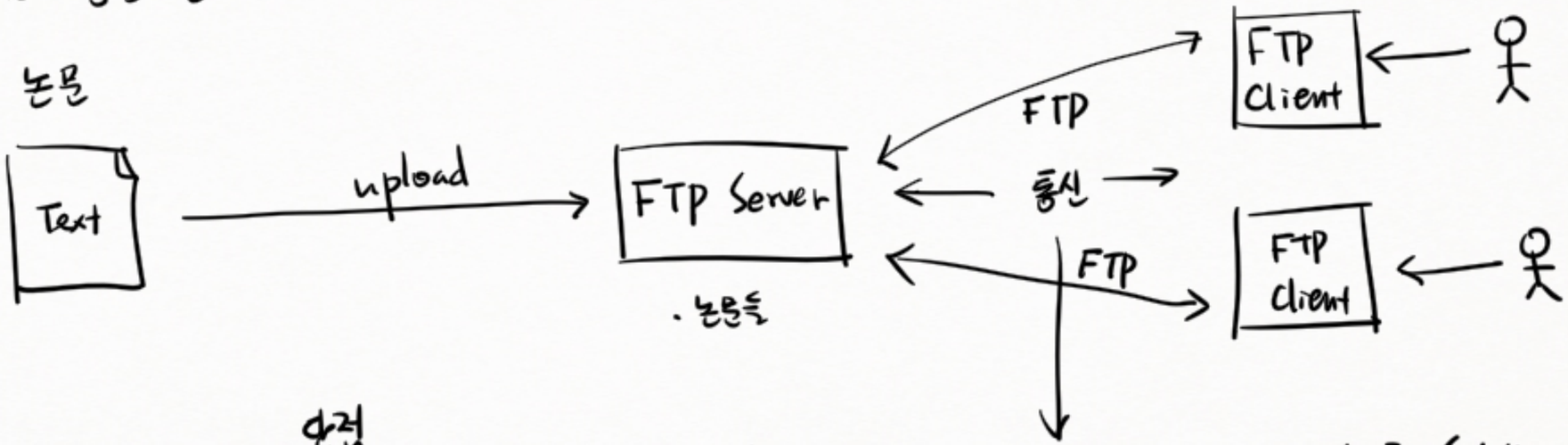
* HTML, HTTP

① HTTP, HTML 등장 전



한계

- 논문이 찾고 하는 다른 논문을 다운로드 받기 번거롭다
- 논문 안에 다른 논문이 업로드된 서버 주소가 없다.

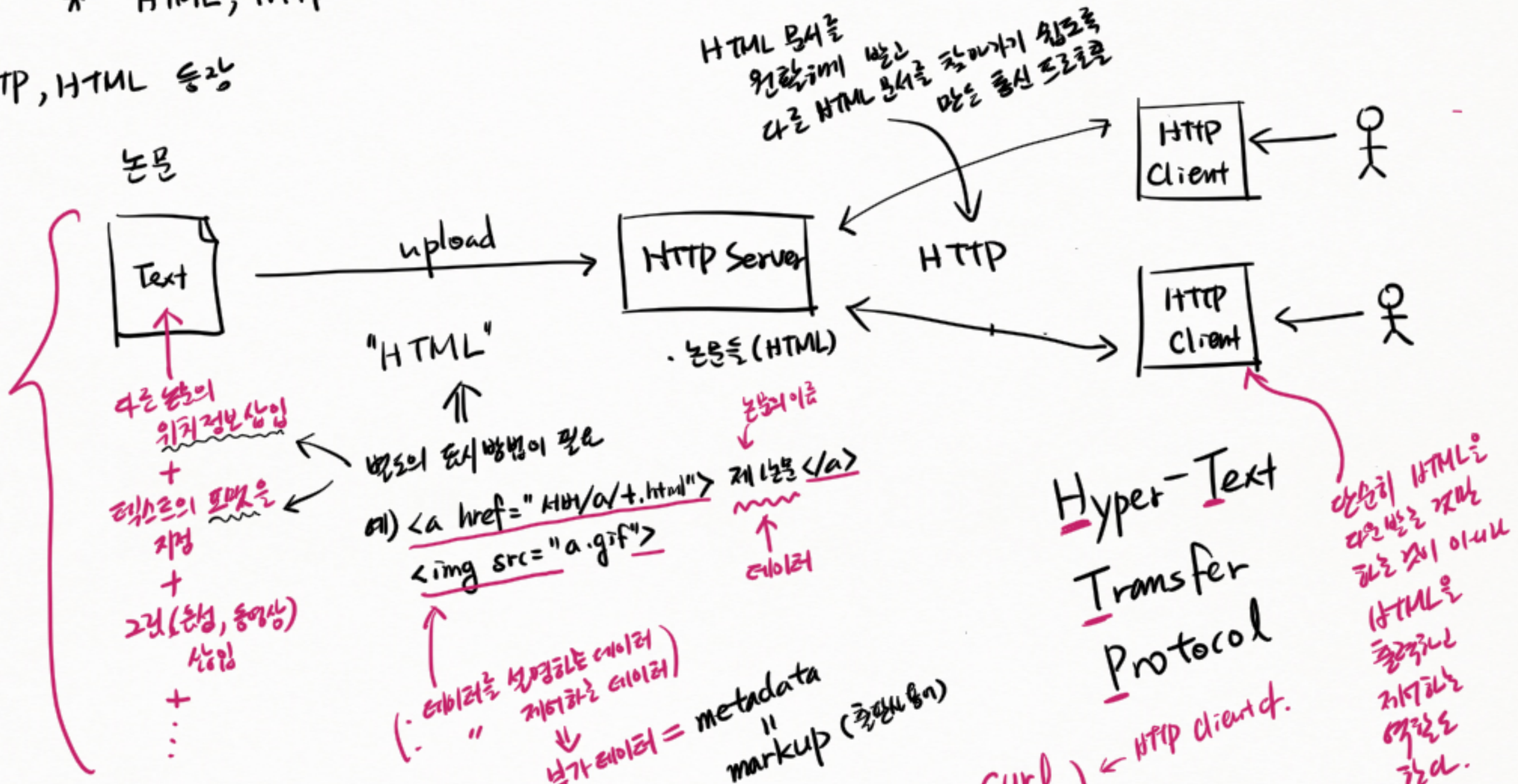
통신 규칙에 따라 Data를 송수신

"protocol"

File Transfer Protocol

* HTML, HTTP

② HTTP, HTML 등장



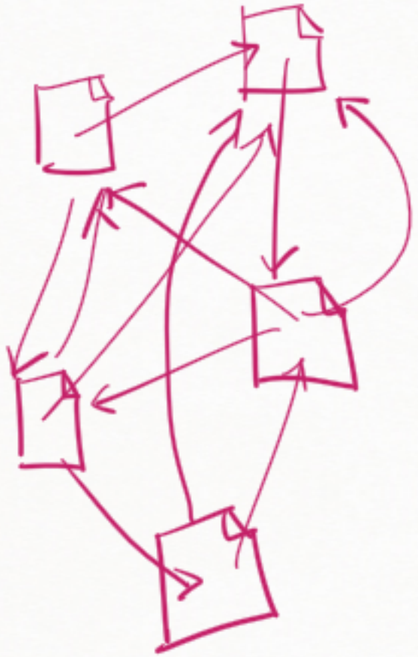
고

Hyper-Text Markup Language

⇓

"HTML"

* HTTP Client / Server App.



"Web Server"

HTTP Server

- Apache HTTP Server
- NginX
- ⋮

① 요청

HTTP

② 응답

HTML 문서를 연결된 모습이 거미줄과 비슷하다.

"Web Browser"

HTTP Client

- Chrome
- Safari
- Firefox
- Edge
- ⋮

(- curl
- wget)

HTTPS

||

HTTP + 보안 (암호화)

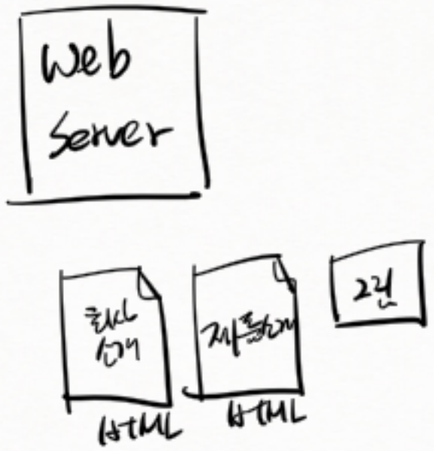
* Web 기술의 활용

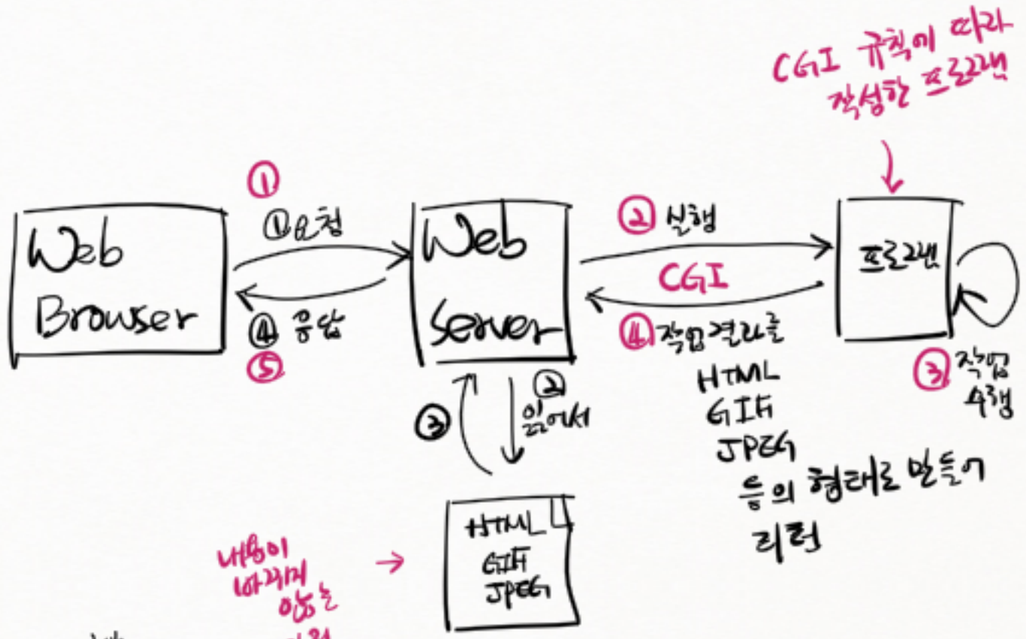
① 1세대 → ② 2세대 —(+ 프로그램)→ ③ 3세대

논문 공유

회사소개
제품소개
⇓
홈페이지

방명록
방문자수
게시판
⋮
⇑
CGI 프로그램





Static resource

* CGI (Common Gateway Interface)
일반
뜻 : Data가 드나드는
규칙

⇓

web server와 프로그램 사이에 데이터를 주고 받는 규칙

⟹ C 언어로 제작 (컴파일 방식)
↓
Perl 스크립트 제작 (인터프리터 방식)

* Web 기술의 활용 II

+ DBMS
+ 스크립트 → ④ 4세대

숫자 형태

+

MS ASP 프로그래밍

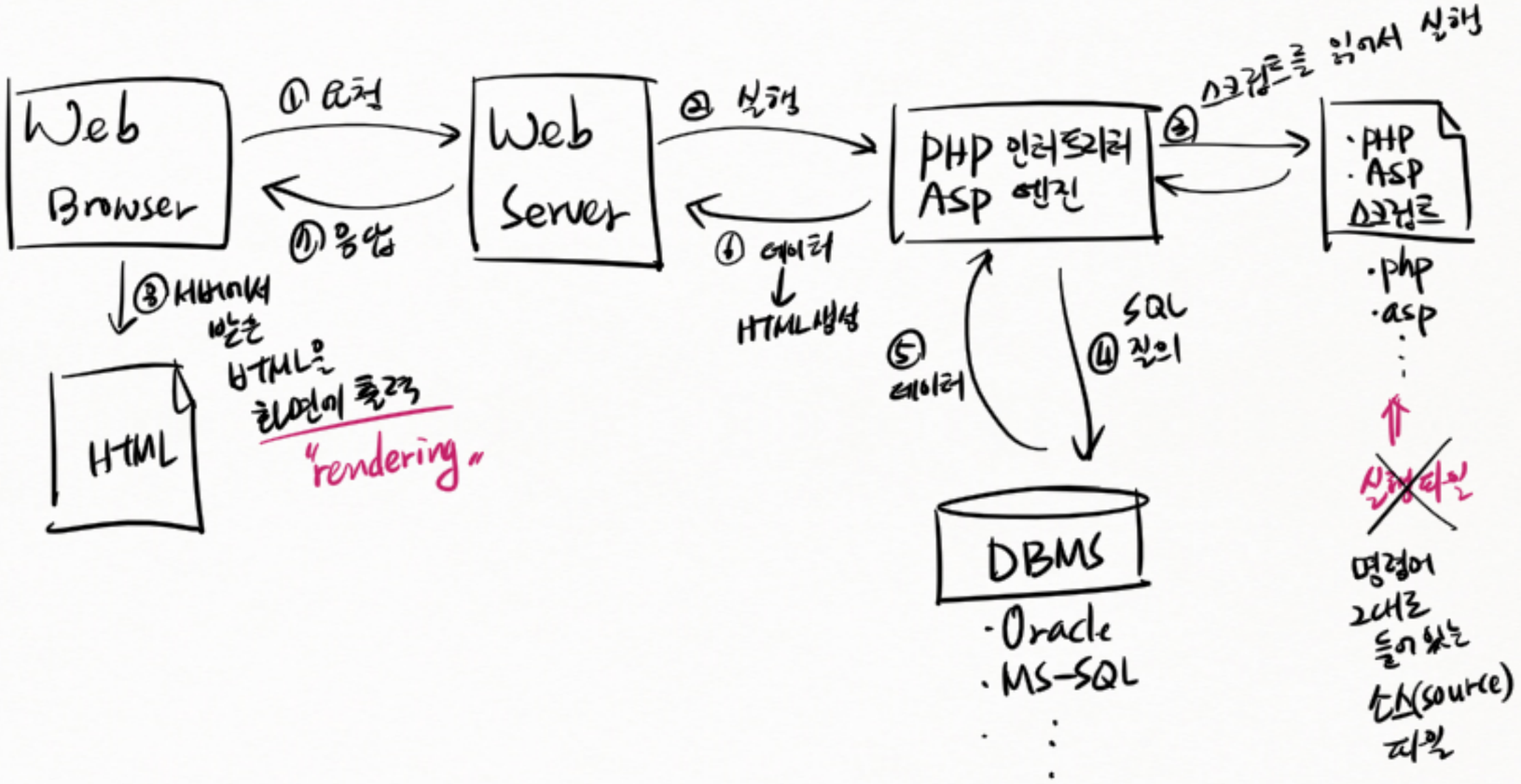
PHP

⋮

+

DBMS

(웹 프로그램)



* Web 기술의 활용 III

연속 프로그램 → ⑤ 5 세대

전문적인 App. 전화

⇓

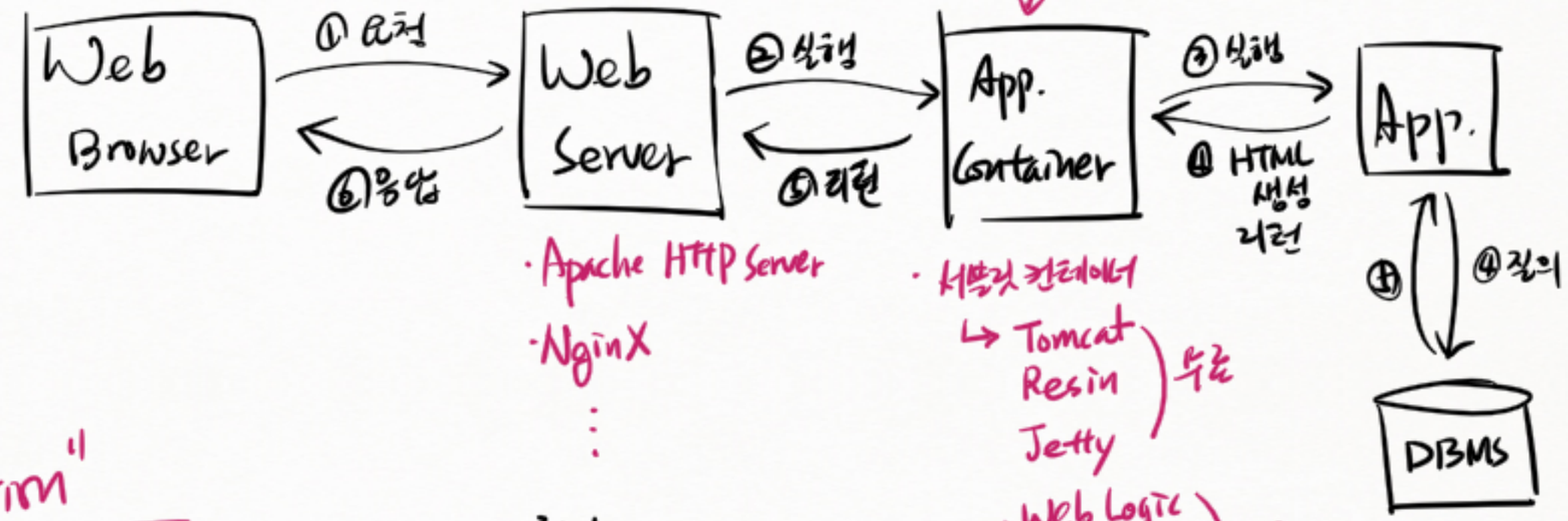
App. 관리 기능 강화

- 인사관리 시스템
- 재고시스템
- 고객관리시스템
- 물류관리시스템
- 웹 메일
- 카페/블로그

⋮

"Web Application"

초창기 논문게시 → 현재 App. 게시



- Apache HTTP Server
- NginX

⋮

- App을 생성 → 실행 → 소멸 } 관리
- DB 연결, 보안 등 공유 자원

- 서블릿 컨테이너
  - ↳ Tomcat, Resin, Jetty } 무료
  - WebLogic, WebSphere, JEUS } 유료

⋮

- NodeJS

⋮

* Web Application 제작 기술

[화면 만드는 기술]

CSS — 콘텐츠의 출력 방법을 제어하는 기술

+

HTML ← 콘텐츠를 구성할 때 사용하는 기술

+

JavaScript ← • 사용자의 액션에 응답
• 콘텐츠를 제어

⋮

"Front-end 개발"

계층 구조로 바라볼 경우

← Full Stack 개발 →

SI/SM

[데이터를 만드는 기술]

✓ Java, JavaScript, PHP, ⋯

✓ SQL

⋮

"Back-end 개발"

HTML

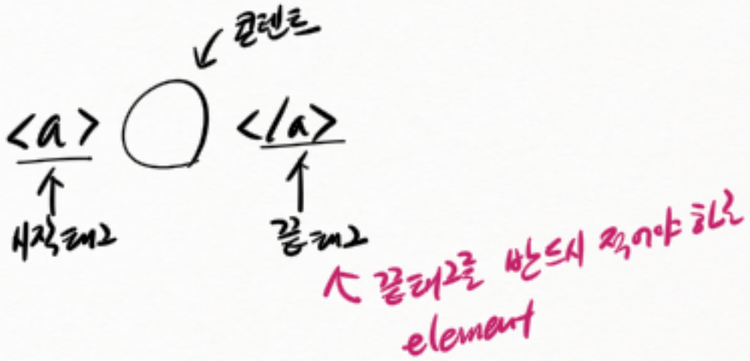
* HTML 기본 구조

root element

↳ 가장 바깥쪽에 있는 element.
문서 당 한 개만 존재해야 한다.

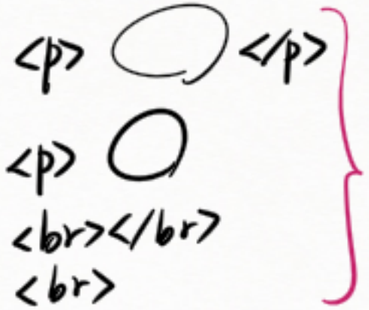
```
<!DOCTYPE html>
<html>
  <head>
    ⋮
  </head>
  <body>
    ⋮
  </body>
</html>
```

- `<html>` ← 시작 태그
- `</html>` ← 끝 태그
- `<head>` 부모 element
- `<body>` html의 자식 element
- `<head>` ~ `</body>` : 콘텐트(content) = 자식 element들
- `<html>` ~ `</html>` : element



☆ `<br></br>` = `<br/>` = `<br>`

- `<br/>` ← 끝태그 생략을 표시
- `<br>` ← 생략 표시 안함



끝태그를 생략해도 되는 element

`<meta>` ← 끝태그를 적지 않아야 하는 태그

* HTML

HTML 3.x → HTML 4.x → XHTML → HTML5

photoshop
⇓
그림, 이미지

웹디자이너

+ HTML, CSS
JavaScript
⇓
간단한 수준
단순형 퍼블리셔
퍼블리셔
(publisher)

⇒ HTML, CSS, JavaScrip
+
UI Framework
(VueJS, Angular, React)
⇓
Client App UI

Front-end 개발자

끝태그 필수

왜?

HTML 문서를 분석하기 쉽다
⇓
문서의 실제 내용을 찾아 추출하기 쉽다.

⇓

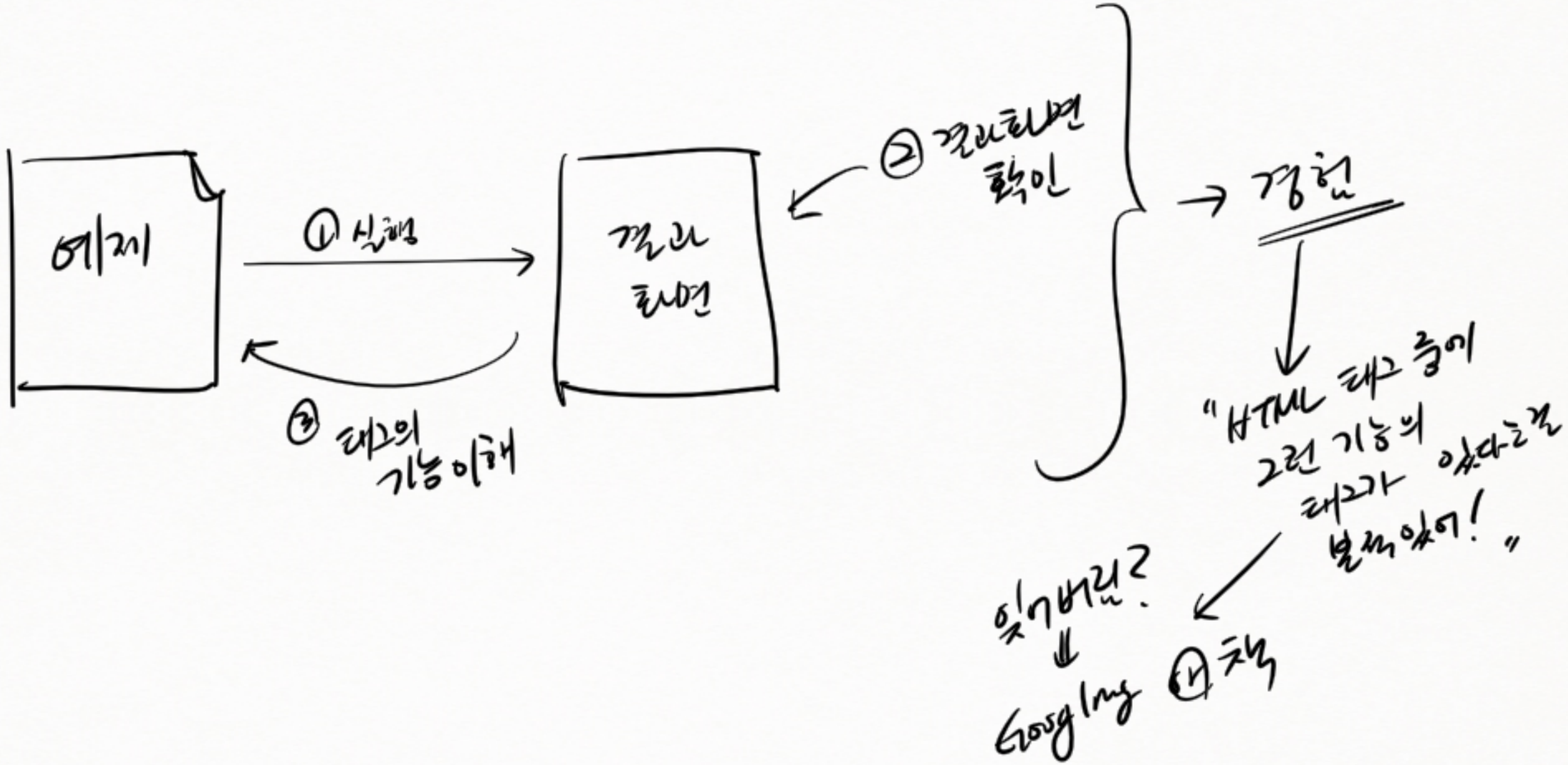
개발자가 번거로워함
⇓
인기 ↓
⇓
폭망

· HTML 4의 자유로운 허용
끝태그 생략

· Semantic 태그 추가
(콘텐츠의 의미를 나타내는 태그)
⇓
<section>
<article>
<aside>
<header>
<footer>
⋮

→ 이전에는 보통 태그로 통칭

* HTML 공부법


예제
① 실행
결과 화면
② 결과화면 확인
③ 태그의 기능 이해
경험
"HTML 태그 중에 그런 기능의 태그가 있다고 알고 있어!"
잊어버림?
Googling ④ 찾기

* HTML4 의 DOCTYPE

문서 규칙의 식별자

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

root element

문서의 규칙의 공개여부

⇕

SYSTEM

+ 공인
- 사설

규칙을 만든 조직의 이름

문서규칙의 이름

↓

?

태그를 사용하는 방법
· 이런 태그가 있으니
이렇게 사용하라
⋮

문서규칙의 언어

문서규칙을 정의한 DTD 파일의 URL

Document Type Definition

I18N Internationalization
18

L10N Localization
10

☆ DTD ← old

⇓

Schema ← new

* tag 와 attribute

attribute

value

콘텐트

<a href="_____">○</a>

시작태그

끝태그

controls
selected
checked
readonly
⋮

속성중에 값에 상관없이 존재 유무로 기능을 수행하는 속성이 있다

* CSS

① <style> 태그

```
<style>
  ≡
</style>
```

② inline 스타일

CSS 코드

```
<tfoot style="——">0</tfoot>
```

inline style

최 위선 !

③ 외부 CSS 파일

a.html

```
<html>
  <head>
    <link href="x.css">
    ⋮
```

여러 HTML 페이지에서 CSS를 공유할 수 있다.

x.css

```
h1 {
  ≡
}
⋮
```

* MIME Type

Multi-purpose
Internet
Mail
Extensions

} 메일의 첨부파일 종류가 어떤 형식인지 상대편에게 알려주기 위함

⇓

현재
메일뿐만아니라 web 등 여러곳에서 사용되고 있다.

HTTP

* 네트워킹

Networking

- Connection-Oriented (연결지향)
  - 연결형 통신
  - 예) 전화
  - TCP : 데이터 전송 신뢰성 확보
    - ↳ HTTP 1, 2

192.168.0.41 ⇄ 192.0.4.5

- Connectionless (비연결)
  - 연결없이 데이터전송
  - 예) 편지, 방송
  - UDP : 데이터 전송 신뢰성 X → 별도의 처리 필요
    - ↳ HTTP 3

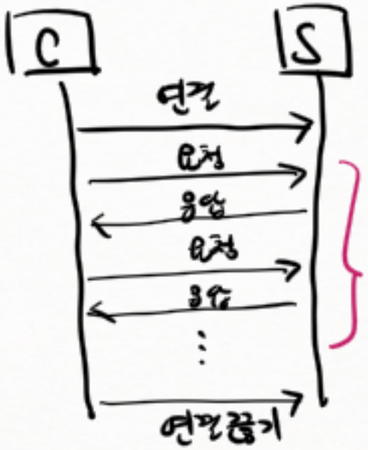
편지: 192.168.0.41 —Data→ 192.0.4.5

방송: 192.168.0.41 —Data→ 192.0.4.0/24

* 네트워킹

Networking → Connection-Oriented

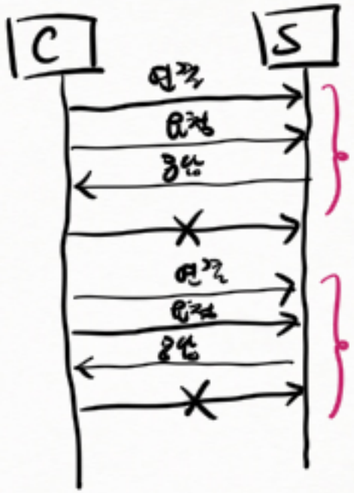
- Stateful ⇒
  - ↓ 적은 수의 client에 대응



한 번 연결한 후 연결을 끊을 때까지 여러번 통신

예) · ssh (telnet + secure)
· ftp
· 채팅
· 구글 미트

- Stateless ⇒
  - ↓ 많은 수의 client에 대응할 수 있다



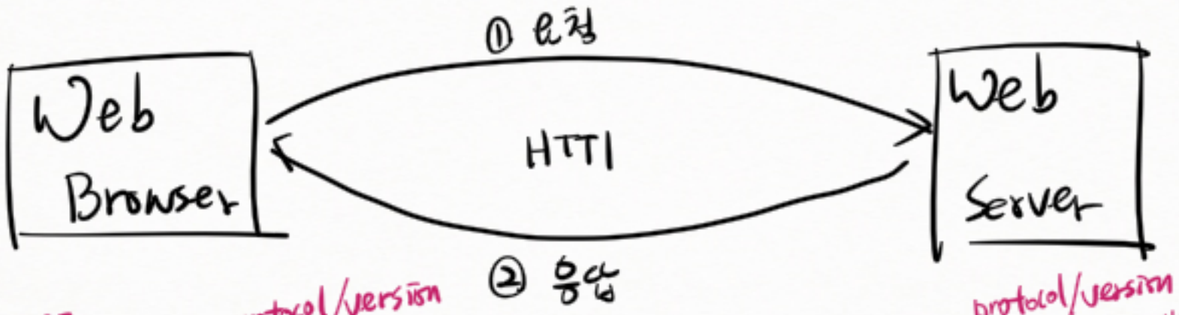
한번 연결에 한번 통신

매번 연결

〃

예) HTTP

* HTTP 요청 / 응답



★ URI (Uniform Resource Identifier)

URL (— Locator)
↳ http://서버:포트/○/○/○

URN (— Name)
↳ ○:○:○

method / URI / protocol/version

request line → GET / — /a.html HTTP/1.1 ↵

header (요청에 대한 보충 정보)
① 요청 헤더
② 공통 헤더
③ 엔티티 헤더

Host: localhost:5500 ↵
User-Agent: — ↵
⋮
↵

POST 메서드일 때 → 보내는 데이터

Message-body

protocol/version / 상태 코드 값 / 상태에 대한 설명

HTTP/1.1 200 OK ↵ ← status line

header (응답에 대한 부연설명)
① 응답 헤더
② 공통 헤더
③ 엔티티 헤더

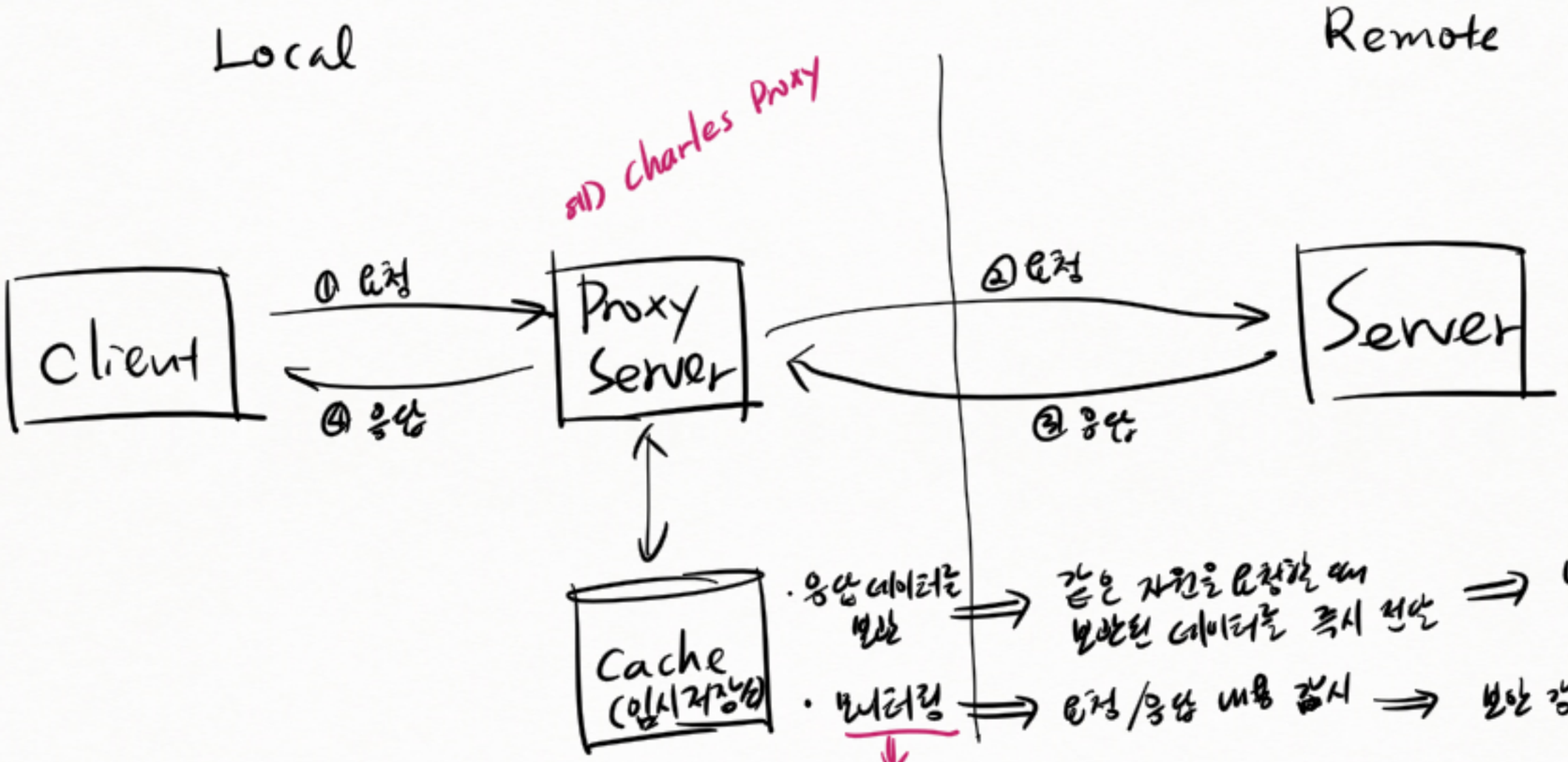
Content-Length: ○ (byte) ↵
Content-Type : text/html; charset=UTF-8 ↵
⋮
↵

<!DOCTYPE HTML>
<HTML>
⋮
</HTML>

Message-body

* Proxy 서버

↖ client 와 Server 중간에서 통신을 중재



· 응답 데이터를 보관 ⟹ 같은 자원을 요청할 때 보관된 데이터를 즉시 전달 ⟹ 네트워크 오버헤드를 줄인다. ⇓ 응답속도 개선

· 모니터링 ⟹ 요청/응답 내용 감시 ⟹ 보안 강화

↓ HTTP 통신을 살펴볼 수 있다.

* GET 요청

파라미터명 / 파라미터값 / 파라미터와 파라미터를 구분하는 구분자

GET /html/form/exam02?name=aaa&age=11 HTTP/1.1 ↵

path (자원의 주소)

"Query String" (서버에 보내는 데이터)

data를 URL에 붙여서 보낸다

☆ 한계

① 바이너리 데이터를 보낼수 없다

↓

binary 데이터를 텍스트로 인코딩 하면 가능

② URI 크기의 제한 때문에 대용량 데이터를 보낼수 없다

③ URL → 브라우저 캐시에 보관

↳ 보안이 안된다

Base64

binary 데이터 ⇄ Text 데이터 (부호화 / 디코딩)

예) Image

☆ text vs binary

① text

일반 텍스트 편집기로 편집 가능한 포맷

예) .txt, .rtf, .html, .css, .js, .xml, .json, .properties, .java, .py, ⋮

② binary

· byte 단위로 포맷

· 일반 텍스트로 편집 불가

· 전용 App 사용

포맷이 깨진다 →

예) .jpeg, .gif, .png, .mp3, .mp4, .avi, .doc

* POST 요청

Query String 없다.

POST /html/form/examo2 HTTP/1.1 ↵ ← request line

Host : 192.168.0.29:5500 ↵
⋮
Content-Length : 15 ↵
Content-Type : application/x-www-form-urlencoded ↵
⋮
↵ ← header

POST 요청일 때만 사용하는 헤더 →

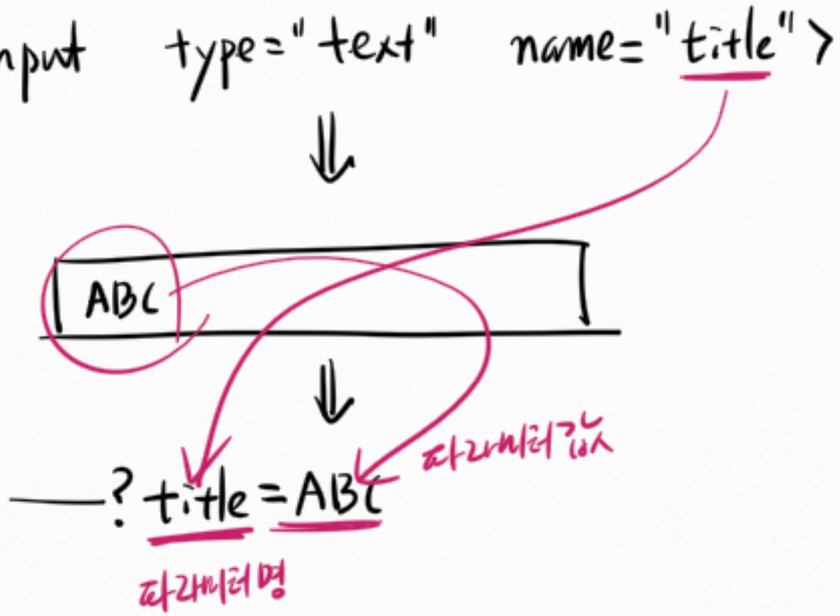
POST 요청은 message-body 부분에 데이터를 보낸다 →

name=aaa&age=22 ← message-body

* 특징

① Binary 데이터 전송 가능
② URL에 노출안됨 → URL에 데이터를 포함해야 하는 경우에 적합하지 않다.
→ 예) 검색 URL, 게시글 조회 URL
③ 여러 개의 파일 첨부 가능
④ 용량 제한 없다.

* name 속성



* name 속성 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
↓
스타일을 적용할 대상을 가리킨다.

pseudo selector
(대상의 상태)

```
.content header > span.title:hover {
    background-color: red;
    color: white;
}
```

style name

style value

⇒ 셀렉터 { 스타일: 값; 스타일: 값; ... }

① selector 사용법

② specificity (스타일 적용순서)

③ box 다루는 방법
(테두리, 여백, 박스 계산법)

④ 폰트 다루는 방법

⑤ 색상 지정하는 방법

⑥ 배경 다루는 방법

⑦ block 과 inline 다루는 방법

⑧ 위치 조정하는 방법

↓

* Selector

* { — }	* { — }
태그명 { — }	body { — }
태그명, 태그명, 태그명 { — }	img, ul { — }
.그룹명 { — } (class)	.c1, .c2 { — }
#태그아이디 { — }	#content { — }
—[속성명=값] { — }	
—:상태명 { — } (pseudo selector)	li:hover { — }

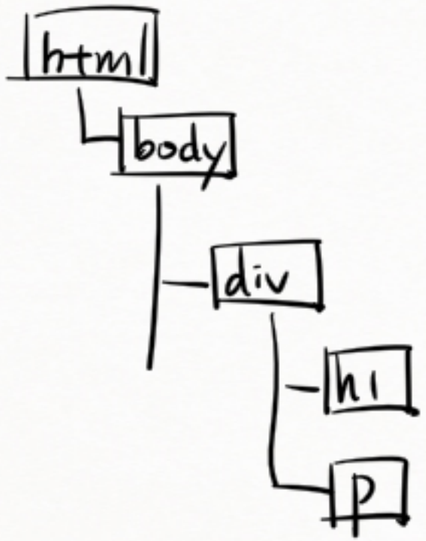
조상 자손 { — }	#content li { — }
부모 > 자식 { — }	ul > li { — }
태그 + 다음태그 { — }	img + ul { — }

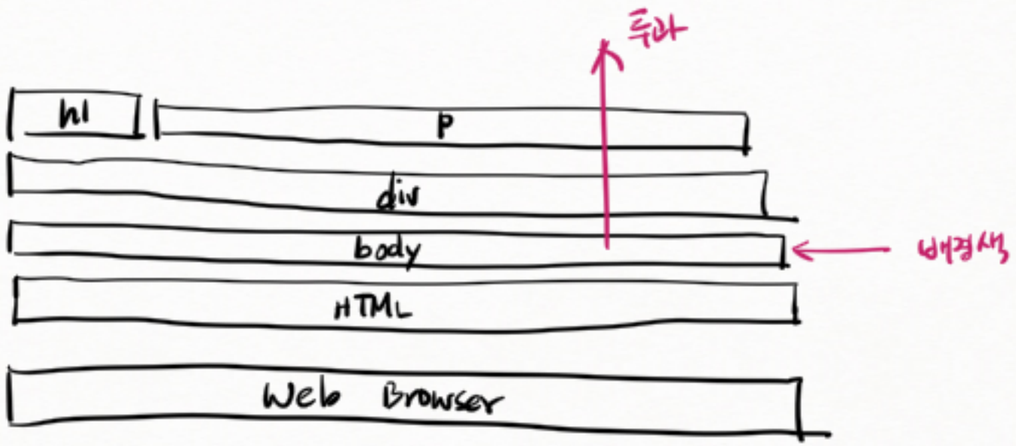
대상과 조건 조건 … { — }

div#menu li.c2 { — }

input[type="text"] { — }

* 부모 태그와 자식 태그의 계층 구조



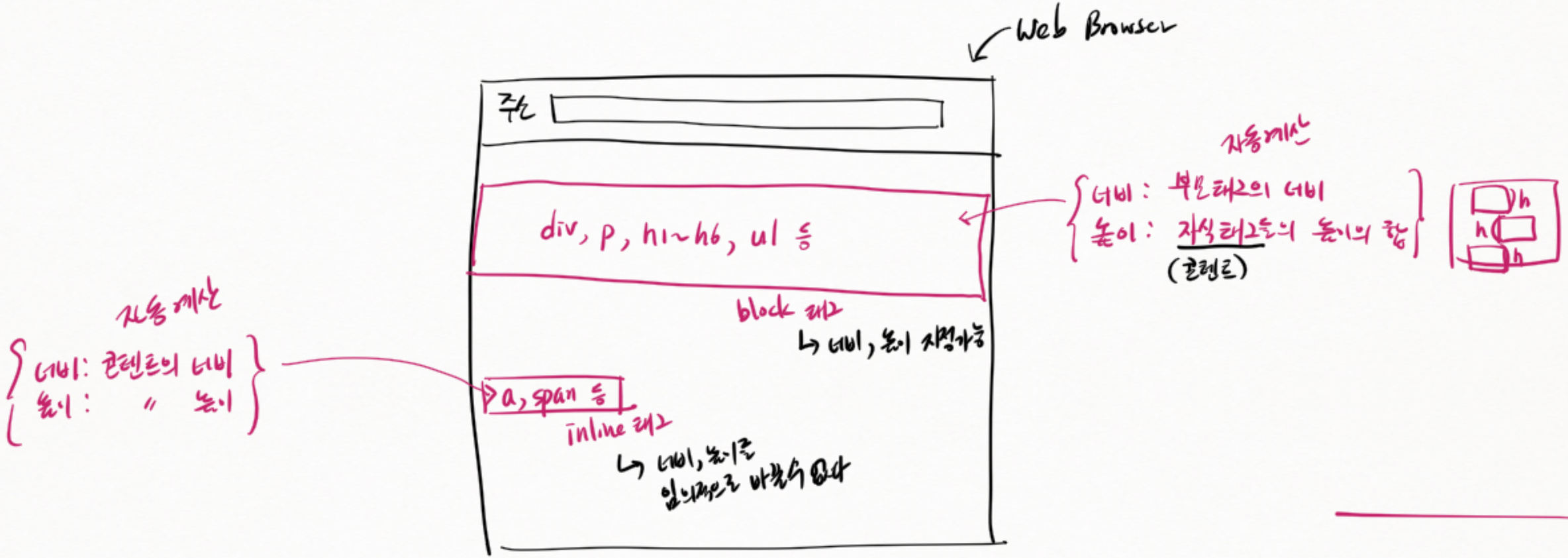


* CSS specificity : 스타일 적용 순서 낮 → 높

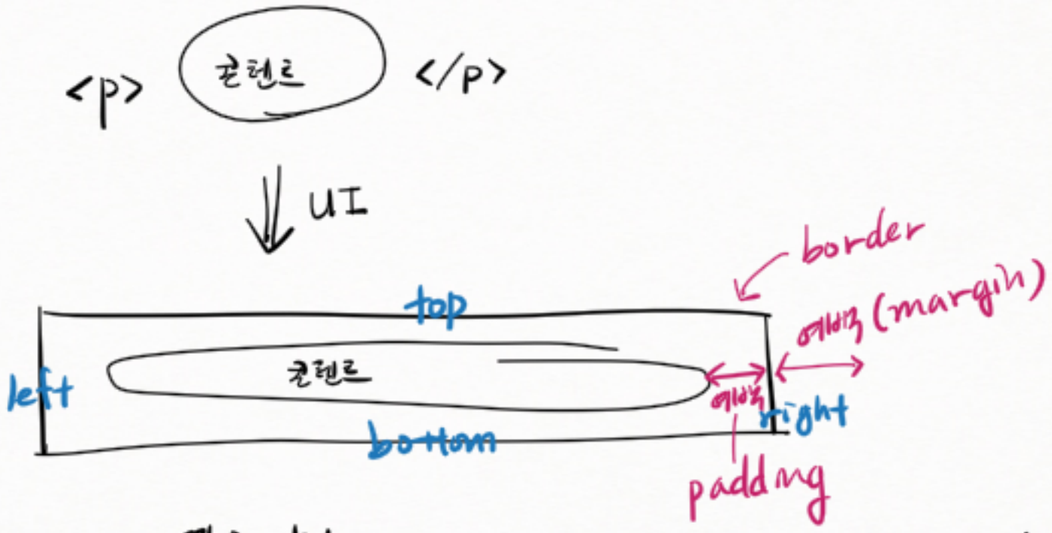
* → 0	* { — }		
태그, pseudo selector → 1	h1 { — }	a:hover { — }	
클래스, 조건 → 10	.cl { — }	input[readonly] { — }	
아이디 → 100	#menu { — }		
inline 스타일 → 1000	<a href="—"	style="—" > ( ) </a>	

인라인 스타일

* 태그의 너비, 높이


Web Browser
주소
div, p, h1~h6, ul 등
block 태그
↳ 너비, 높이 지정가능
a, span 등
inline 태그
↳ 너비, 높이를 임의대로 바꿀수 없다
자동계산
너비: 부모태그의 너비
높이: 자식태그들의 높이의 합
(콘텐츠)
자동계산
너비: 콘텐츠의 너비
높이: 〃 높이

* 테두리 (border)



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

⟹ 짧은 명령

border - top : color style width ;
bottom
right
left

⟹ 더 짧은 명령

border : color style width ;
↓
테두리 모든 방향

border - style : solid ;
↓
모든 방향의 테두리 스타일

* Raster 폰트 Vector 폰트

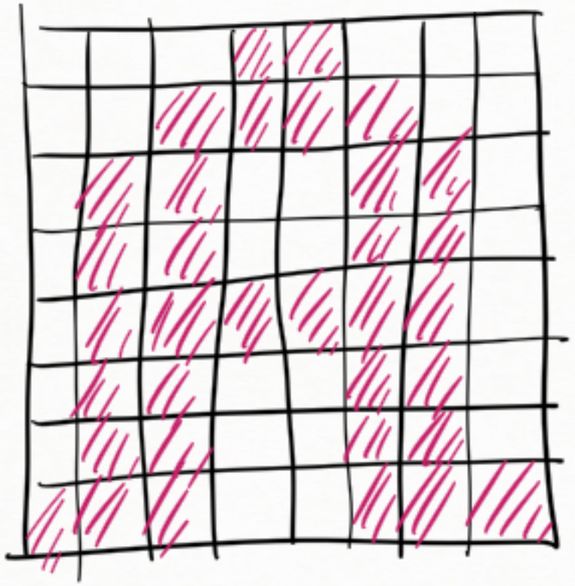
↗ 비트맵 이미지 예) .bmp, .gif, .png, .jpg ...

Raster 폰트 ⟶ Vector 폰트

↗ 벡터이미지 예) 츠컴아트, 캐드

↳ 픽셀단위로 글자를 만든다

- "출력 속도가 빠르다"
- 폰트크기에 따라 각 문자를 만든다
  ⇓
- 정해진 크기보다 더 크게 출력하면 단순히 각 픽셀의 크기를 늘리기 때문에 계단현상 발생
- 이미지의 복잡도와 상관없이 파일크기는 같다. (해상도)



예) courier

☆ 폰트크기 = 높이

4~5 line
3~6 line
(3,3) (1,8) line
⋮

예) True-Type 폰트
courier new

→ 글자를 그리는 명령어를 작성한다
- 글자를 그리는 명령을 수행
  ↓
  "출력 속도가 느리다"
  (용량은 CPU가 빠르기 때문에 상관하지 않는다)
- 글자 크기를 늘리더라도 명령을 통해 그리기 때문에
  ⇓
  계단현상이 발생하지 않는다.
- 이미지 단순 → 파일크기 ↓
  〃 복잡 → 〃 ↑

* document 요약

document

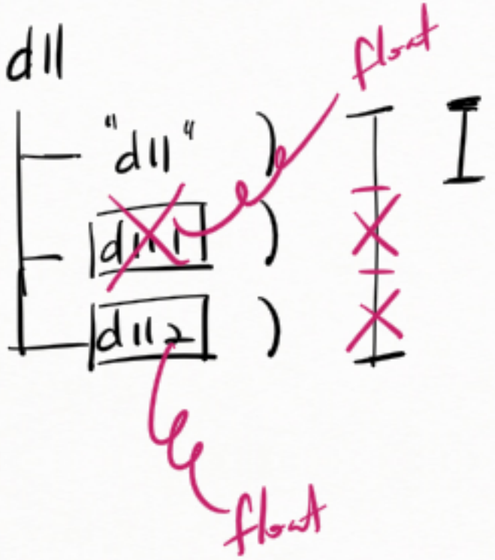
↓

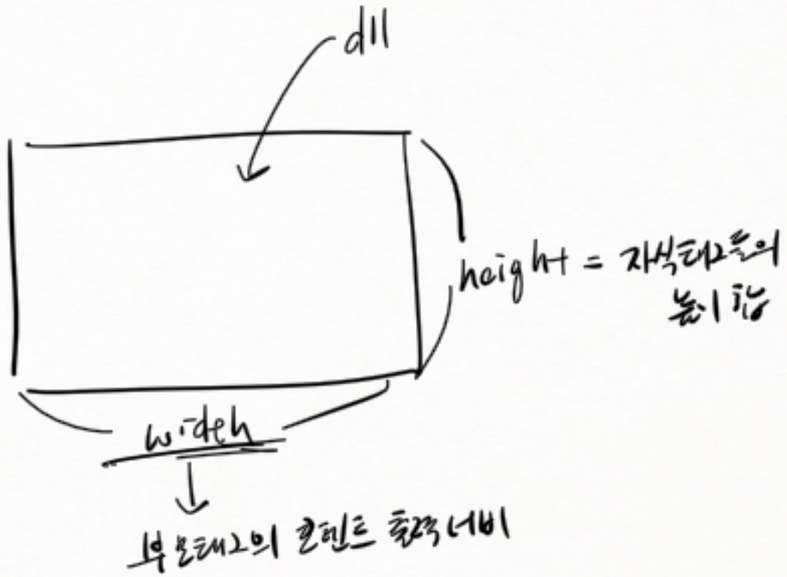
태그를 다루는 기능이 있다

- getElementById(태그 아이디) → 리턴 → 태그 객체
- getElementsByTagName(태그이름) → 리턴 → 태그 목록
- createElement(태그이름) → 리턴 → 태그객체를 생성해서 리턴
- ⋮
- setAttribute(속성명, 속성값) : 태그의 속성 값을 설정

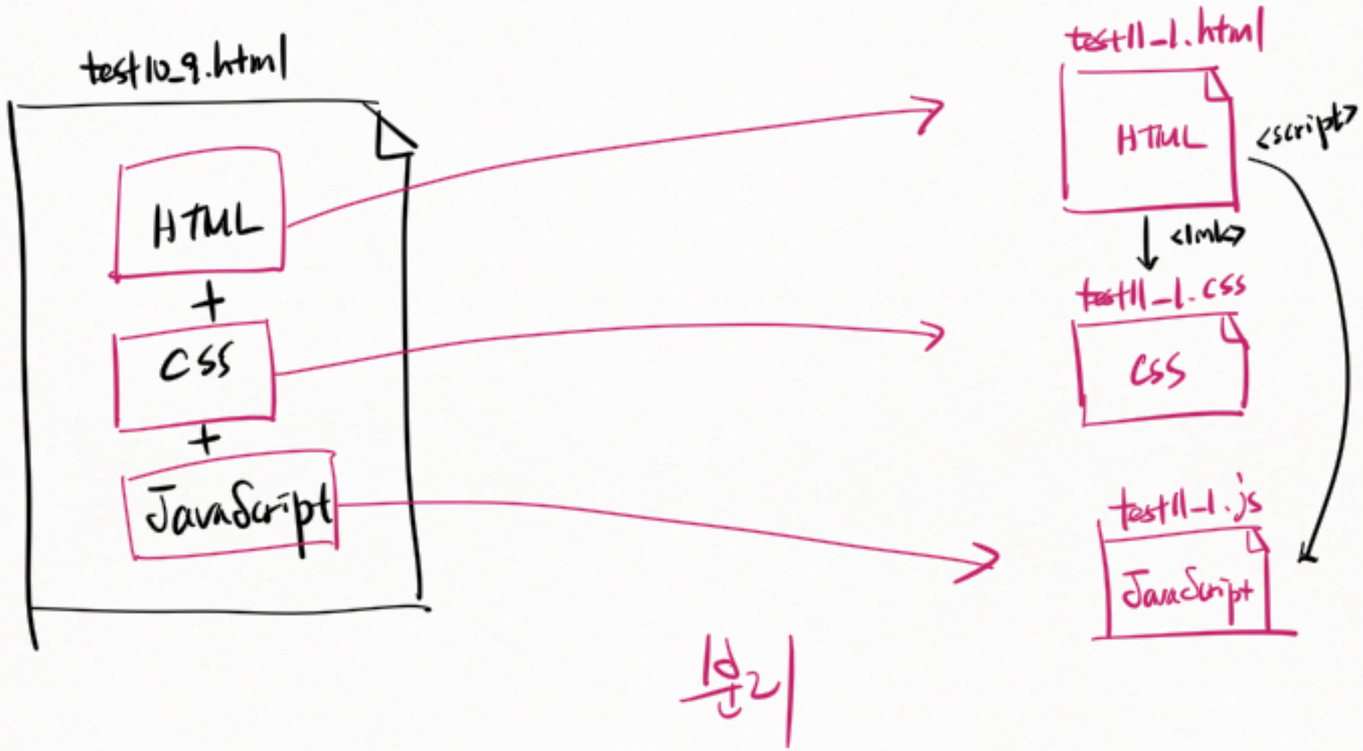
* float 과 absolute

↳ 부모태그의 크기를 계산할 때 float과 absolute 태그는 제외한다.





* CSS 나 JavaScript 코드의 분리



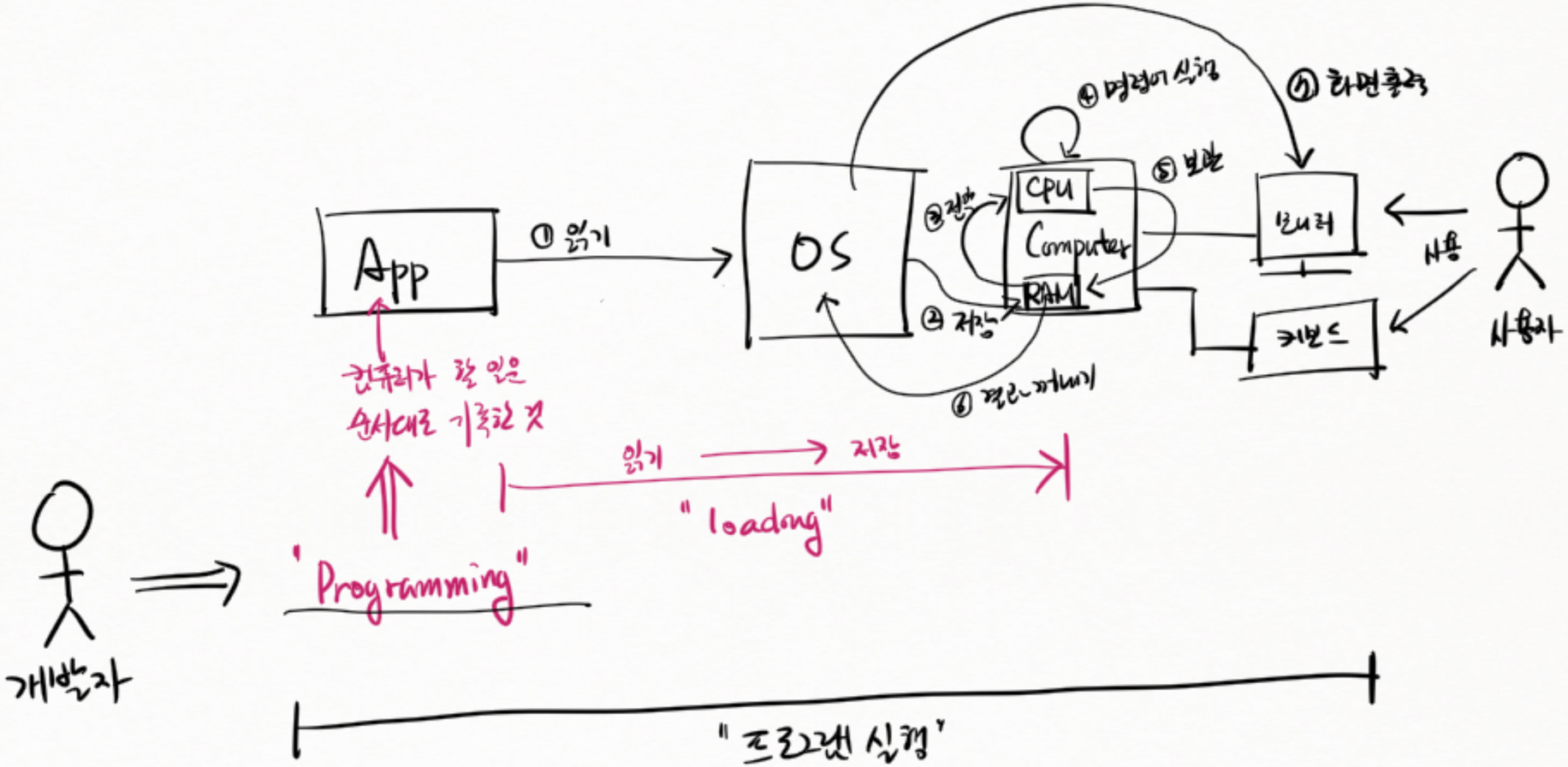
분리의 이점

① 가독성이 좋아진다

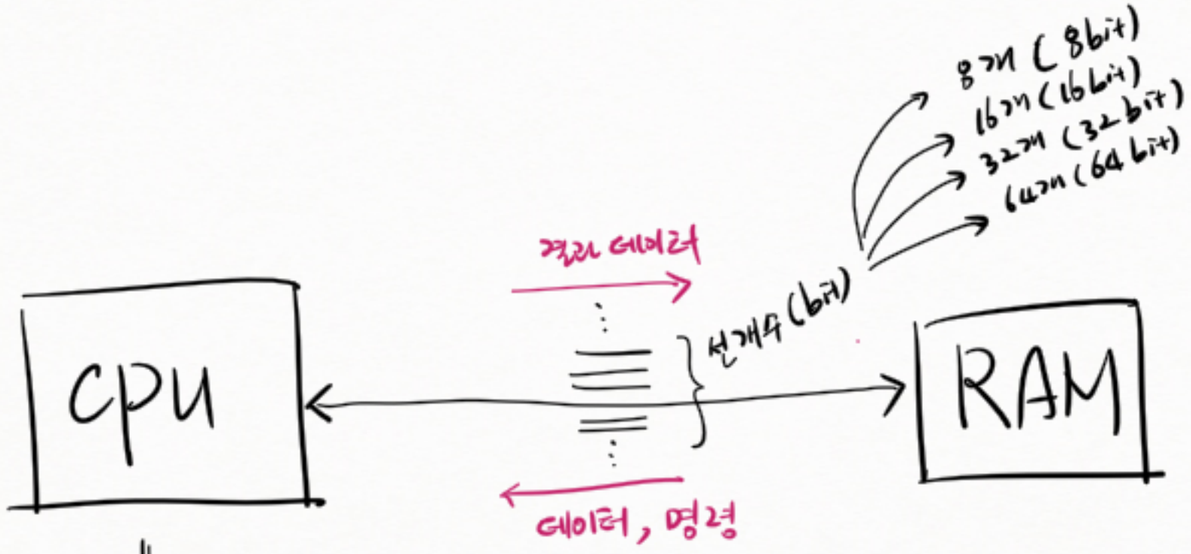
② CSS 나 JavaScript 코드를 다른 페이지에서 재사용하기가 쉽다.

JavaScript

* 프로그램 실행과 프로그래밍



* CPU 와 RAM , bit



CPU 제조사는
CPU를 통해 명령을 실행시키는
방법을 명세서로 제공한다

(신호 보내는 비트 규칙)

명령	코드
+	10110 ····001
−	10101 ···000
*	11001 ···010
/	00110 ···011
OR	00000 ···111
⋮	

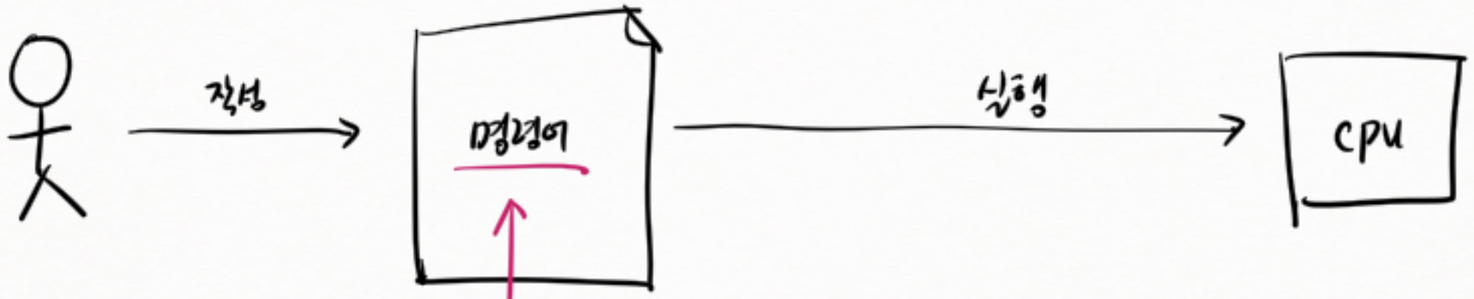
instructions set
(명령어 셋)

- CPU 버전↑ ⇒ 명령 규칙이 추가
- CPU 마다 명령이 다름 : i제로 vs 제온제로
- CPU 제조사 마다 명령이 다름
  ↳ 전기신호를 보내는 규칙

1 1 1 1 0 1 0 1
0 0 0 0 1 0 1 0 +
1 1 1 0 0 0 1 1 −
1 1 0 0 0 0 1 0 *

* 명령어 작성 : 기계어

① CPU instructions set 명세서를 보고 직접 명령어 작성



기계어
=
CPU instructions set

비트를 직접 작성

```
{ 001100111 ... 001
  110101110 ... 110
  ⋮                 }
```

★ 특징

① CPU bit가 적은 86과 시절에는 개발자가 직접 작성하기도 했다

② 보통 2진수를 짧게 표현한 16진수를 사용해서 작성한다

③ 인간의 숫자와 거리가 멀기 때문에 작성하기 매우 불편하고 힘들다.

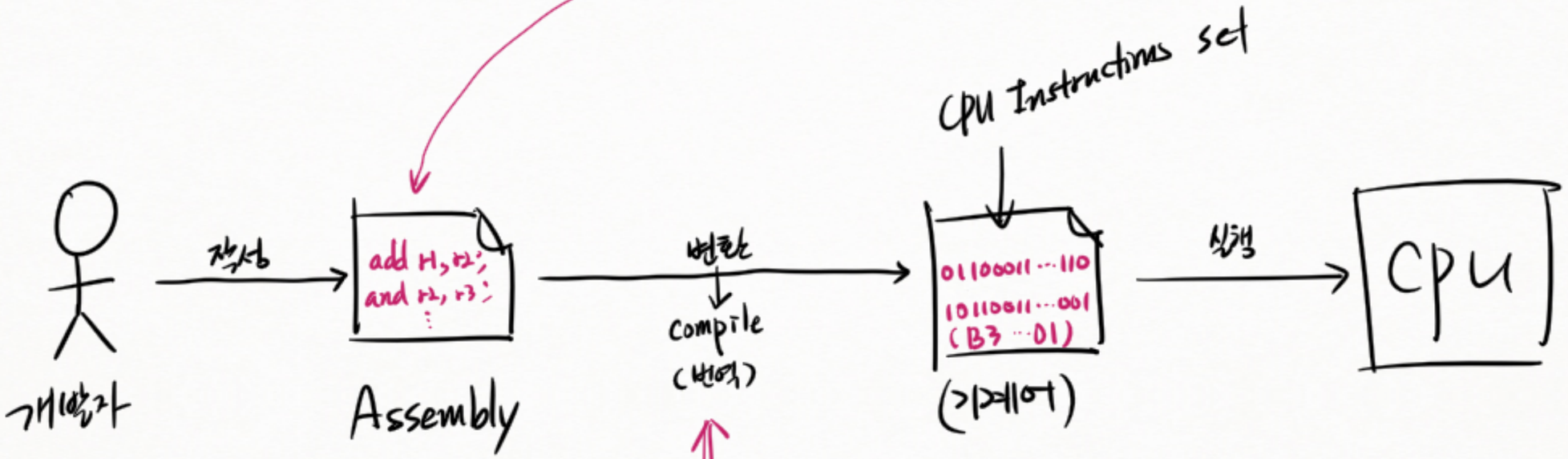
④ CPU bit가 32bit, 64bit로 늘어나면서 더 어렵게 되었다

⑤ CPU 마다 명령어가 다르기 때문에 다양한 CPU에서 실행할수 있는 명령어를 작성하기 매우 힘들다.

전기적 신호
=
비트규칙

* 명령어 작성 : Assembly

② 직접 기계어로 작성하는 대신 간결한 영어 단어로 이루어진 명령어 사용하기



Compiler ← Assembly 언어로 작성된 명령어를 CPU가 이해할 수 있는 기계어 바꿔주는 프로그램

★ 특징

① 명령어 영어 단어라 바꾸기에 작성하기 편하다

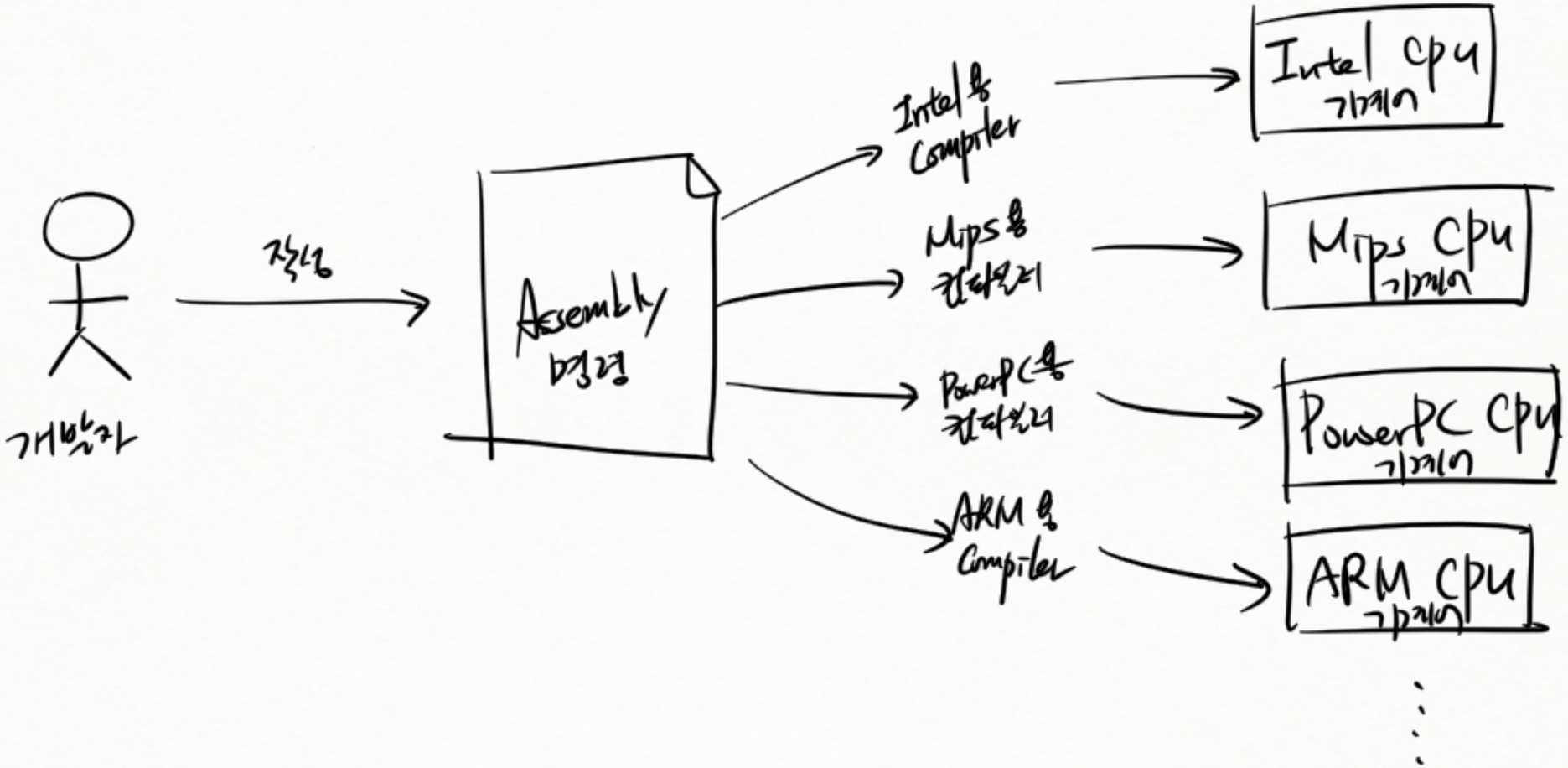
② CPU 마다 다르게 작성할 필요가 없다

⇓

왜?

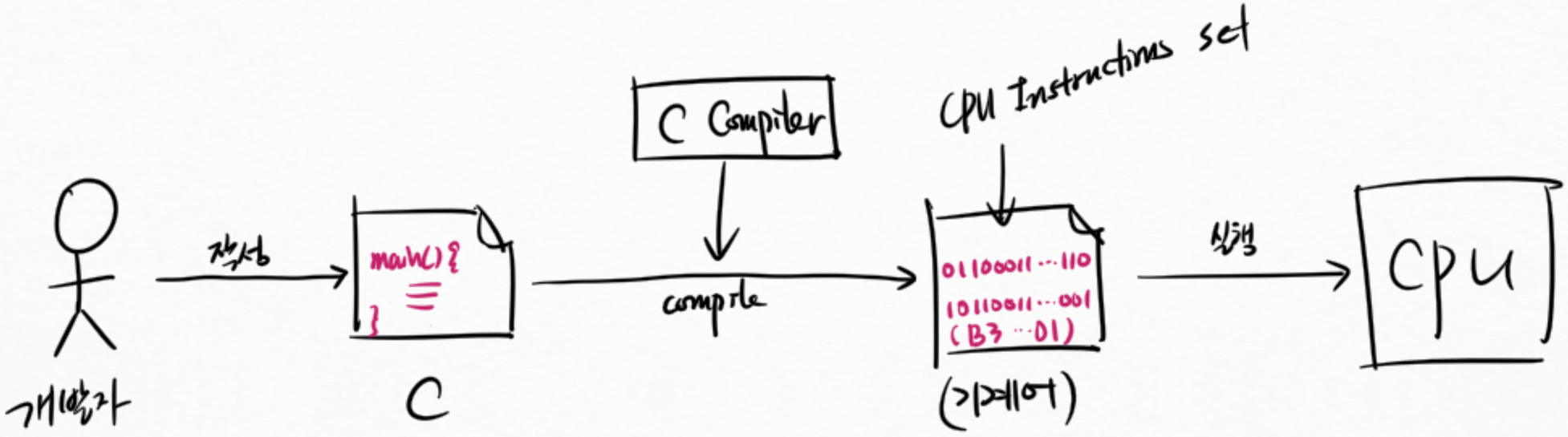
컴파일러가 CPU에 맞춰서 기계어로 번역해 준다.

* 명령어 작성 : Assembly 나 컴파일러

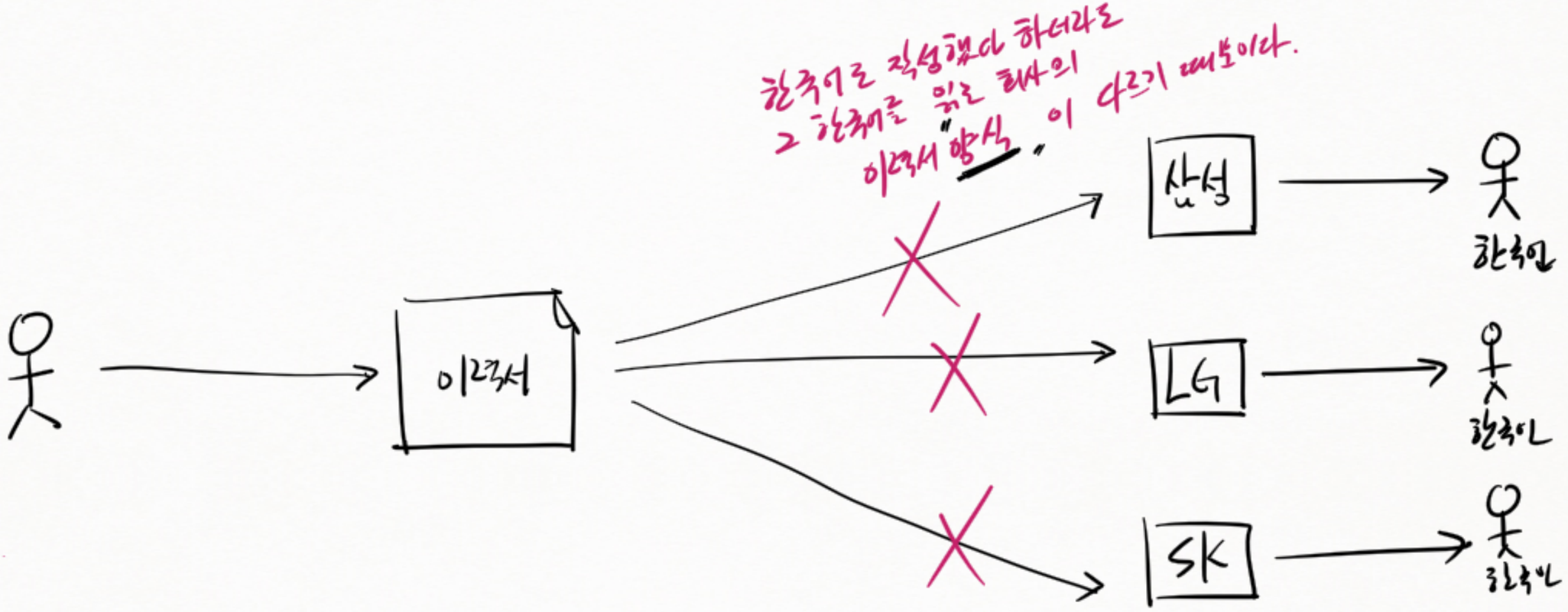


* 명령어 작성 : C

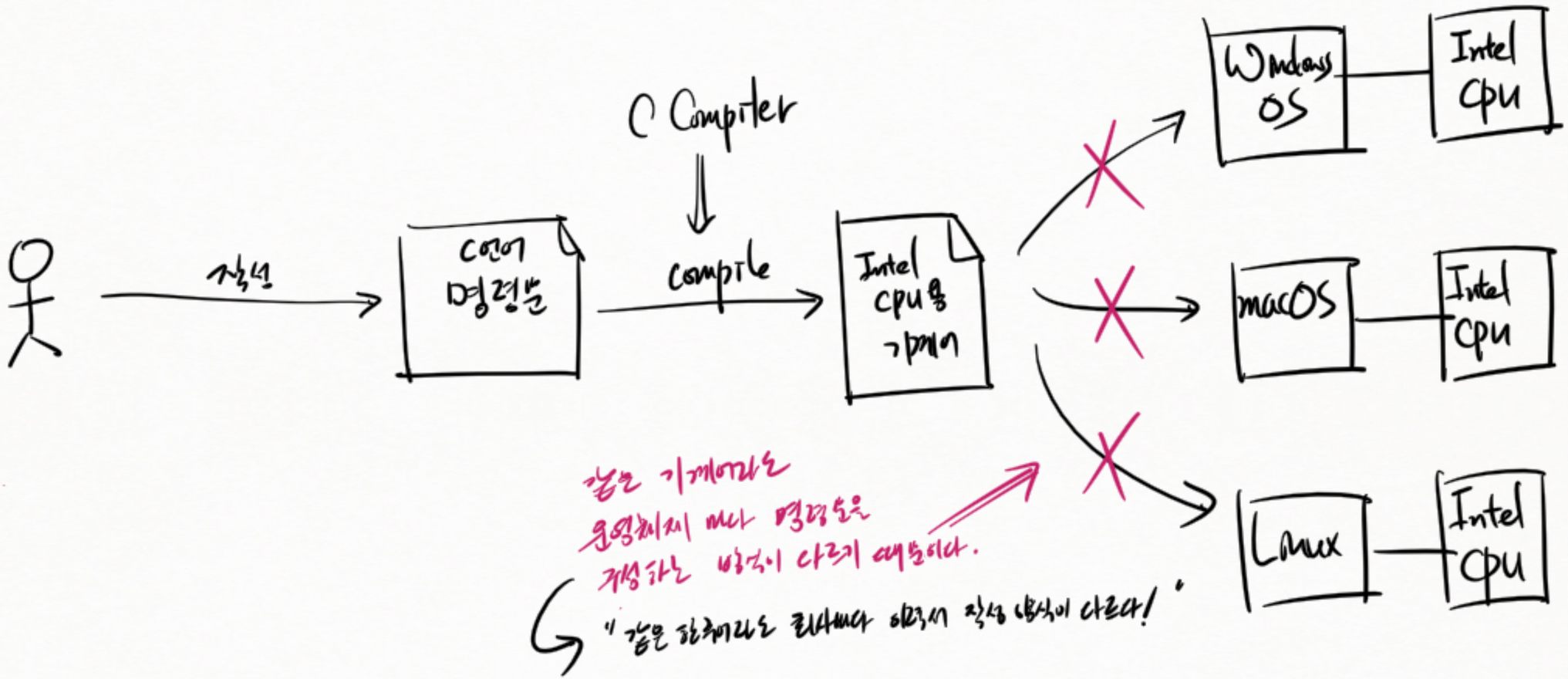
③ 더 인간 친화적인 프로그래밍 언어로 명령어 작성하기



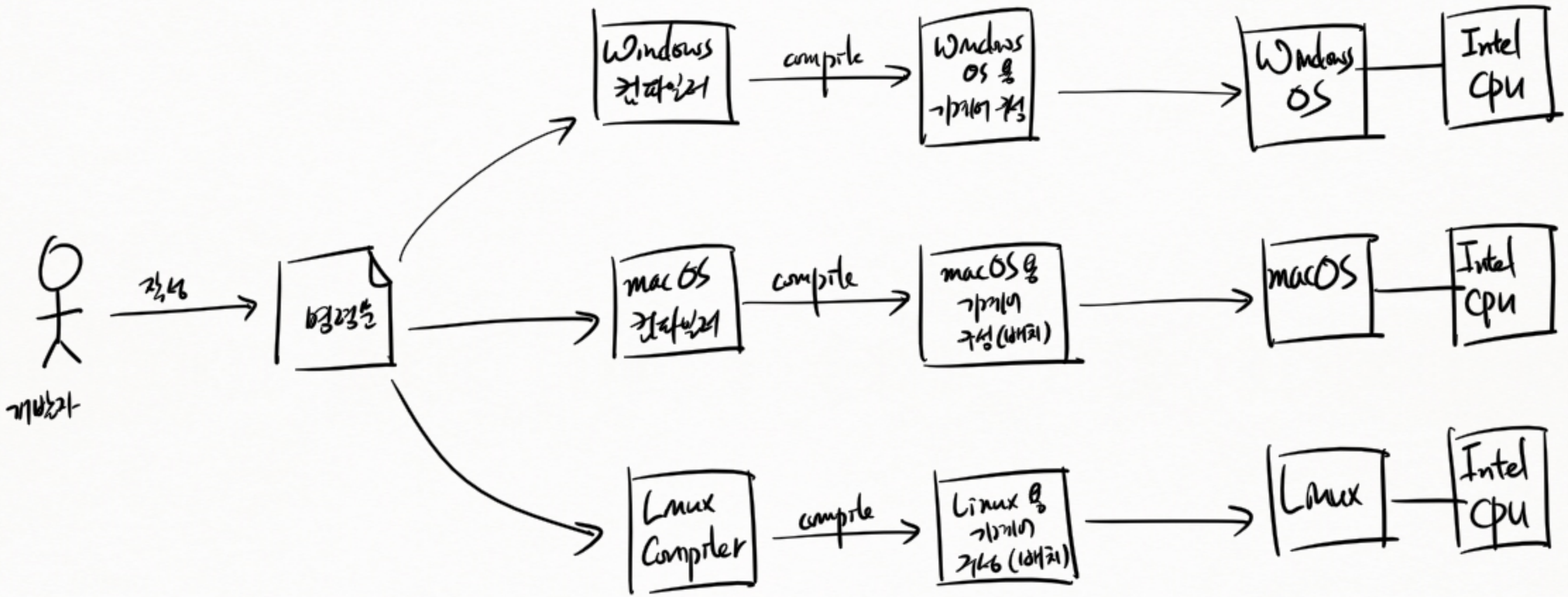
* 기계어, OS, CPU


한국어로 작성했다 하더라도
그 한국어를 읽는 회사의
이력서 "양식"이 다르기 때문이다.
이력서
삼성
LG
SK
한국어
한국어
한국어

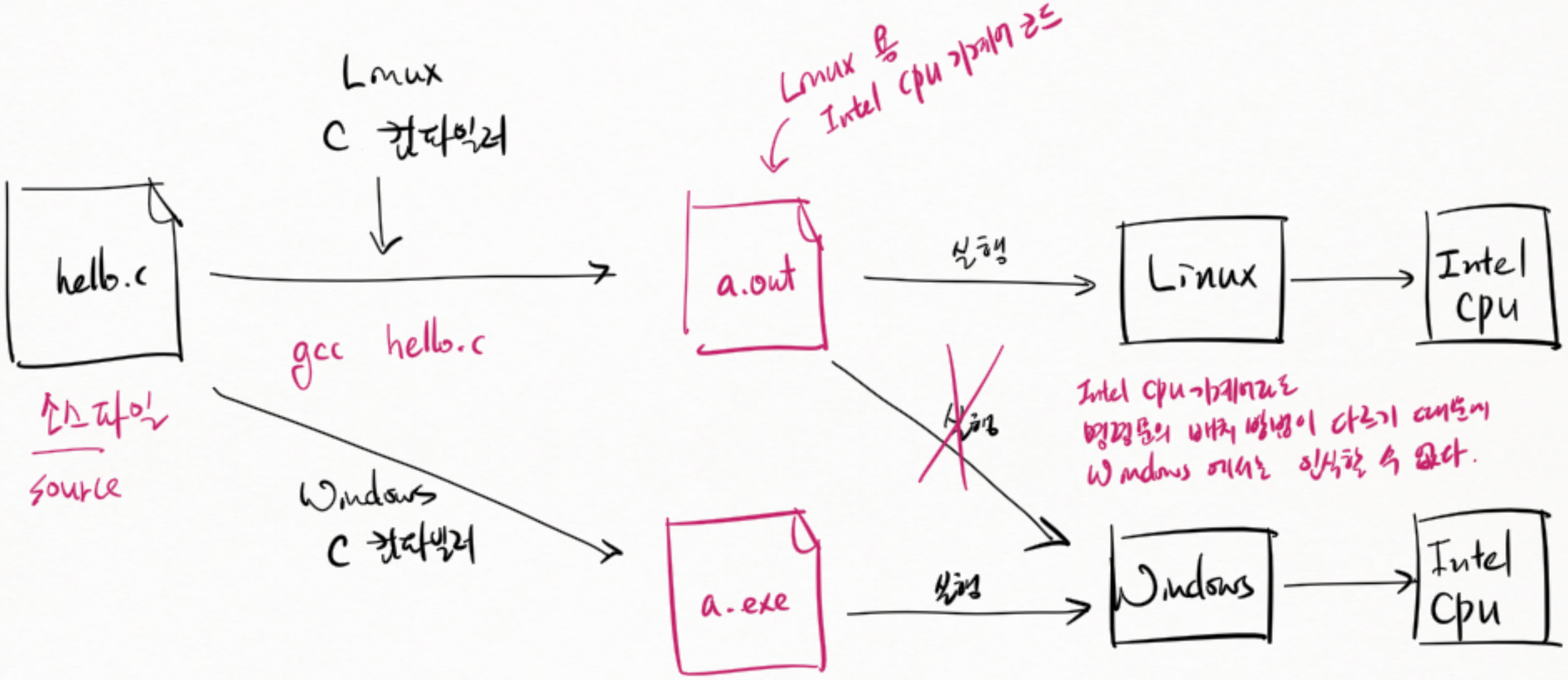
* 기계어, OS, CPU


작성
C언어 명령문
C Compiler
compile
Intel CPU용 기계어
Windows OS
Intel CPU
macOS
Intel CPU
Linux
Intel CPU
같은 기계어라도
운영체제 마다 명령문을
처리하는 방식이 다르기 때문이다.
"같은 한국어라도 회사마다 양식이 작성 방식이 다르다!"

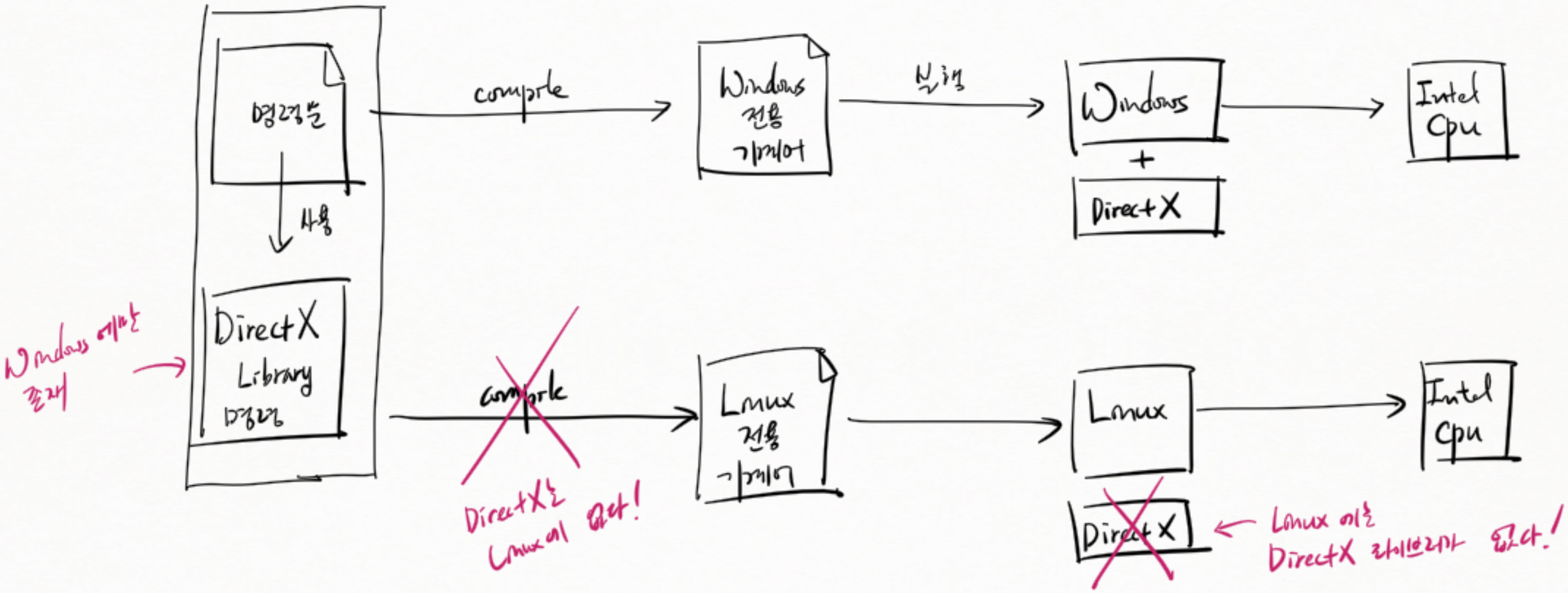
* 기계어, OS, CPU


개발자
작성
명령문
Windows 컴파일러
compile
Windows OS 용 기계어 구성
Windows OS
Intel CPU
macOS 컴파일러
compile
macOS용 기계어 구성(바이)
macOS
Intel CPU
Linux Compiler
compile
Linux 용 기계어 구성(바이)
Linux
Intel CPU

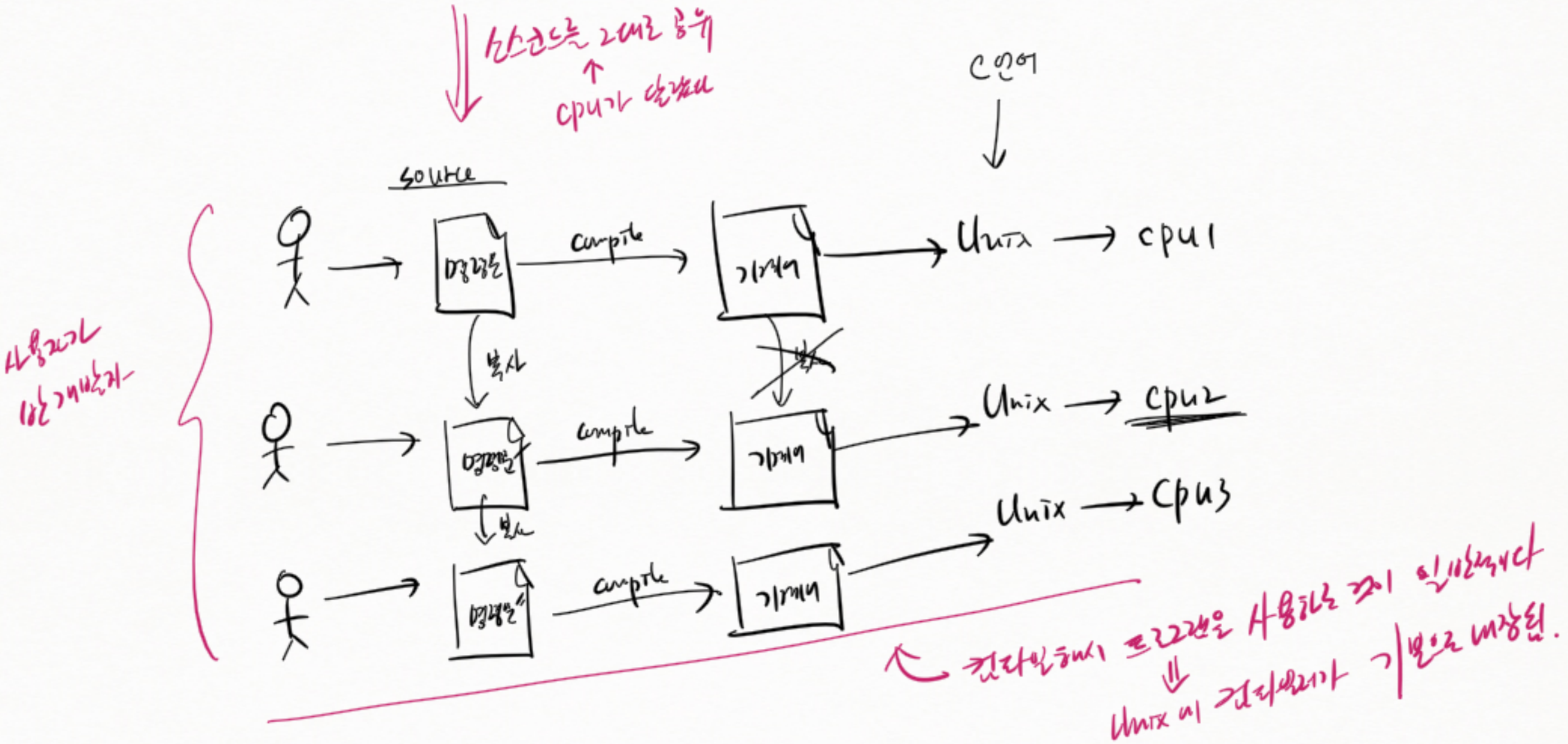
* C 컴파일


Linux
C 컴파일러
hello.c
소스파일
source
gcc hello.c
Windows
C 컴파일러
Linux 용
Intel cpu 기계어 코드
a.out
a.exe
실행
실행
실행
Linux
Windows
Intel Cpu
Intel cpu 기계어로
명령문의 배치 방법이 다르기 때문에
Windows 에서는 인식할 수 없다.

* C 컴파일 + OS 전용 명령 사용

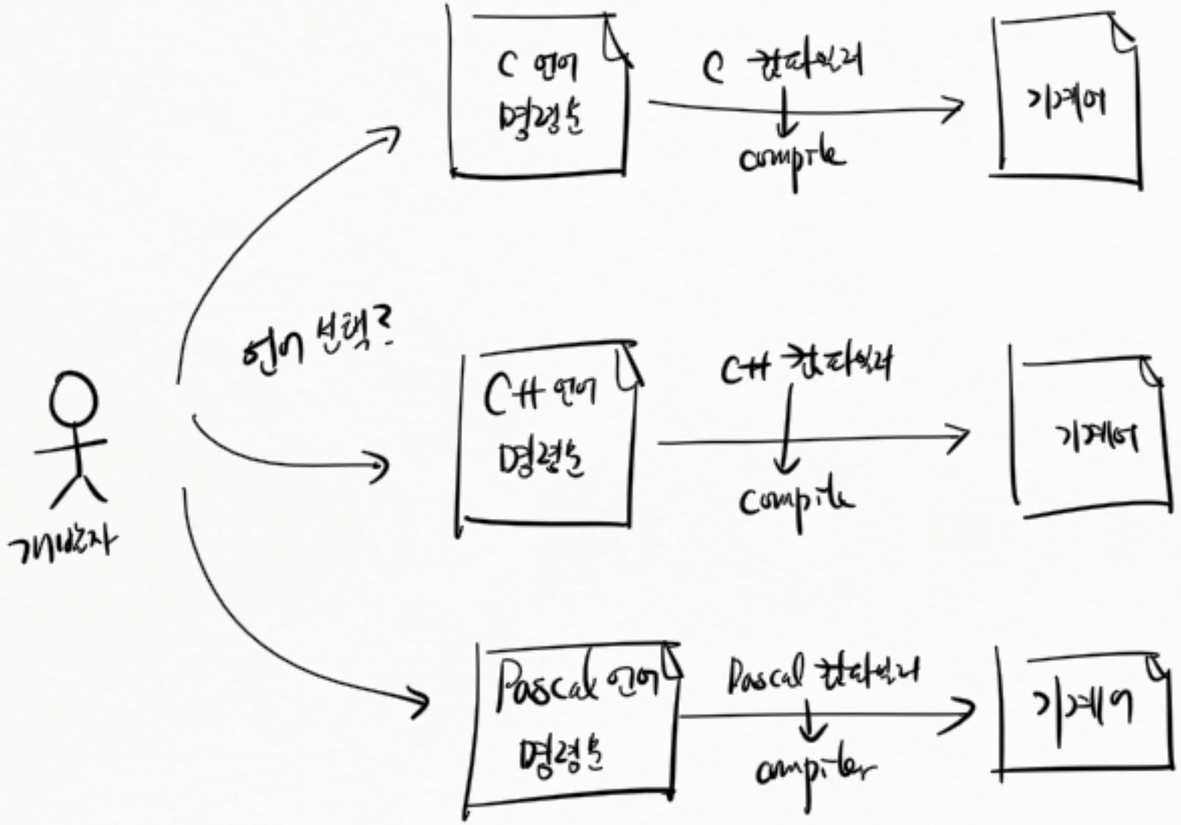


* 예전의 프로그램 공유 문화



컴파일해서 프로그램을 사용하는 것이 일반적이다
⇓
Unix에 컴파일러가 기본으로 내장됨.

* 프로그래밍 언어 와 컴파일러



☆ 언어 선택?

프로그래밍 언어마다 특징점이 있다

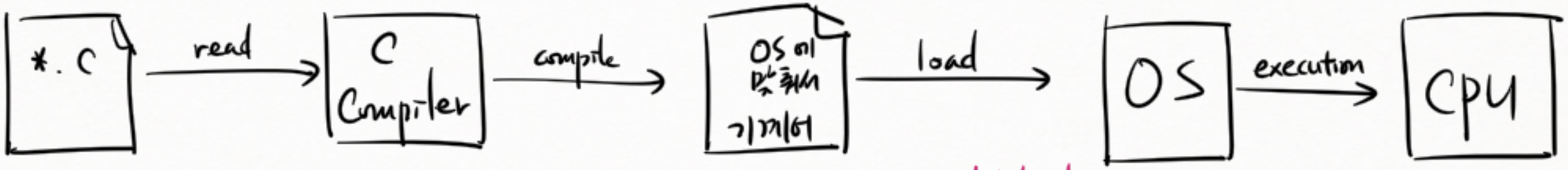
⇓

프로젝트 성격에 따라 언어를 선택해야 한다.

- 인공지능, 머신러닝, 딥러닝 : Python
- web app. : ☆ java, php, go
  국내
- web UI : ☆ javascript, TypeScript
- 통계 : R

* 컴파일 방식과 인터프리트 방식

컴파일 방식 :

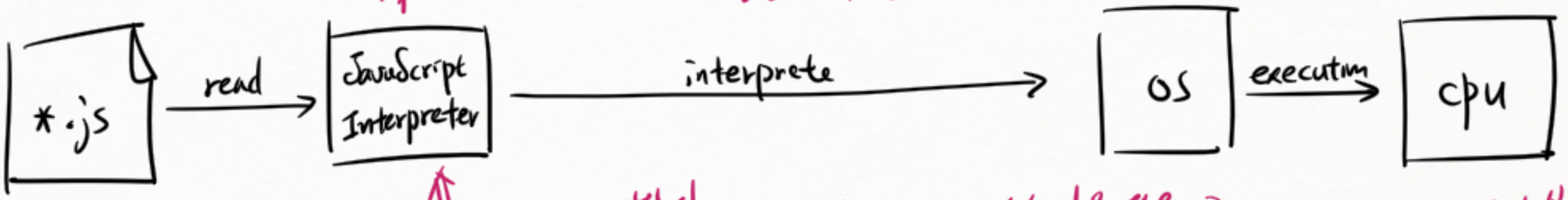


- 실행할 때 기계어 파일만 있으면 된다.
- 소스파일 필요 없다 → 소스 파일은 보관(자산)
- 실행할 때 컴파일러 불필요.

} 기계어 바로 실행 ⇓ 실행속도 빠르다

- VM
- viewer
- engine
- player

인터프리트 방식 :

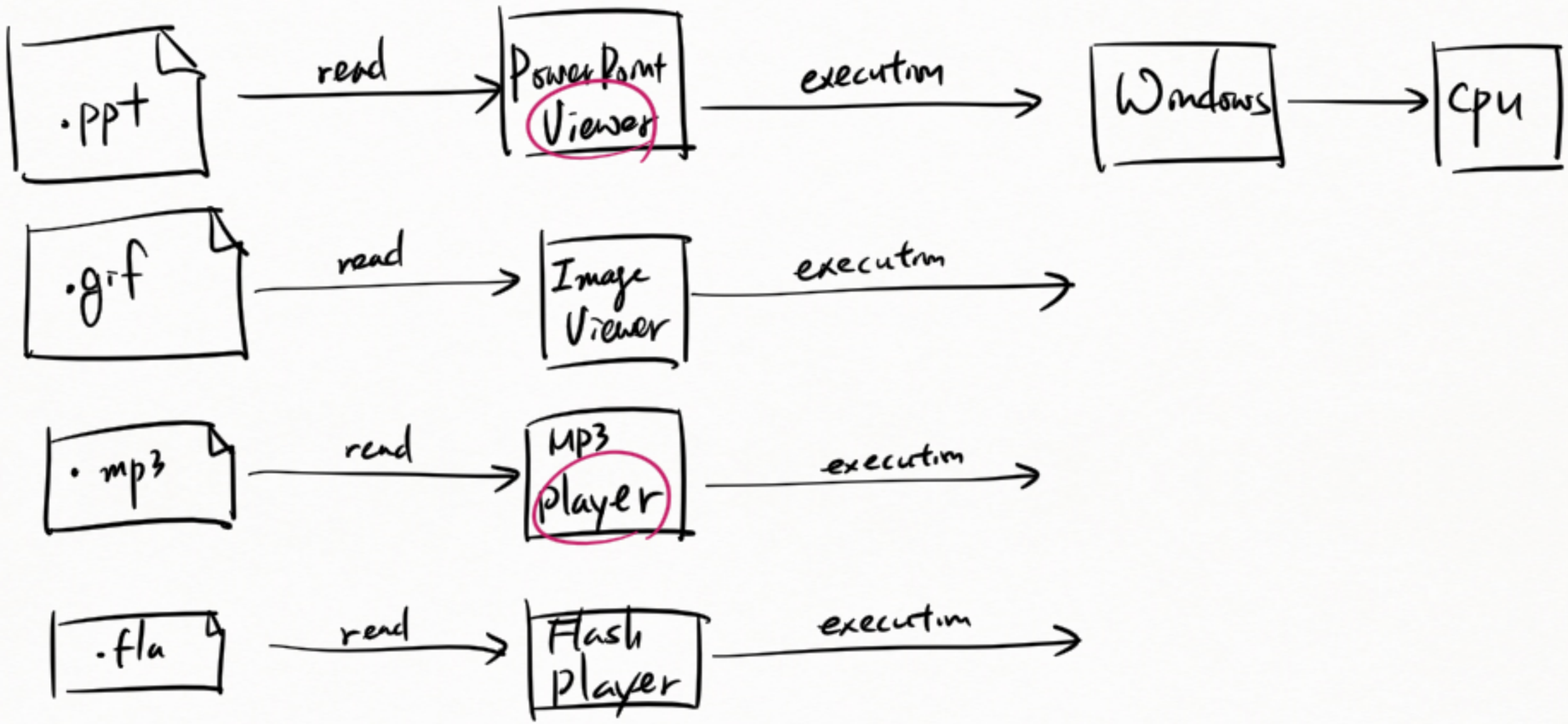


컴파일 하지 않는다

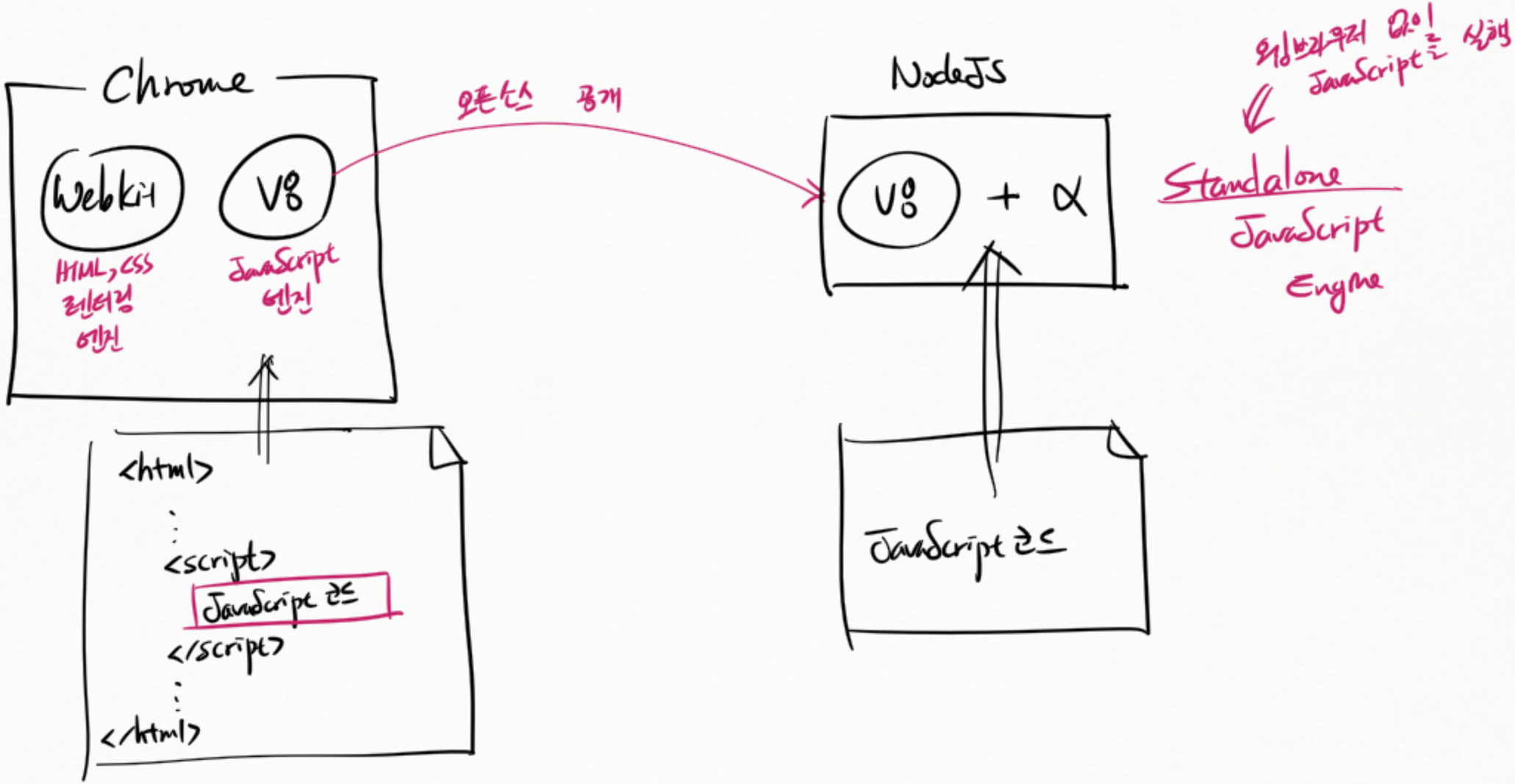
- 실행할 때마다 소스 파일 필요
  ↓
  소스파일 유m
  ↓
  자산으로 보관하기 힘들다.
- 실행하려면 인터프리터 필요.

} 매번 명령어를 해석해서 실행 ⇓ 실행속도 느리다.

* 인터프리터 = viewer = player = engine = virtual machine

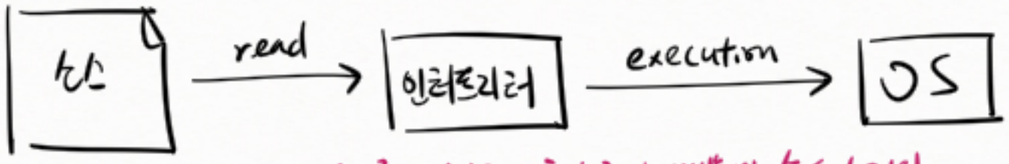


* JavaScript (Interpreter) Engine



* JIT Compile 과 AOT Compile

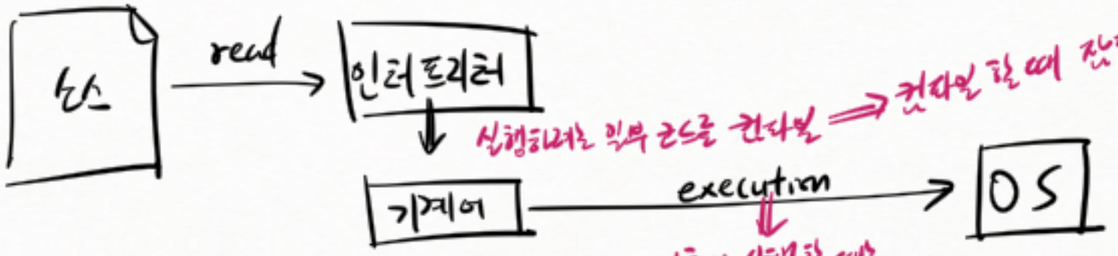
① Plain 인터프리터 :



매번 소스를 검사하고 해석하기 때문에 속도 느리다

② JIT (Just in Time) 컴파일 :

바로 그 시점에 일부 코드를 컴파일

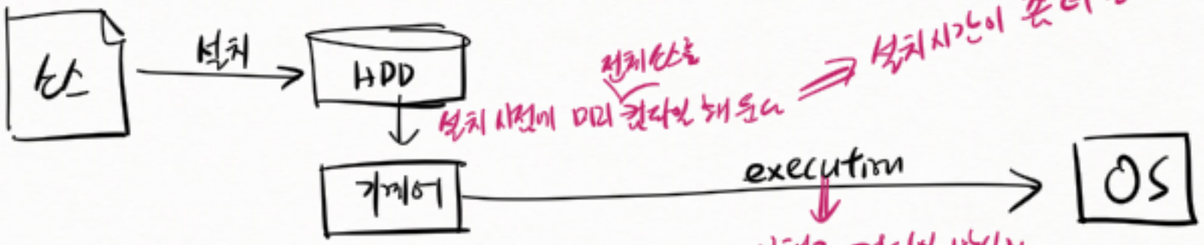


실행하려는 일부 코드를 컴파일 ⟹ 컴파일 할 때 잠깐 느려지지만

이후에 실행할 때는 빨라진다

③ AOT (Ahead of Time) 컴파일 :

특정 시점에 미리 컴파일 (앞서)

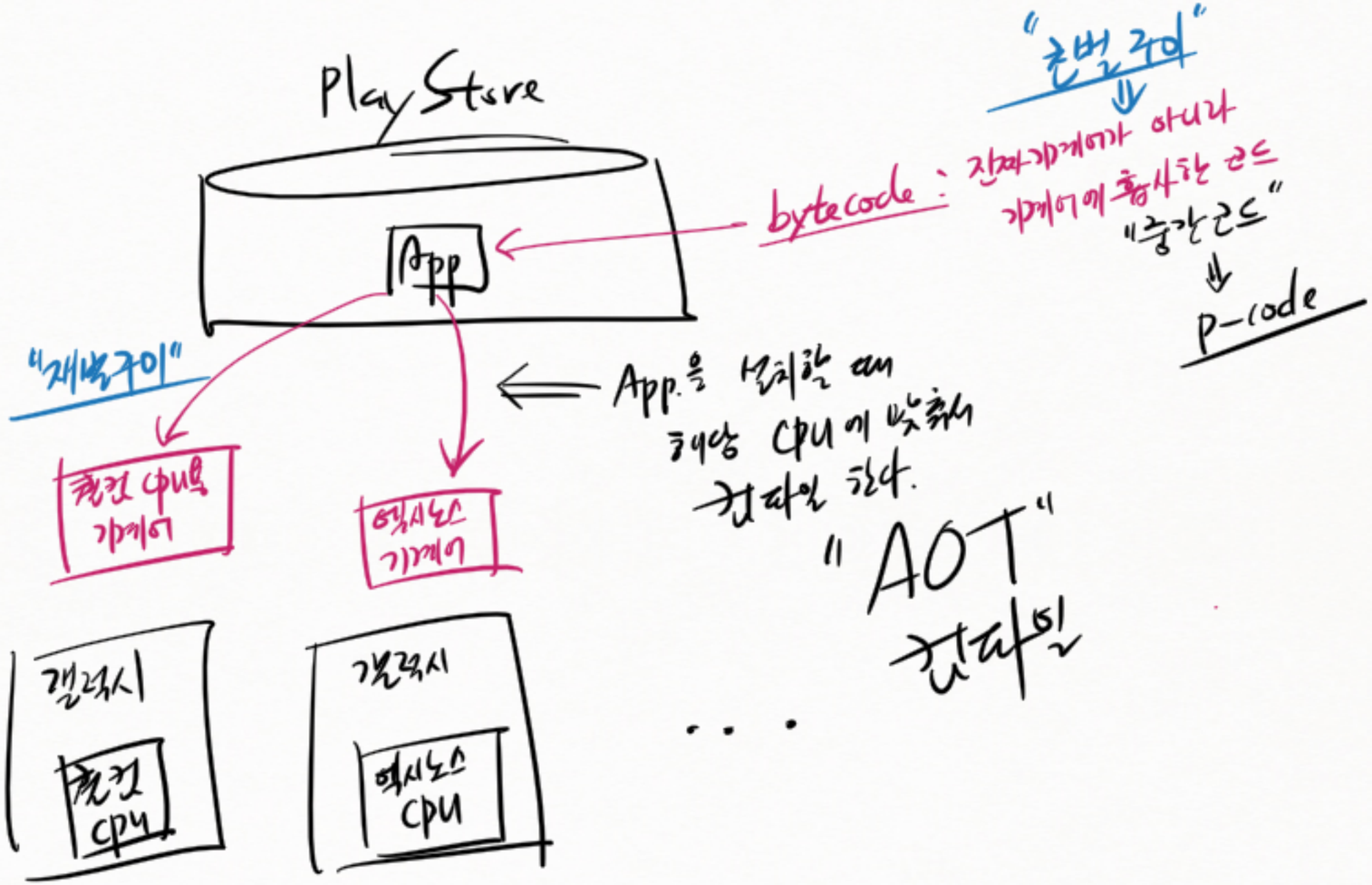


설치 시점에 미리 전체 소스를 컴파일 해 둔다 ⟹ 설치 시간이 좀 더 걸린다

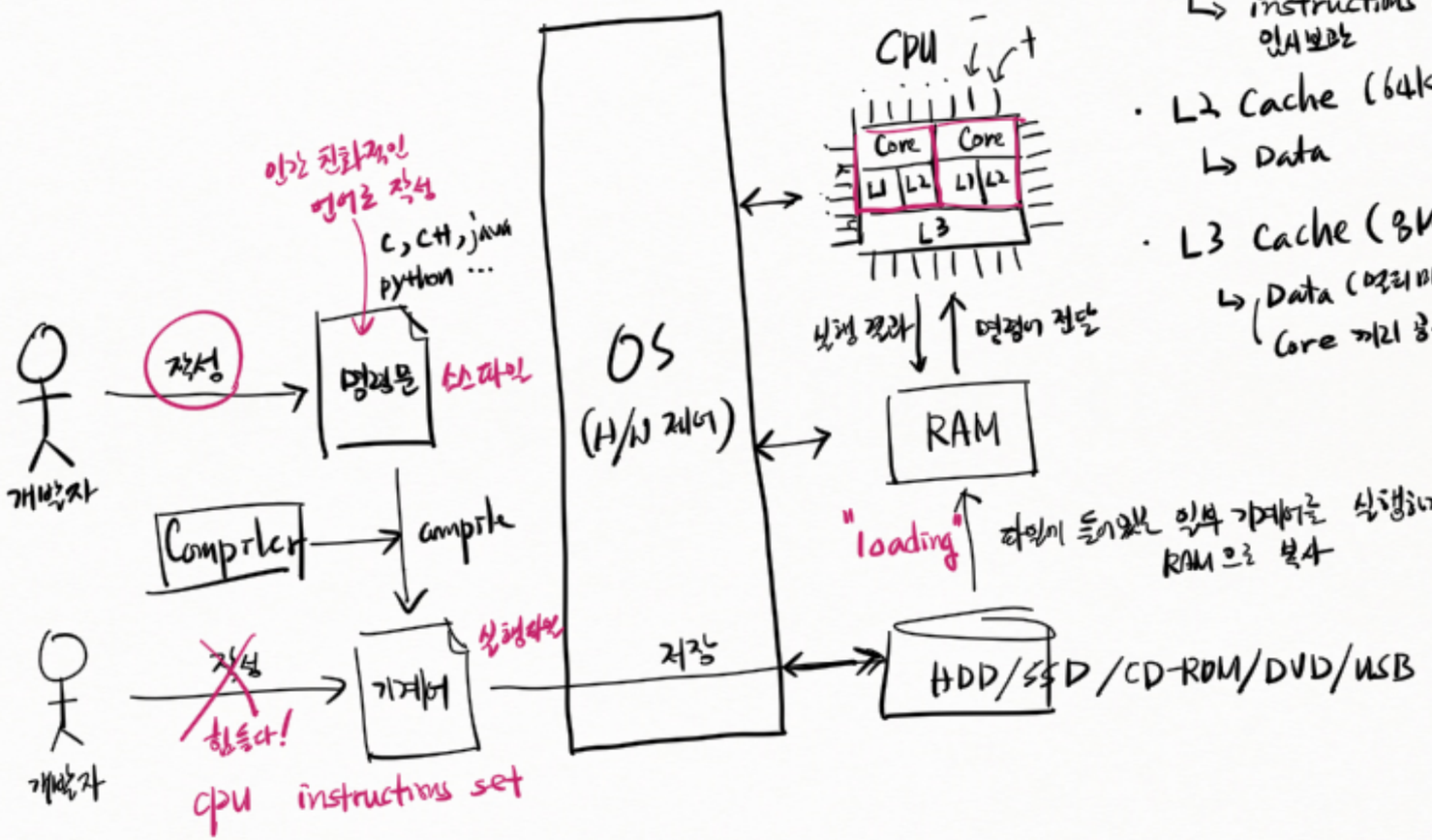
실행은 컴파일 방식과 같다.

예) Android App

* Android 나 App, AOT 컴파일



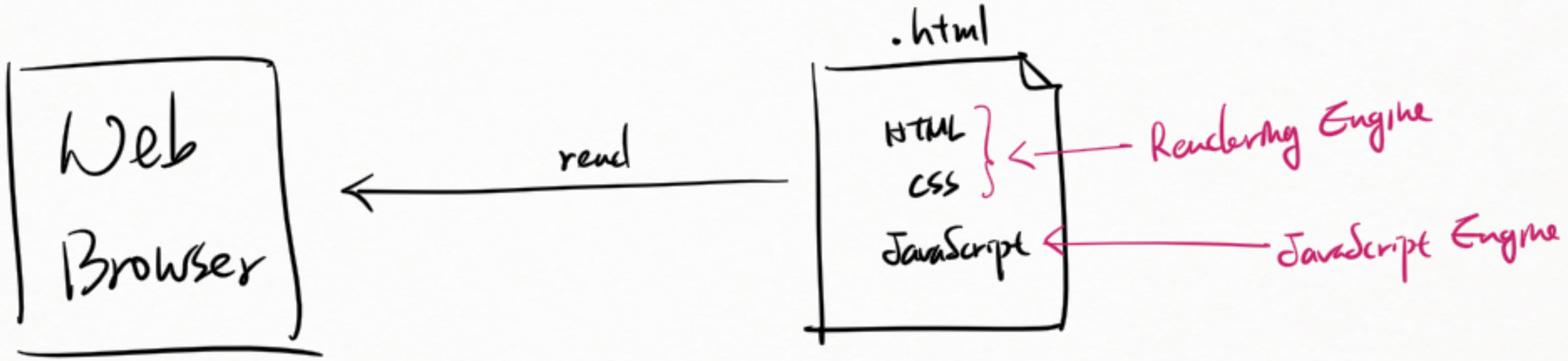
* OS, CPU, RAM, HDD, 기계어, 명령문 관계도



임시보관소

- L1 Cache (8KB ~ 64KB)
  - ↳ instructions set (명령) 임시보관
- L2 Cache (64KB ~ 4MB)
  - ↳ Data
- L3 Cache (8MB ~ 64MB)
  - ↳ Data (멀티미디어) Core 끼리 공유

* Web Browser 와 JavaScript



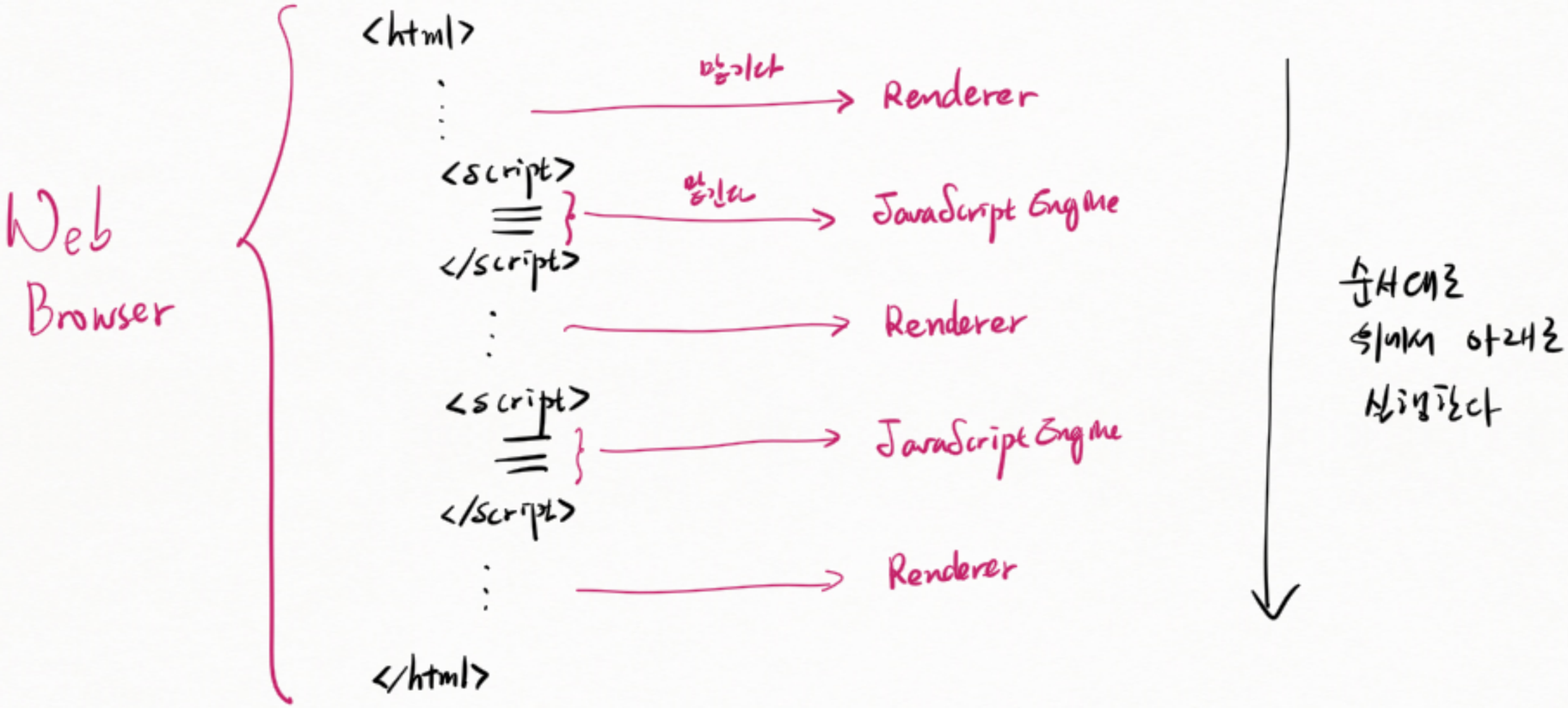
<script>

자바스크립트 코드

(명령문)

</script>

* <script> 태그


Web Browser
<html>
맡기다
Renderer
<script>
맡긴다
JavaScript Engine
</script>
Renderer
<script>
JavaScript Engine
</script>
Renderer
</html>
순서대로
위에서 아래로
실행한다

JavaScript 문법 요약

① console 출력

대한민국 서울 강남구 빌딩 A동 전화번호
객체 . 객체 . 객체 . 객체 . 객체 . 함수();

비트캠프 . 찾아라("홍길동")
↓
학생 . 전화번호()
⇓
비트캠프 . 찾아라("홍길동") . 전화번호()
함수 → 리턴객체 → 실행
함수

console . log (값, 값, 값, ...) ;

console ← (도구) = object(객체)

log ← (기능) = function(함수)

( ) ← (기능을 수행하는데 필요한 값을 전달) = argument(인자) / parameter(파라미터)

값 → Web Browser의 개발자 모드 콘솔창에 값을 출력

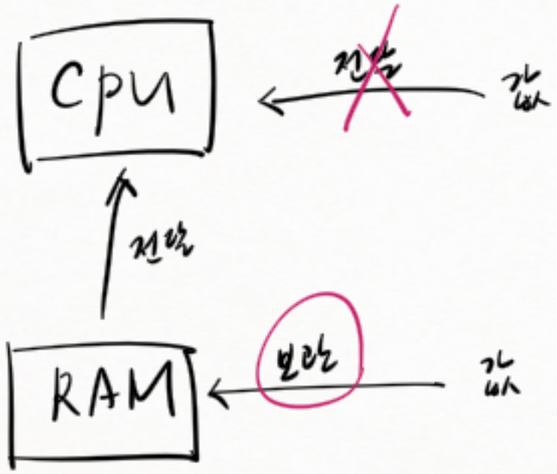
"aaa" / 'aaa' } (문자열) string

30 / 3.14 } (숫자) number

true / false } (논리) boolean

⋮

② 변수와 값 할당



var 변수명;
let 변수명;
const 변수명;

⇑ 문법

" 값을 저장할 메모리를 확보하라고 명령문 " ⟹ 변수 선언 (variables) (declaration)

" 메모리에 값을 저장하라고 명령문 " ⟹ 할당, 대입 (assignment) ⟹ 문법

assignment operator

변수명 = 값;
변수명 = 변수명;
변수명 = 함수();

변수에 들어있는 값 저장

함수의 반환값을 저장

* = : assignment operator
기능

size
| 10 |

message
| true |

PI
| 3.14159 |

localVar
| 100 |

저장

변수 = (변수 / 값 / 함수 / ⋮) ;

l-value
반드시 메모리여야 한다

r-value

* for 반복문

size

10

i

1

working

true

반복문을 실행하는 동안 사용될 변수. 루프 반복의 횟수를 제어하기 위해 선언

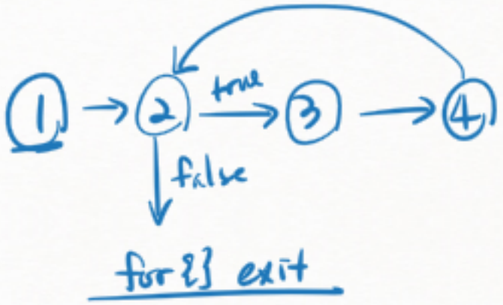
```
for ( var i = 0 ; i < size ; i++ ) {
}
```

① 처음 딱 한번만 실행

② 조건이 참이면 본문 안의 명령을 실행

④ i 메모리에 들어 있는 값을 1 증가시킨다.

③



* function (함수)

↳ 특정 기능을 수행하도록 짜놓은 명령문을 묶어서 이름을 붙인것.

↳ 필요할 때 언제든 재실행 가능!

⇓

코드 재사용성은 높이는 목적

함수 정의 (definition)

```
function plus(a, b) {
    return a + b;
}
```

함수이름

parameter

함수 실행 완료후 리턴할 결과값

함수 사용법 호출 (call)

```
var sum = plus(100, 200);
```

함수를 실행시켜 넘겨주는 값 (argument)

함수 호출

함수의 리턴값을 저장

argument를 받는 변수

⇓

parameter

* 중첩 함수 호출

① 호출 : plus() 함수 실행

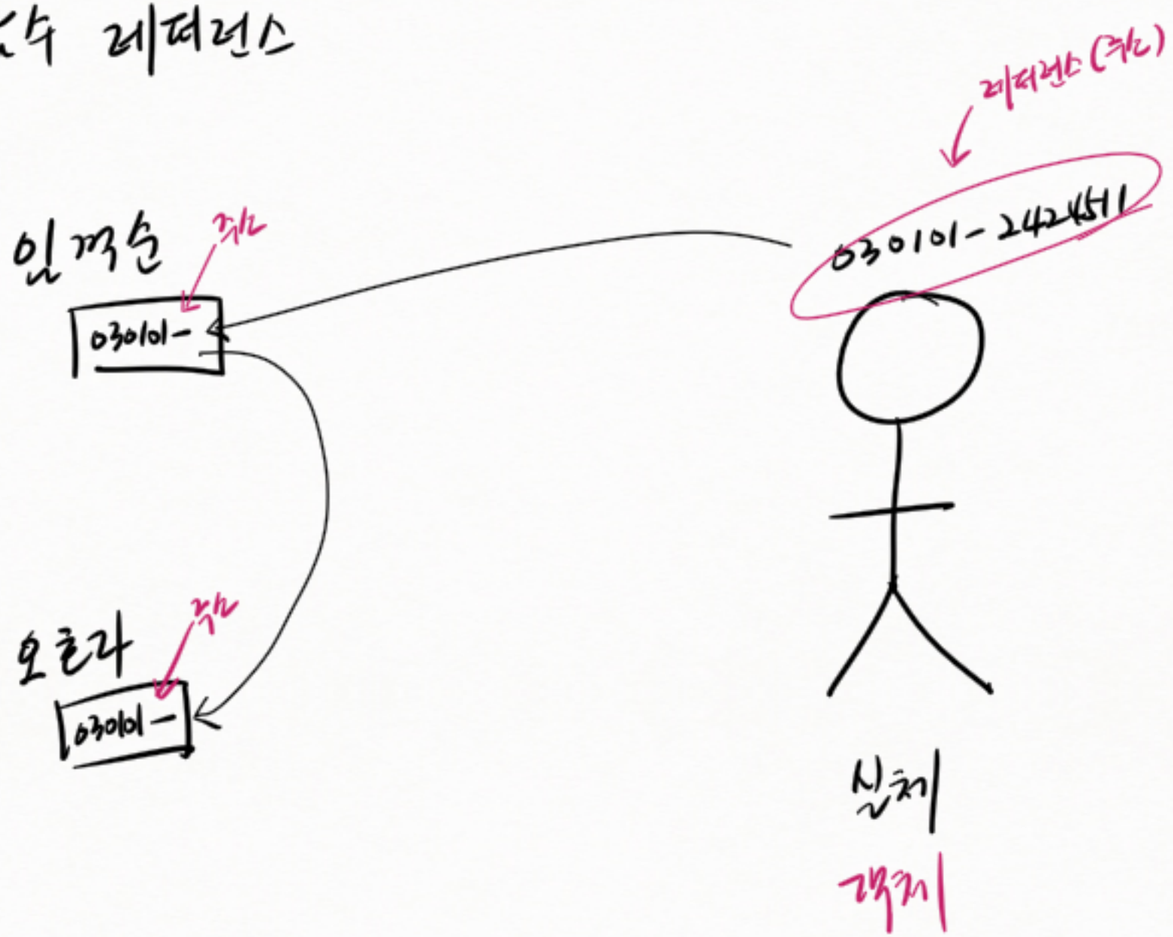
plus(100, 200)

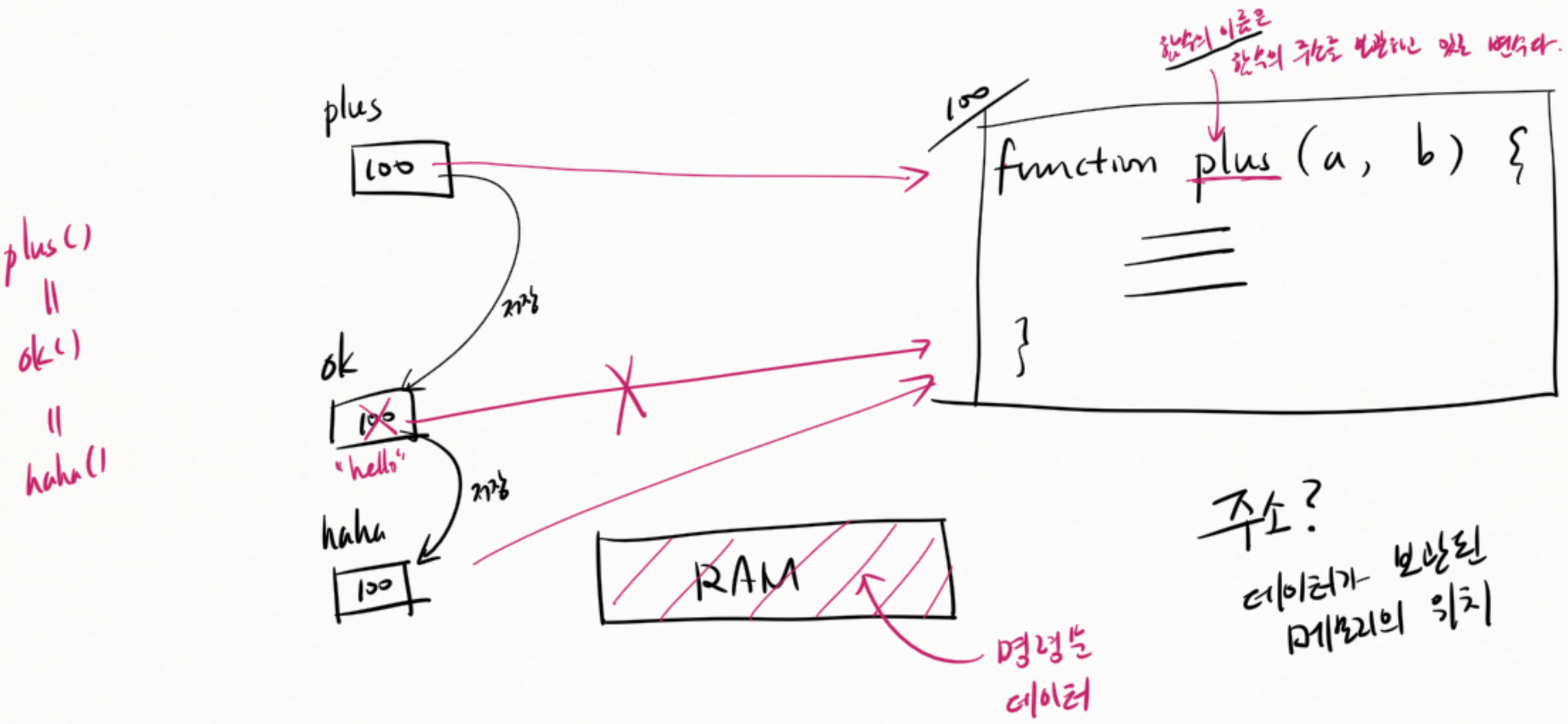
② 리턴 값을 받는다

console.log( )

③ 실행

* 함수 레퍼런스




함수의 이름은
함수의 주소를 반환하는 역할 변수다.
plus
100
ok
100
"hello"
haha
100
저장
저장
function plus (a, b) {
}
100
plus()
||
ok()
||
haha()
RAM
명령문
데이터
주소?
데이터가 보관된
메모리의 위치

* static type binding vs dynamic type binding

Java

int a; ← 변수 선언할 때 메모리의 용도와 공간 int 값만 저장 가능

a

a = 100;

a = -20;

a ≠ "Helb";

컴파일 언어 컴파일러

Language 류의 프로그래밍 언어

엄격 → 프로그래밍이 까다롭다

↓

유지보수가 쉽다

JavaScript

var a;

a

a = 100; ← a는 정수 변수

a = "Hello"; ← a는 문자열 변수

a = true; ← a는 boolean 변수

값을 넣을 때 변수의 유형이 바뀐다.

자바스크립트

Script 류의 프로그래밍 언어

자유분방 → 프로그래밍이 쉽다

↓

유지보수가 어렵다

* 객체 : 덩어리 (변수 + 함수 + 객체)

```
var name = "홍길동";
var age = 20;
var working = false;
```

→

```
var obj = new Object();
```

obj

200

```
obj.name = "홍길동";
obj.age = 20;
obj.working = false;
```

200

key	value
name	"홍길동"
age	20
working	false

(덩어리)

객체 = object

* 객체 : 덩어리 (변수 + 함수 + 객체)

```
var name = "홍길동";
var age = 20;
var working = false;
```

→

```
var obj = {
    name : "홍길동",
    age : 20,
    working : false
};
```

"object literal"

객체는 자바스크립트 코드로 표현한 것

obj

100

100

key	value
name	"홍길동"
age	20
working	false

Literal (값을 코드로 표현한 것)

문자열 → "abc", 'abc'

숫자 → 10, 3.14

논리 → true, false

변수에 값이 저장된적이 없는 상태 → undefined

객체 주소가 없는 상태 → null

⋮

* DOM API – 태그를 다루는 프로그램을 짤 때 사용하는 도구
(programming)

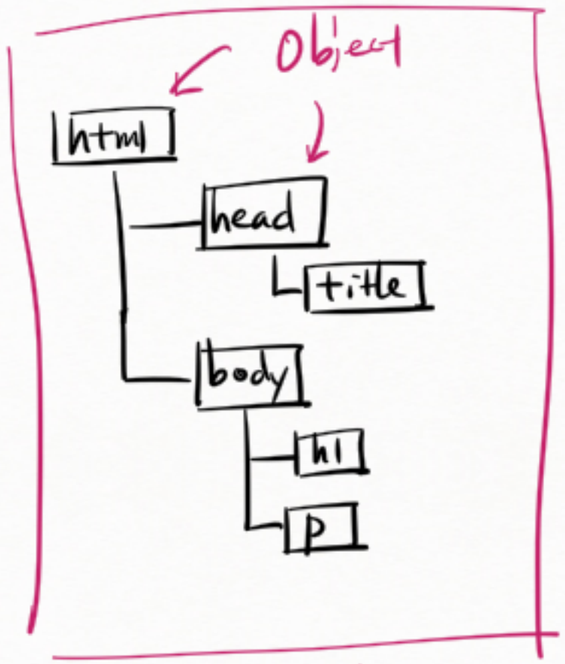
↓

Document ← HTML
Object ← 객체
Model ← 구조로 만든 것.

Application Programming Interface (도구) } App.을 프로그래밍 할 때 사용하는 도구.

```
<!DOCTYPE html>
<html>
    <head>
        <title> —— </title>
    </head>
    <body>
        <h1> —— </h1>
        <p> —— </p>
    </body>
</html>
```

Document



Model

tag를 tree 구조로 객체화 시킨것

⇓

tag를 찾고, 변경하고, 추가하고, 삭제하기 쉽도록.

⇓

"tag를 다루기 쉽도록"

* 객체와 함수, 파라미터의 관계

함수
=
메서드
=
operator

함수가 기본데이터를 다룰 때 사용할 값

document.getElementById("태그아이디")

객체

함수 (실제 일하는 놈)

함수가 일할 때 참고할 데이터

기본데이터

작업이 사용하는 복가데이터

학생명단.find("홍길동")

기본데이터

함수 = 작업자 (operator)

학생정보 덩어리 (객체)

* JavaScript 와 ECMAScript 관계

JavaScript → 1997.6 → 1.5 → ... →

1997.6 ↓ 표준으로 제출

2000.11 ↑ 1.5

... ↑ 1.5

기준 ⇒ ECMA 단체 : ECMA-262 (제목) → ECMAScript 3rd → 5.1 (2011) → 2015 (6) → 2016 (7) → 2017 (8) → 2019 (9) → ...

2015 (6) ↓

Adobe Flash : ActionScript → ... → X

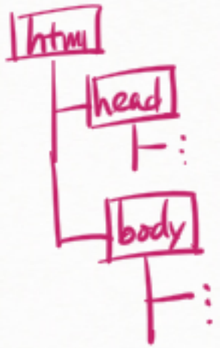
* 태그 찾기 ⟶ 이벤트 리스너 등록

(document가 가리키는 DOM Tree에서)
HTML 문서에서 'btn1'이라고
태그를 찾아 그 객체의 주소를 리턴

```
var tag1 = document.getElementById("btn1")
```

('btn1' 태그의 주소)
버튼태그

DOM Tree를 가리킨다



특정 이벤트가 발생하였을 때 호출되는 함수,
"event handler"
||
"event listener"

사용자가 tag1을 마우스로 클릭하면
'onclick'이라는 이름으로 저장된 함수를 호출한다.
만약 이 이름으로 저장된 함수 없으면
아무일도 일어나지 않는다.

```
tag1.onclick = function() { — };
```

'onclick' 이름으로 함수(의 주소) 저장

함수 정의
↓
메모리에 보관

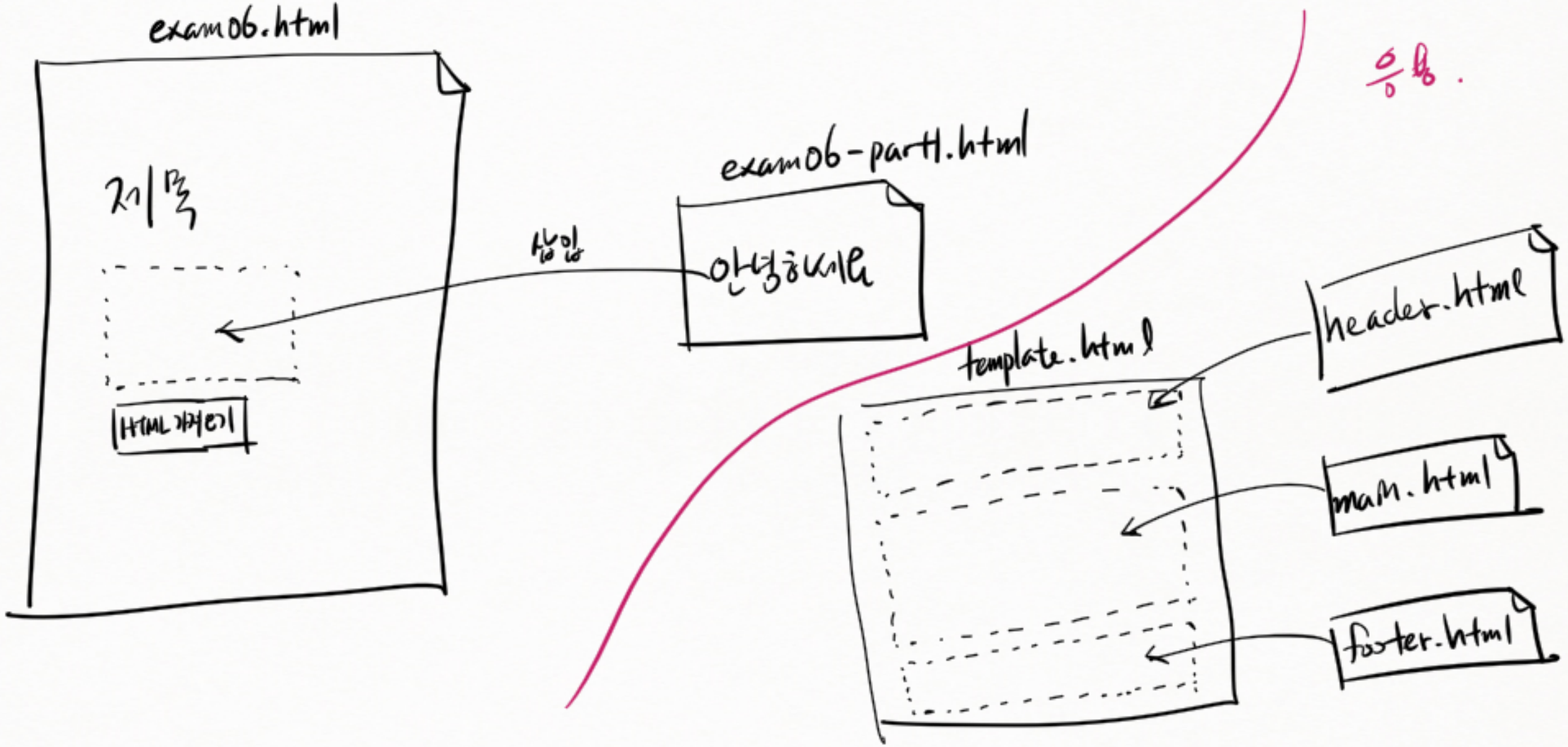
click 이벤트에 대해
리스너를 등록하는 코드

* tag의 content 설정

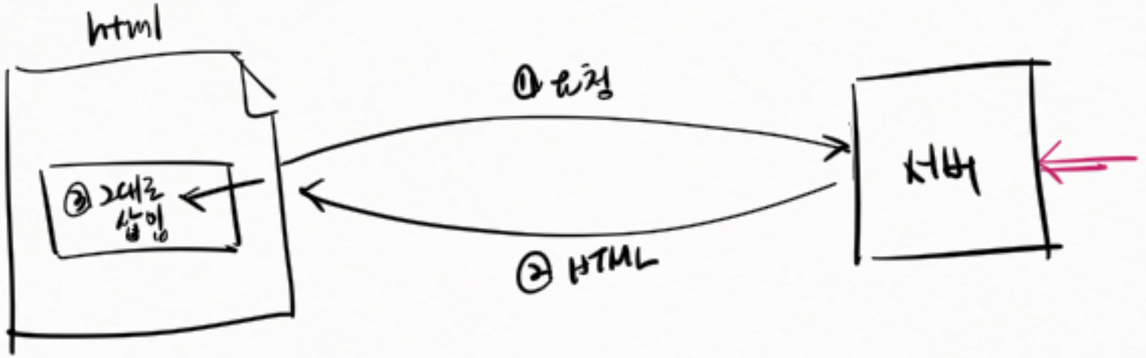
```
document.getElementById('p1');
↓
tag2.innerHTML = "하하하";
```

 하하하

* 웹페이지 조립


exam06.html
제목
HTML 파일
삽입
exam06-part1.html
안녕하세요
template.html
header.html
main.html
footer.html
응용.

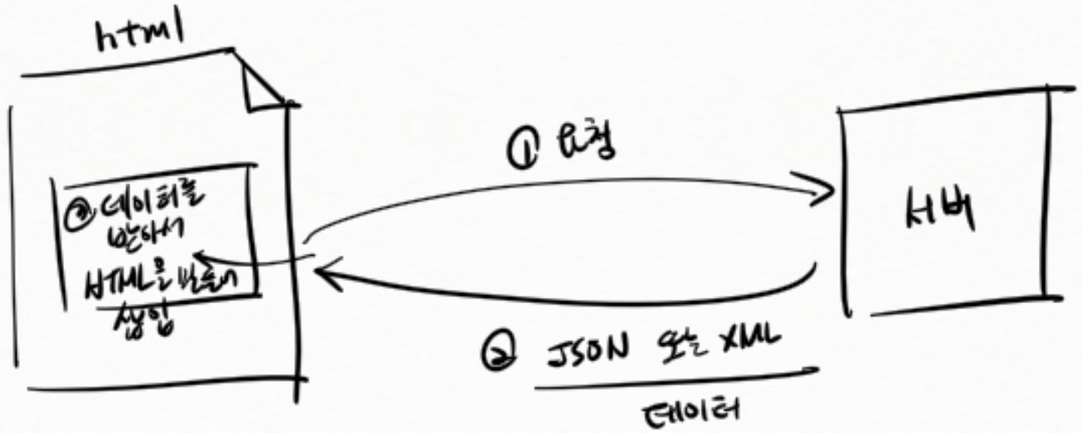
* 서버에 자원을 요청 : "server-side rendering" vs "client-side rendering"



server-side rendering 방식
↳ 서버에서 HTML을 만들어 준다

예) JSP, Thymleaf 등

client-side rendering 방식
↳ 클라이언트에서 HTML을 만든다



예) AJAX 기술
↳ React, Angular 등

* JSON 포맷

```
var obj = {
    name: '홍길동',
    age: 20,
    working: false,
    hell: function() { ... }
};
```

object literal

JavaScript 의 object literal 문법을 참고하여 만든 포맷

```
{
  "name": "홍길동",
  "age": 20,
  "working": false
}
```

객체 (변수) property 이름을 반드시 큰따옴표로 표현

큰따옴표로 " " 으로 표현

함수로 표현할수 없다 변수만 가능

JavaScript Object Notation

"JSON"

* JSON Built-in 객체

JSON 포맷 문자열

JSON.parse( )

return

자바스크립트 객체

자바스크립트 객체

JSON.stringify( )

return

JSON 포맷 문자열

* 과제 : tictactoe 게임 만들기
(2022-12-02)

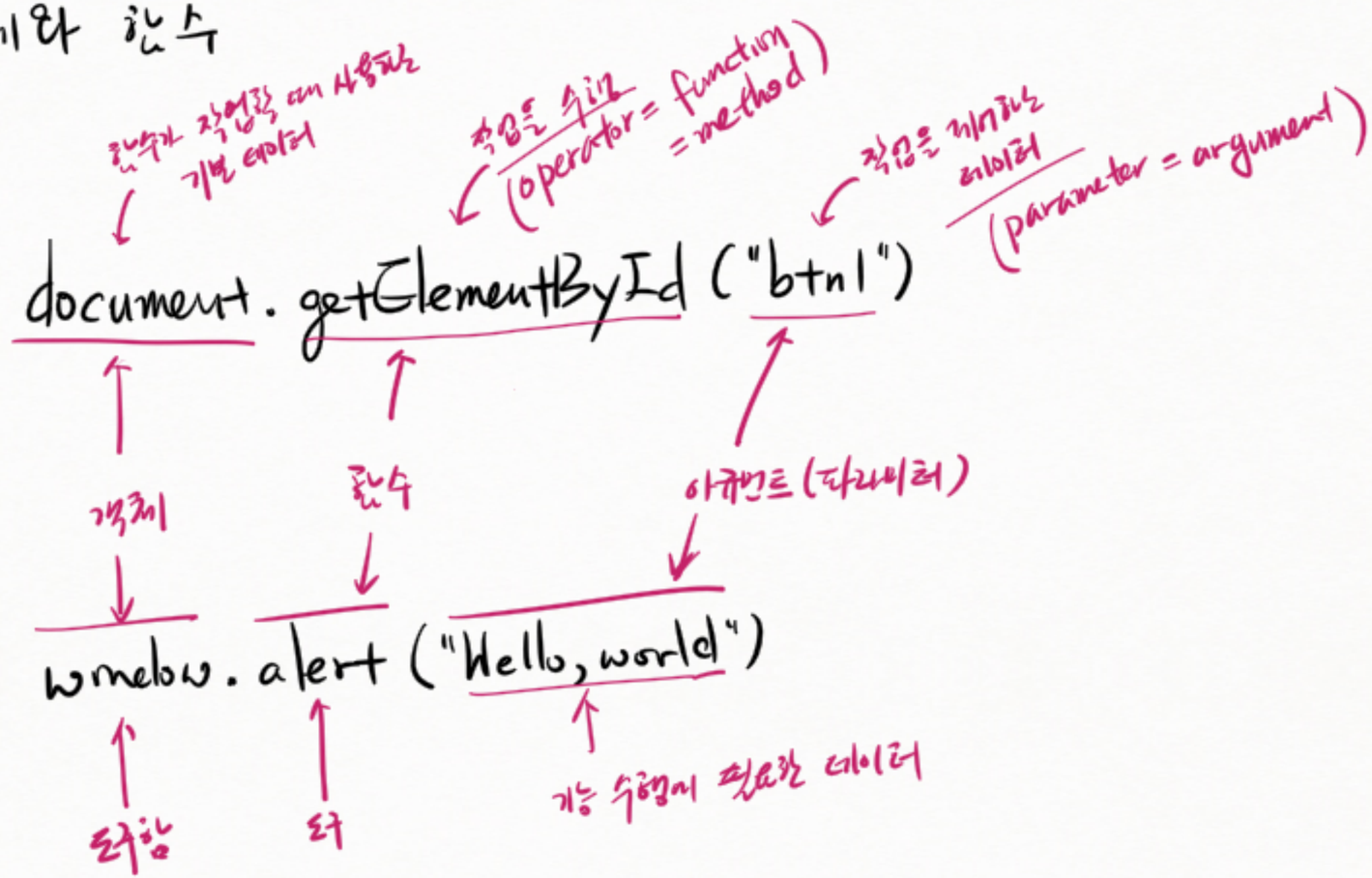
O	O	X
O	X	O
X	X	O

X가 이겼습니다!

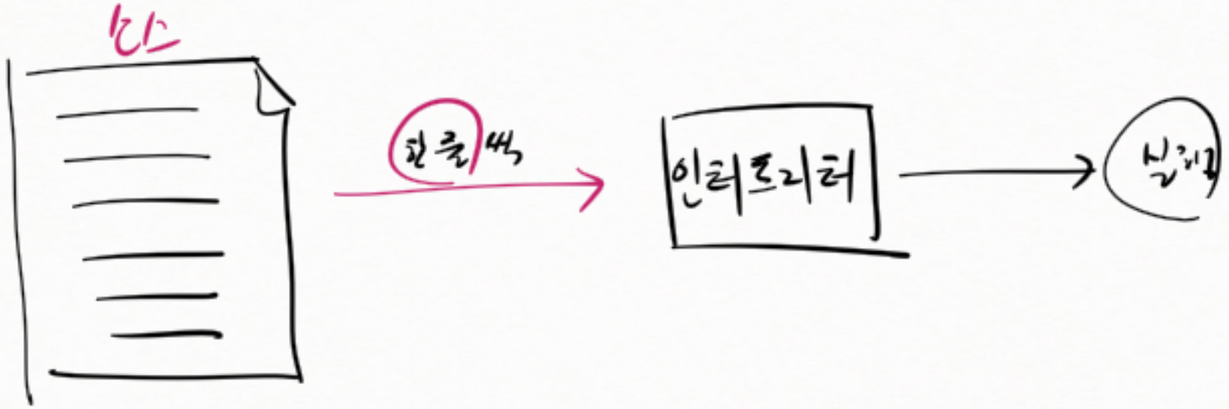
O이 이겼습니다!

무승부 입니다!

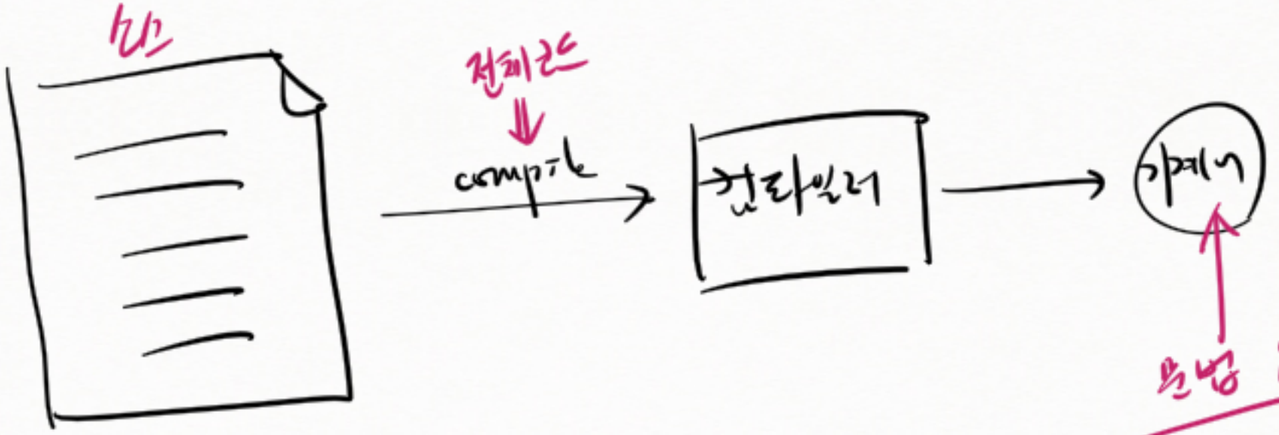
* 객체와 함수


함수와 작업할 때 사용되는 기본 데이터
작업을 수행 (operator = function = method)
작업을 제어하는 데이터 (parameter = argument)
document . getElementById ("btn1")
객체
함수
아규먼트 (파라미터)
window . alert ("Hello, world")
도큐먼트
창
기능 수행에 필요한 데이터

* 인터프리트 방식과 컴파일 방식



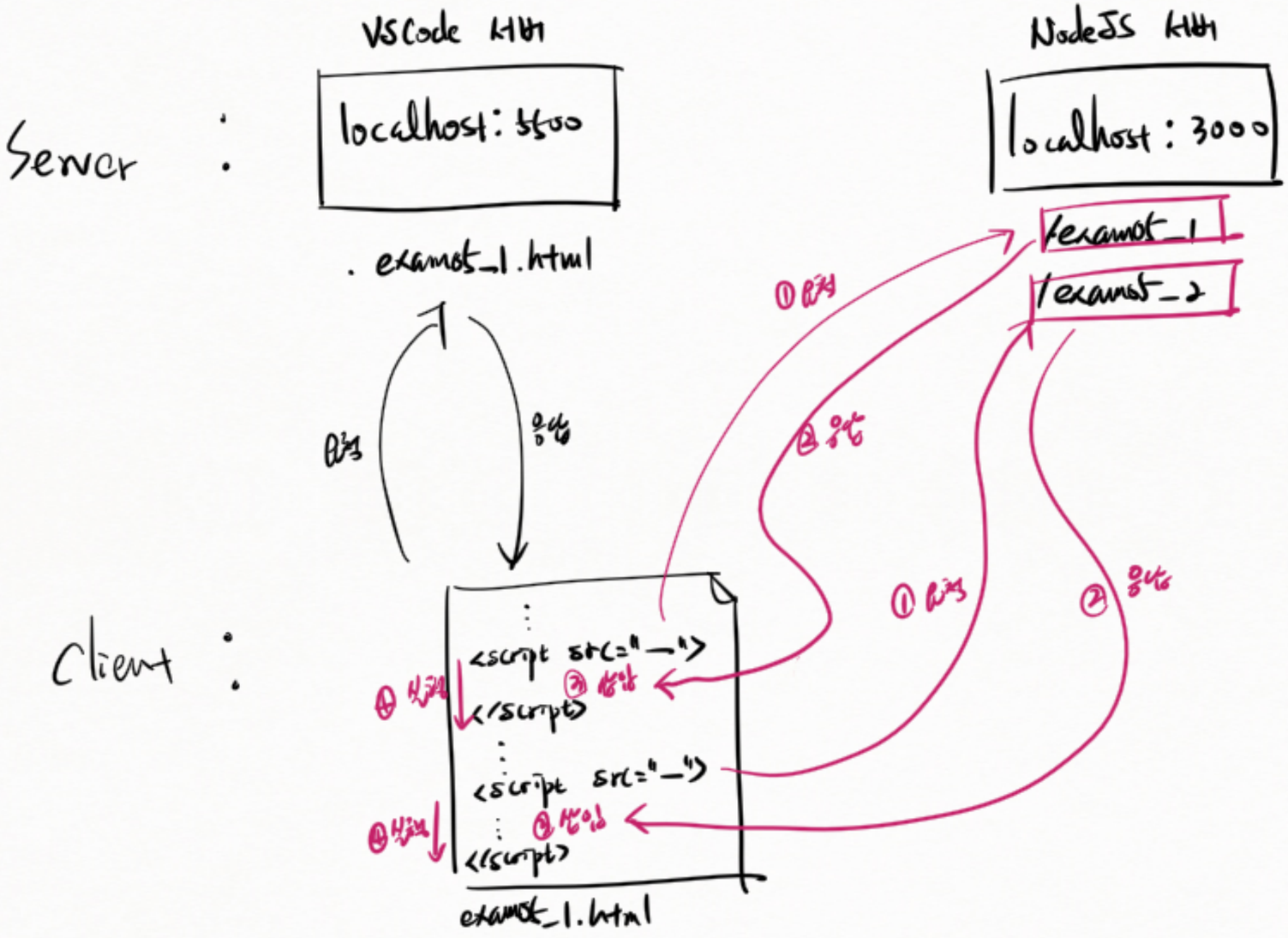
* 명령문이 잘못 작성되었을 때 바로 실행하기 전까지는 알 수 없다



* 명령문이 잘못됐으면 컴파일 과정에서 모두 걸러진다

문법 오류는 없는 상태

*

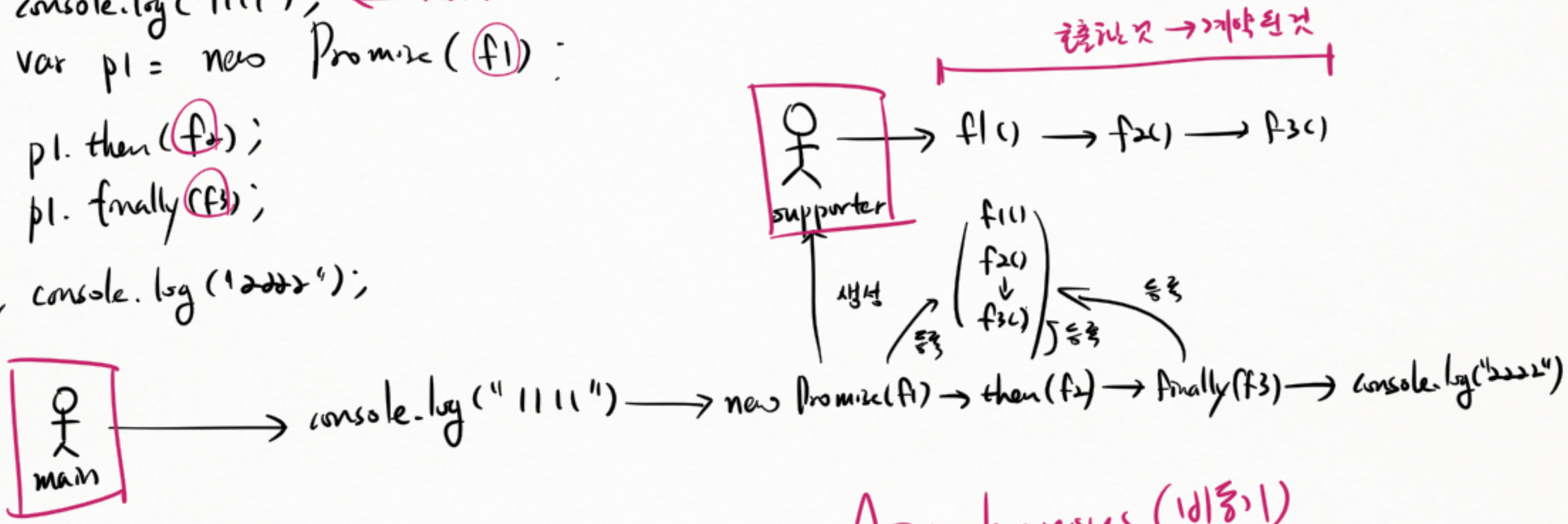

VSCode 서버
NodeJS 서버
Server :
localhost: 5500
localhost : 3000
/examot_1
/examot_2
. examot_1.html
요청
응답
① 요청
② 응답
① 요청
② 응답
Client :
<script src="—">
③ 생성
④ 실행
</script>
<script src="—">
⑤ 생성
⑥ 실행
</script>
examot_1.html

* Promise

```
console.log("1111");   ← main
var p1 = new Promise(f1);

p1.then(f2);
p1.finally(f3);

console.log("2222");
```



Asynchronous (비동기)

* var 변수와 블록

var

① global 변수

↳ window 객체에 소속됨

② 함수 변수

↳ 함수를 실행하는 동안 생성

↓

호출이 끝나면 삭제

+ let 변수 와 블록

↓

블록 영역의 종속 : 블록에 진입 → 생성

블록에서 나갈 때 → 소멸

```
let a;
{
  let a = 100;
}
```

같지 않다.

* const 변수

~~const v1;~~ 오류

변수 선언 + 값 할당

const v1 = "hello";

"initializer"

변수 선언 시에 값 설정 불가능

* 변수 사용 우선순위

① const

↓

② let

↓

③ var

```
const obj = {
    name : "홍길동",
    age : 20
    tel : "010-1111-2222"
};
```

obj ≠ {...} → 200

상수는 메모리의 값을 바꿀 수 없다

200

k	v

200

k	v
name	홍길동
age	20
tel	"010-"

```
obj.name = "김개똥"
```

obj 변수를 바꾸는 것이 아니라!

* 배열 만들기

↳ 여러 개의 변수를 손쉽게 만드는 문법

```
var arr = new Array();
```

arr

200

배열 객체의 값

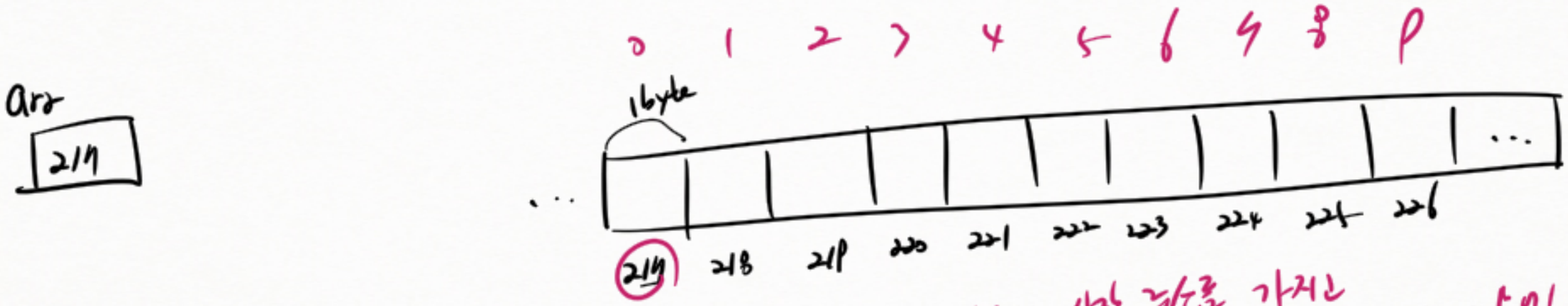
length push ⋮	4 function() {-} — —
0	"홍길동"
1	"임꺽정"
2	"유비"
3	"안중근"

① new
⇓
빈 객체 생성

② Array()
⇓
배열 관리에 필요한 변수, 함수 추가

```
arr.push("홍길동");
arr.push("임꺽정");
arr.push("유비");
arr.push("안중근");
```

* 인덱스 0 과 메모리 주소



메모리 시작 주소를 가지고
배열의 항목을 가리키기 때문에
인덱스는 0부터 시작한다.

첫번째 항목 → 214 = 214 + 0 → arr[0]

두번째 〃 → 218 = 214 + 1 → arr[1]

세번째 〃 → 224 = 214 + 7 → arr[7]

⋮