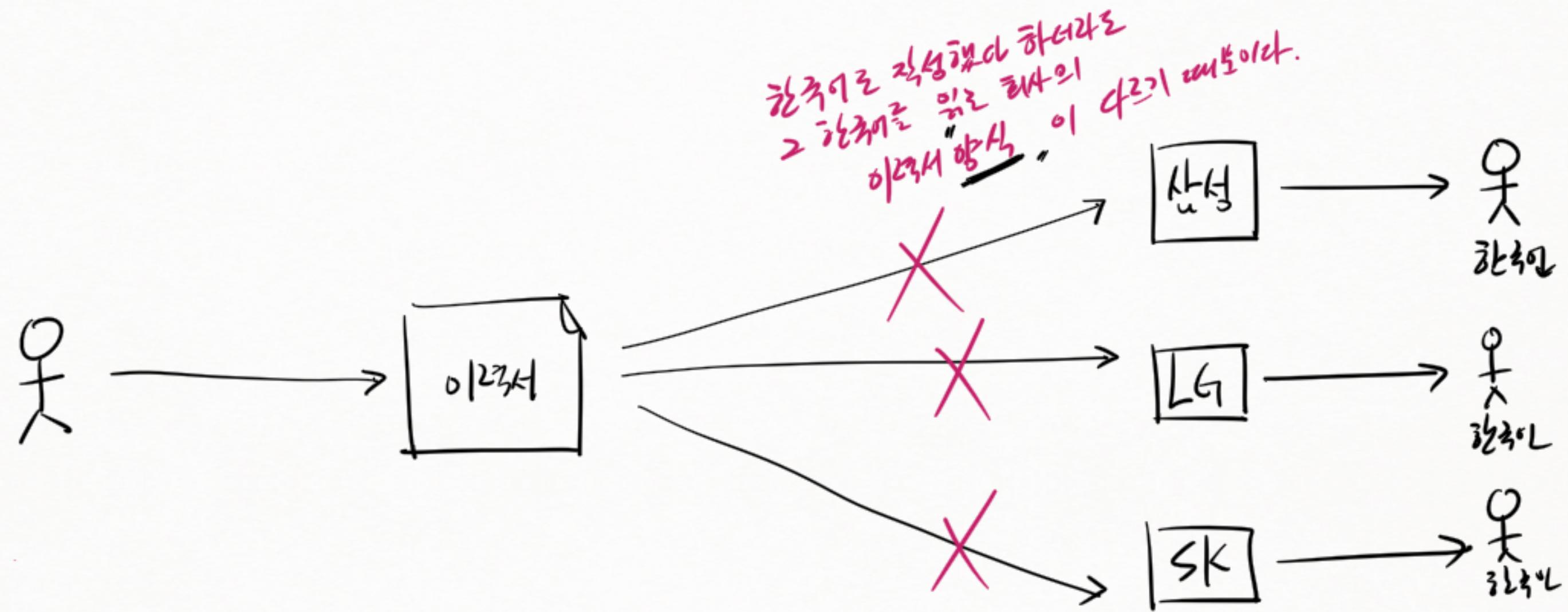


\* 명령어 작성 : C

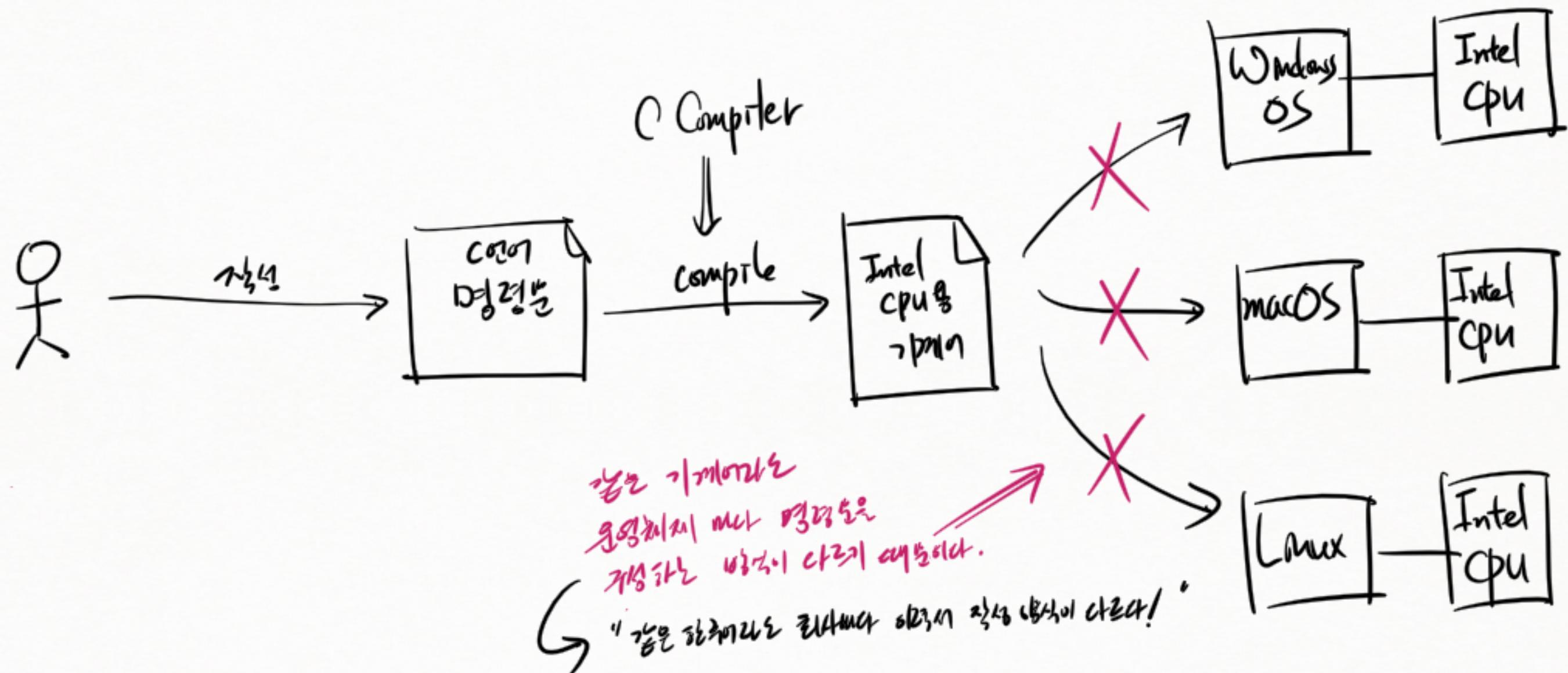
③ 더 많은 친화적인 프로그래밍 언어로 프로그램 작성하기



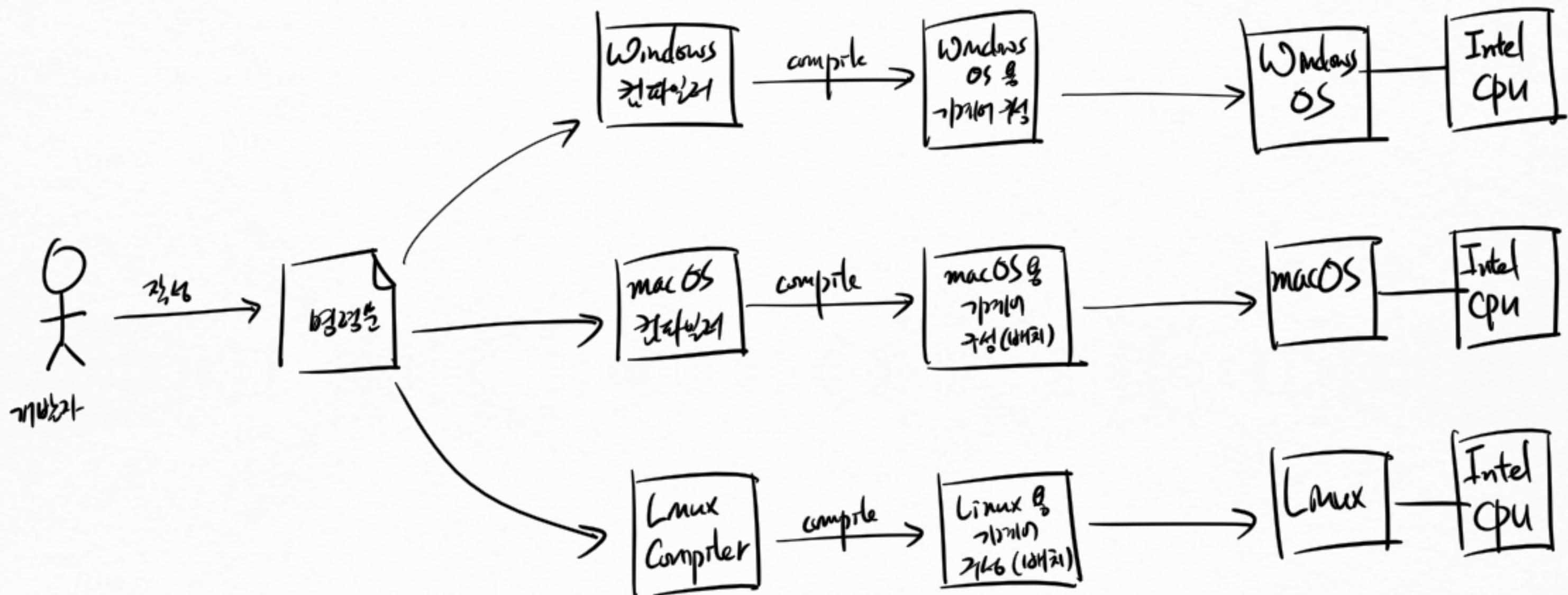
\* 가기어, OS, CPU



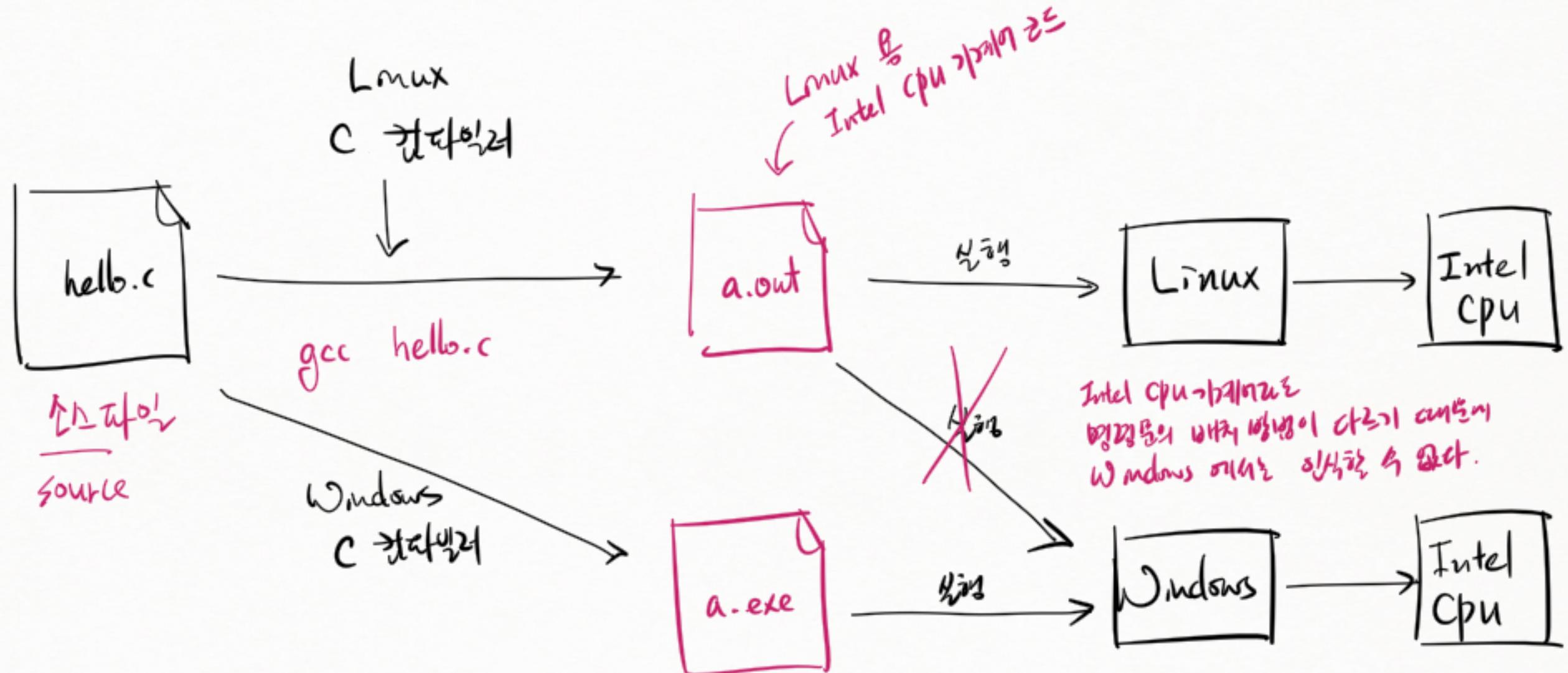
\* 가상화, OS, CPU



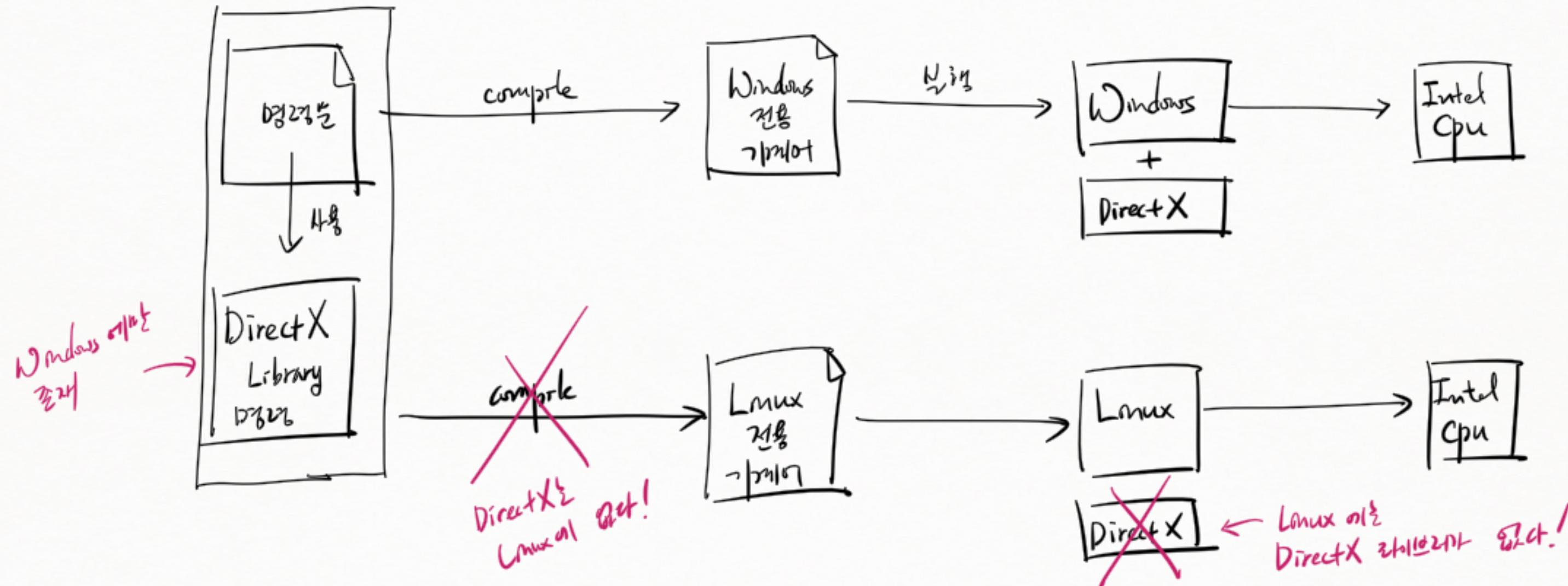
## \* 가기어, OS, CPU



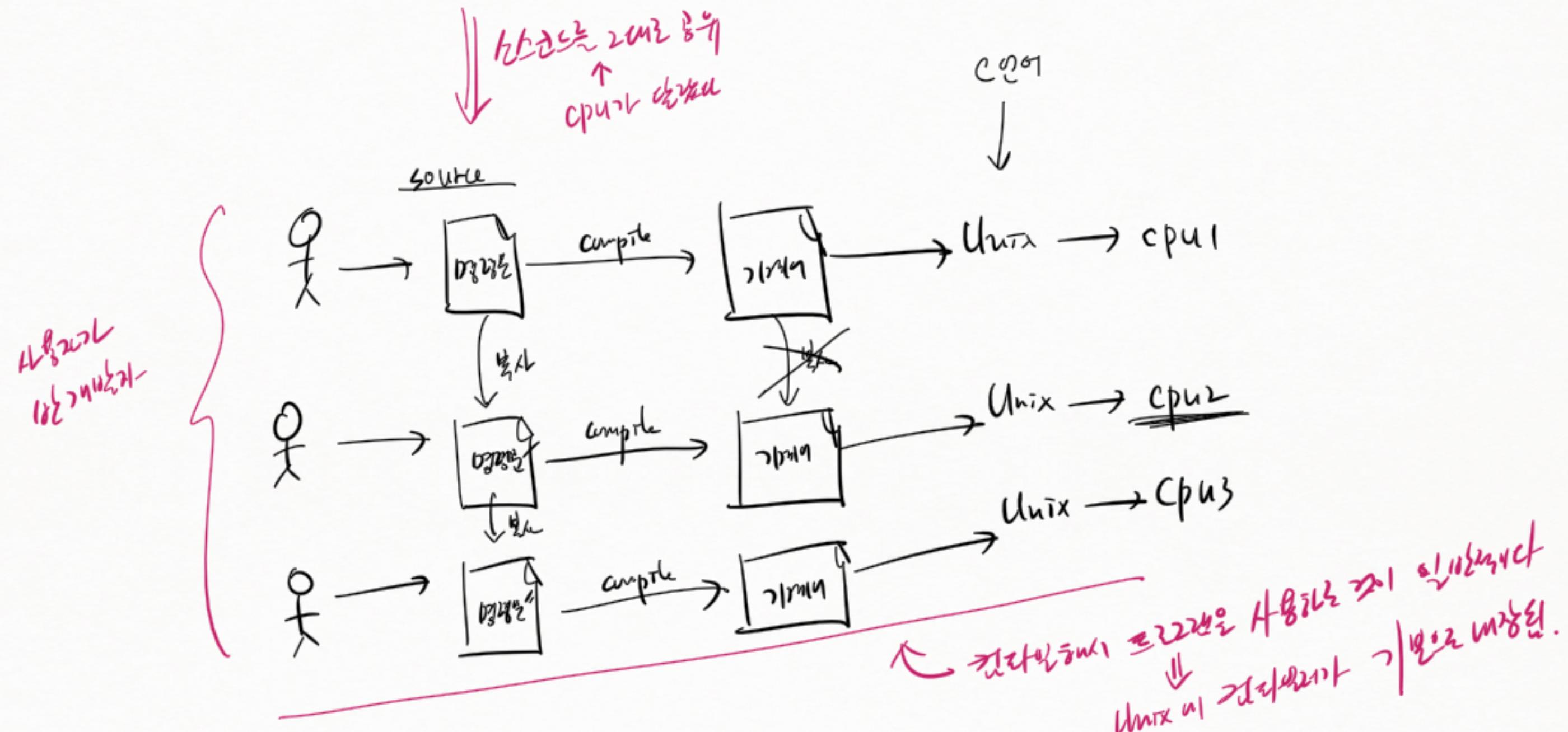
\* C → 실행



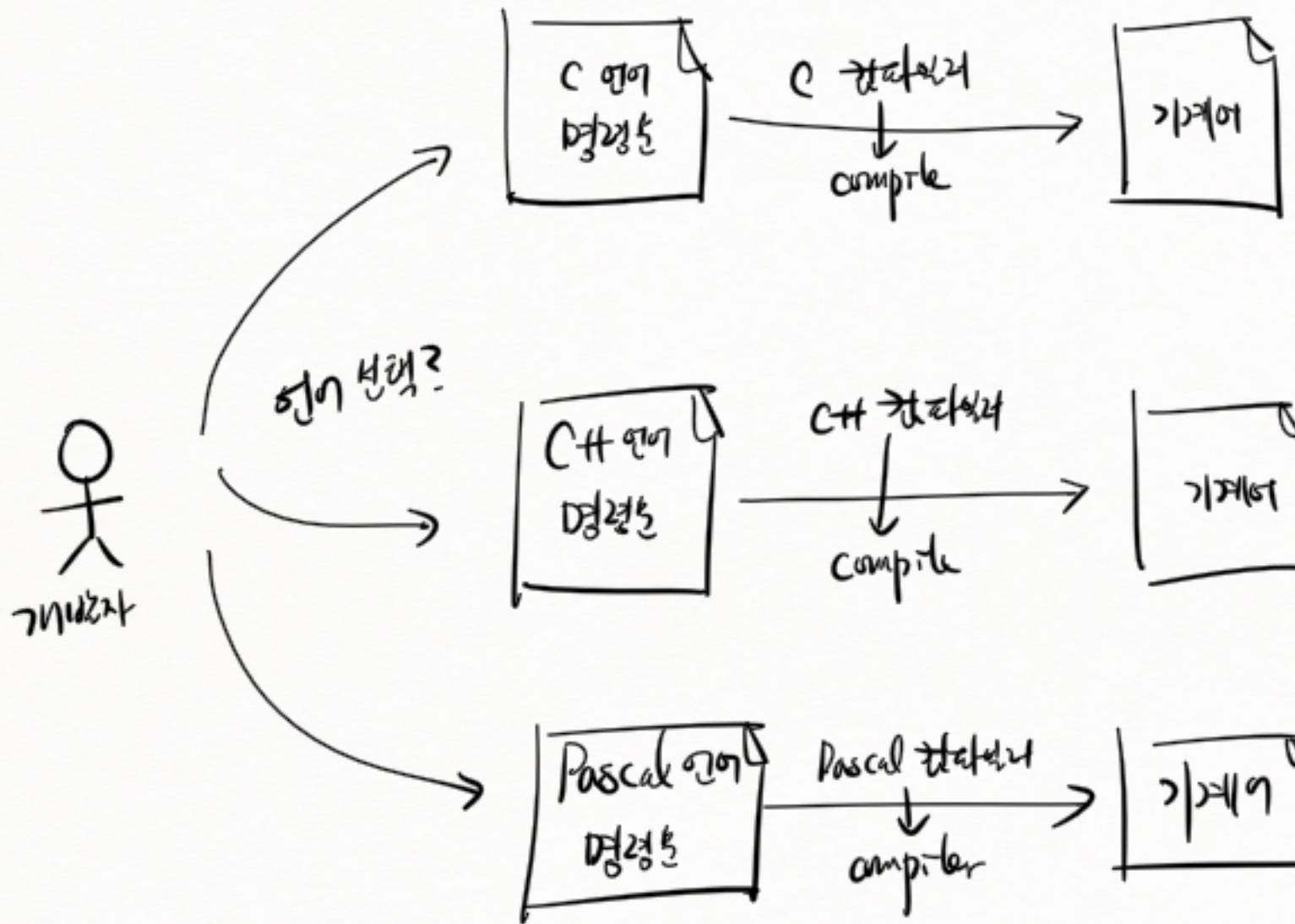
\* C 언어 + OS 전용 명령 사용



\* 예전의 경제는 종유 투자



\* 프로그래밍 언어와 → 컴파일러



\* 언어 선택?

프로그래밍 언어마다 특장점이 있다



프로그래밍 언어마다 언어를 이해하는 데에

{ 인공지능, 머신러닝, 딥러닝 : Python

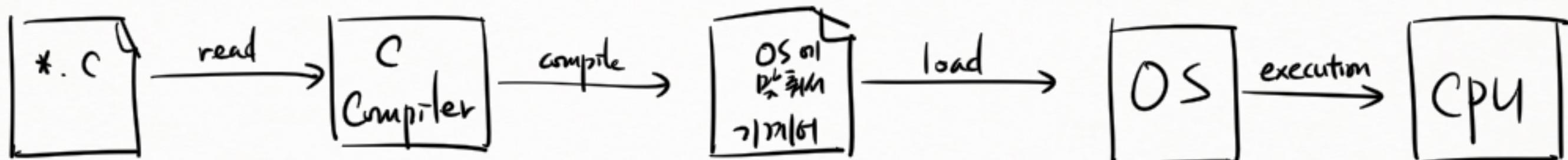
web app : java, php, go

web UI : javascript, TypeScript

통계 : R

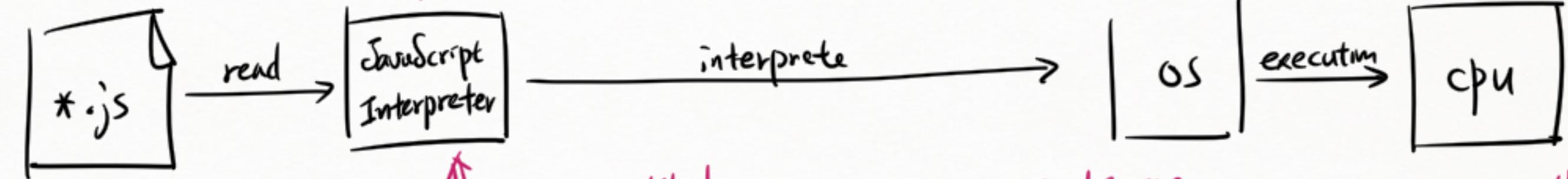
\* 컴파일 방식과 인터프리트 방식

→ 컴파일 방식 :



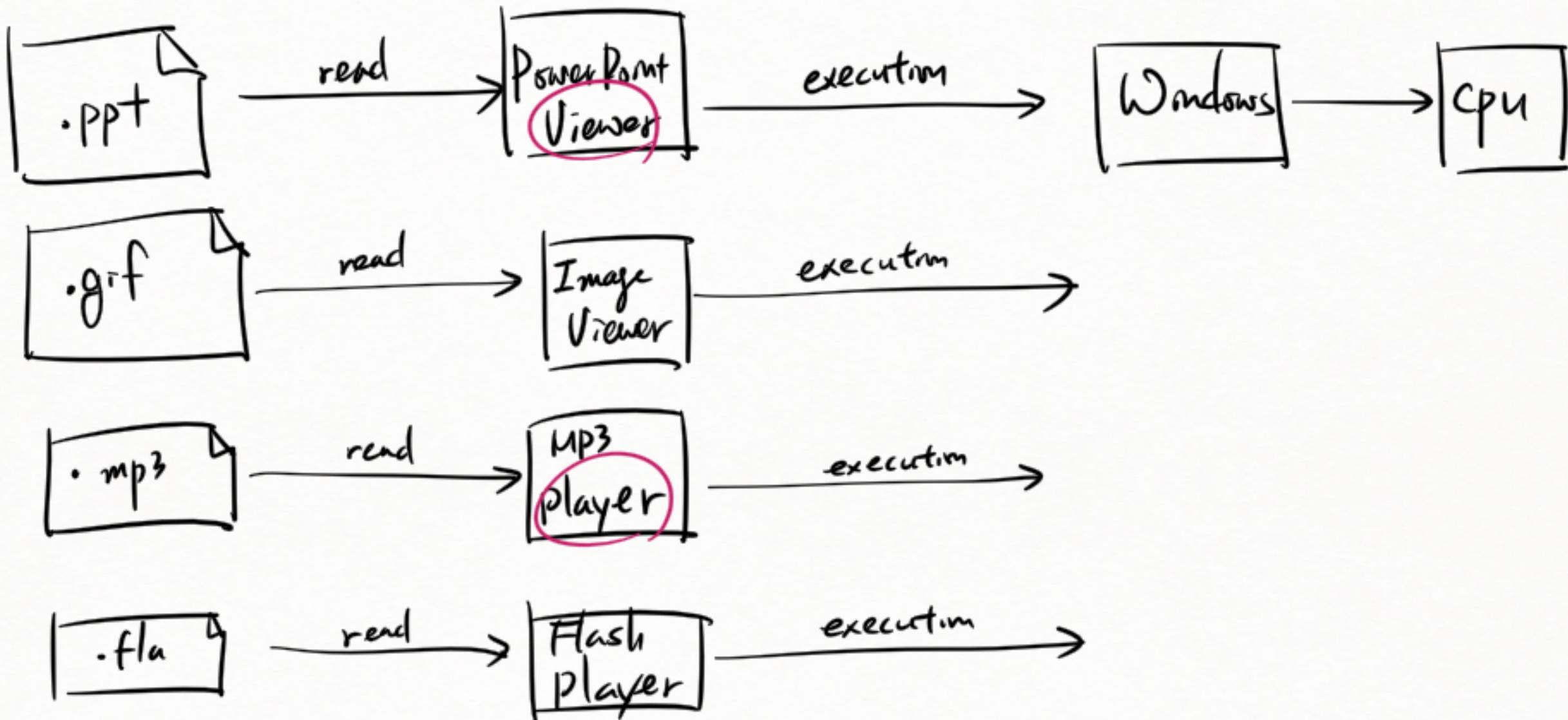
- 실행할 때 기계어 파일은 있으려야 한다.
  - 소스파일 필요없다 → 소스파일을 보호(자산)
  - 실행할 때 컴파일러 불필요.
- } 기계어 바로 실행  
보통 속도 빠르다

인터프리트 방식 :

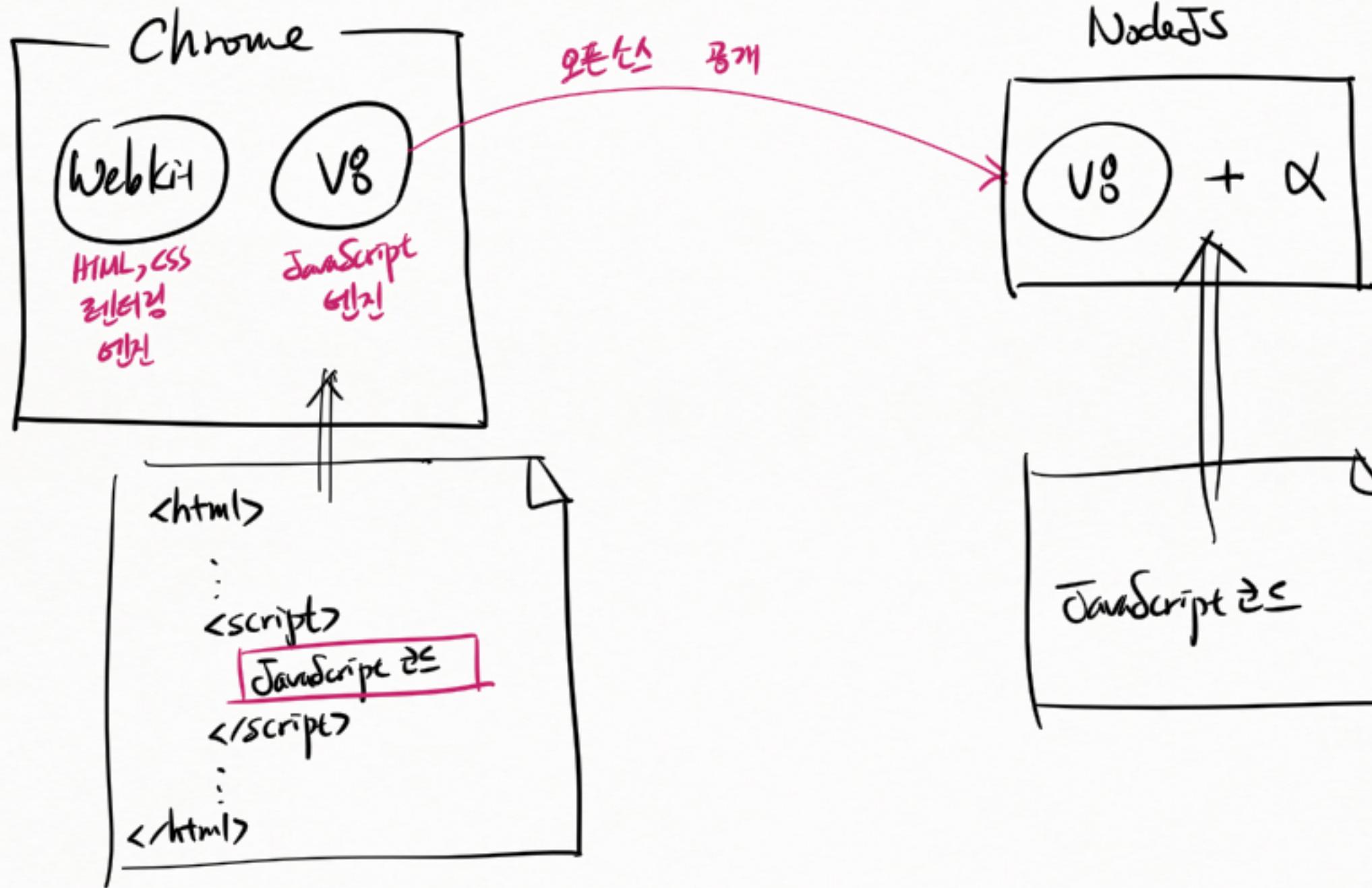


- ↑  
컴파일 하지 않는다.
- 실행할 때 소스 파일 필요  
↓  
소스파일 보호  
↓  
자산으로 보호하기 힘들다.
  - 실행하는데 인터프리터 필요.
- } 매번 명령어를 해석하는 데  
시간 걸리고  
속도 느린다.

\* ~~viewer=player~~ = viewer = player = engine = virtual machine



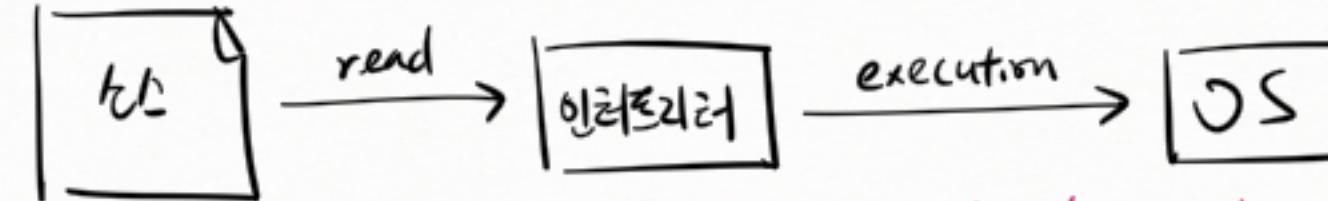
## \* JavaScript (Interpreter) Engine



모든스 공개  
V8  
+ Alpha  
Standalone  
JavaScript  
Engine

## \* JIT Compile 와 AOT Compile

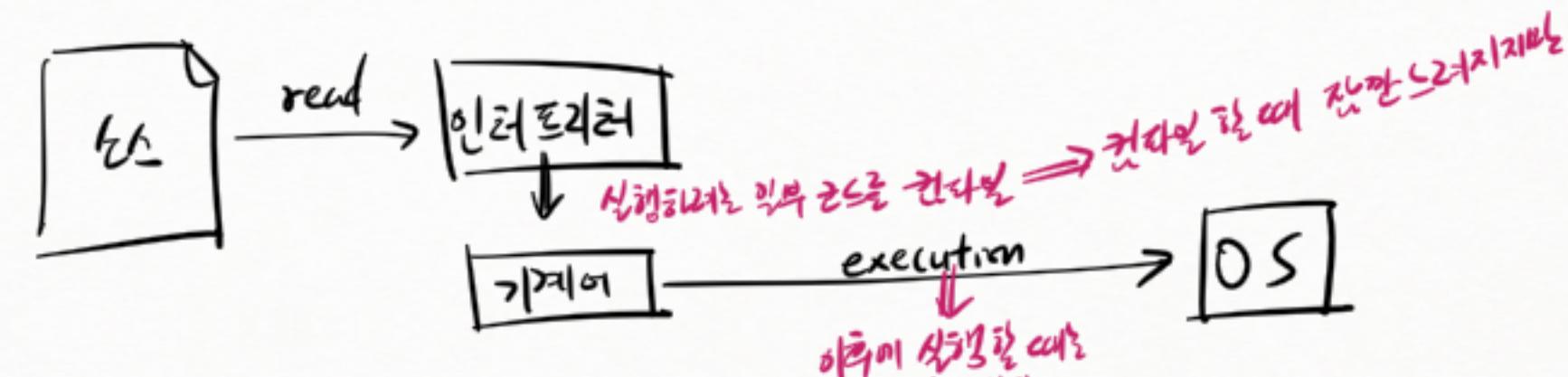
① Plain 인터프리터 :



매번 소스를 검사하고 해석하기 때문에 속도느리다

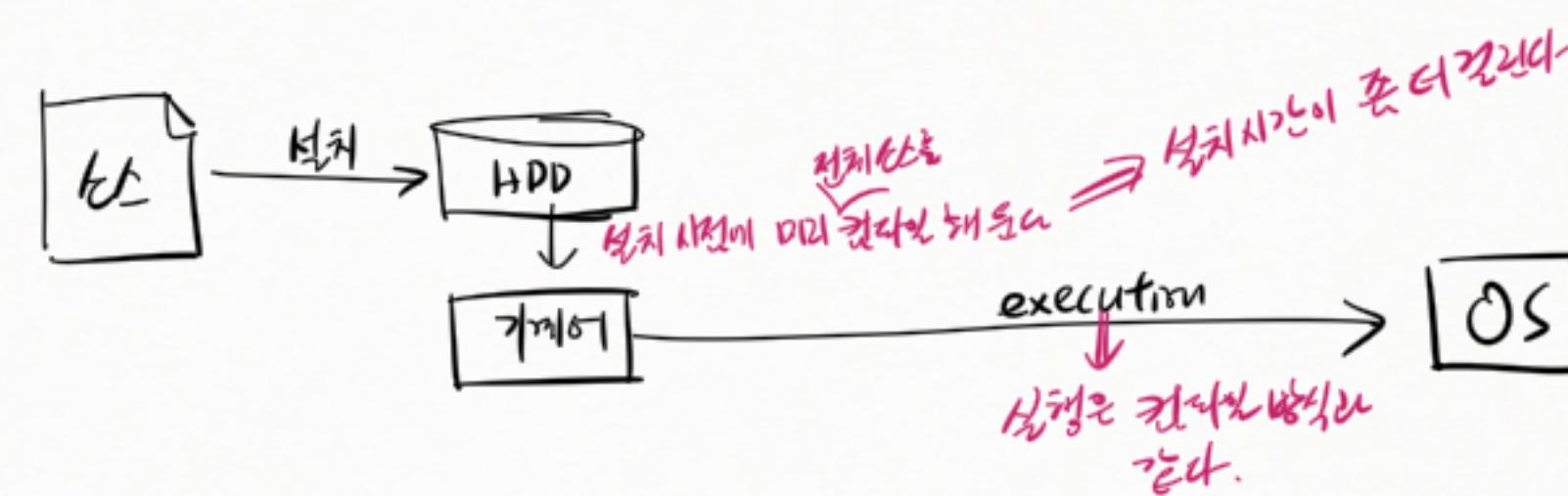
② JIT (Just in Time) 컴파일 :

바로 그 시점에  
직역코드를 컴파일



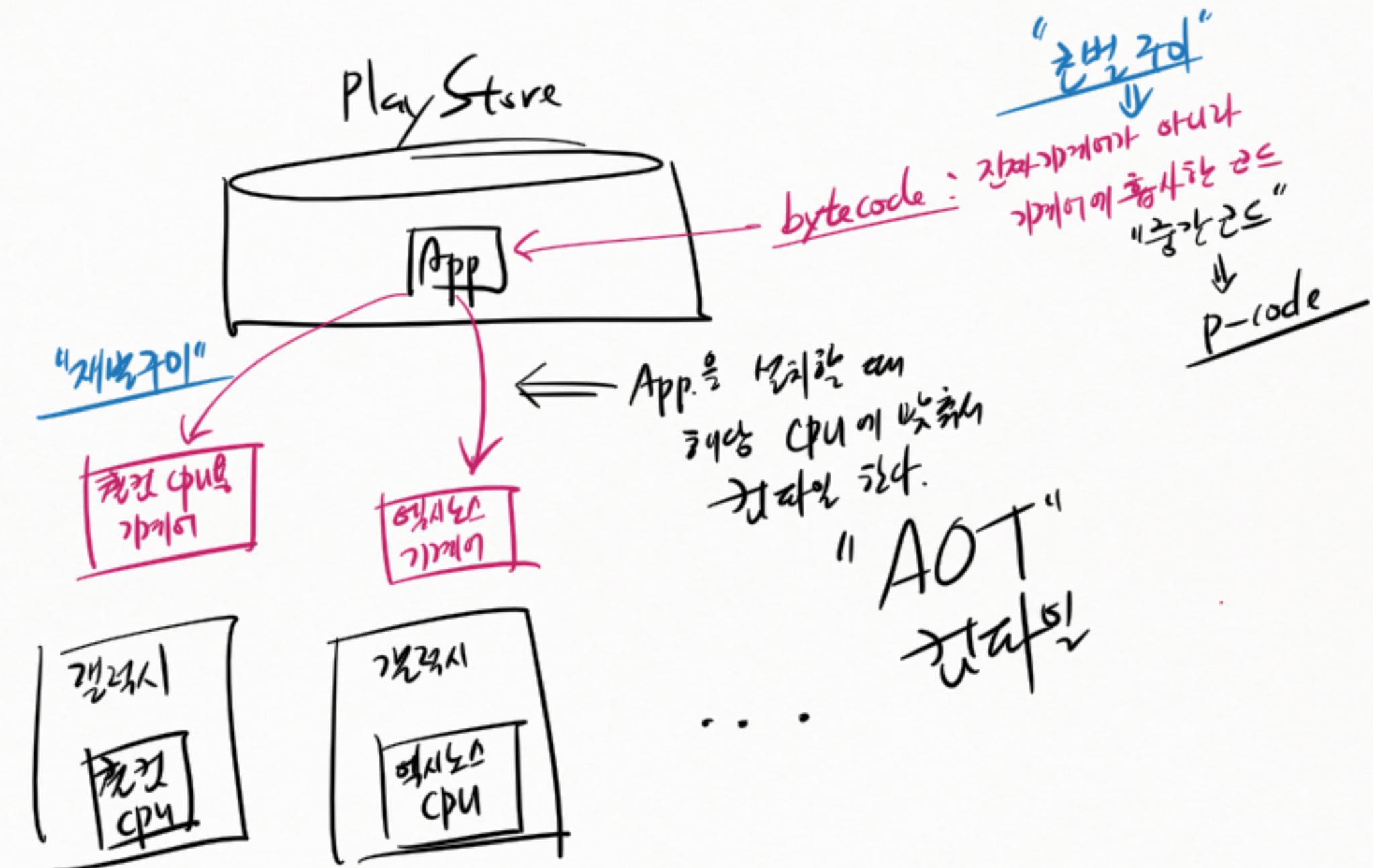
③ AOT (Ahead of Time) 컴파일 :

앞장 시점에 미리  
컴파일(생성)

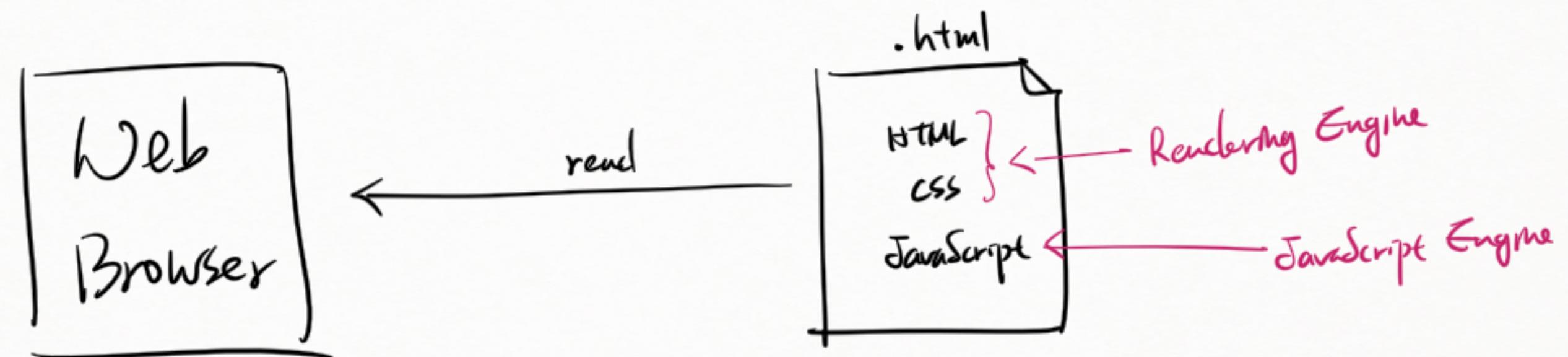


ex) Android  
App

\* Android 와 App , AOT 컴파일



## \* Web Browser et JavaScript



<script>  
자바스크립트 코드  
</script> (명령문)