

**Notebook**

## 0.0 Section

---

Chapter

### 0.0.0 Subtitle    Location: x-1.3, y-1.3

Keyword In English	Sample Page

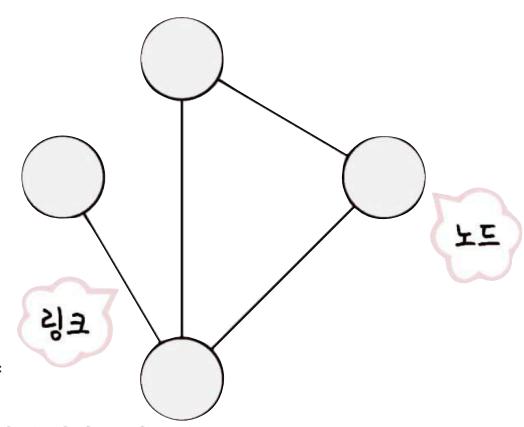
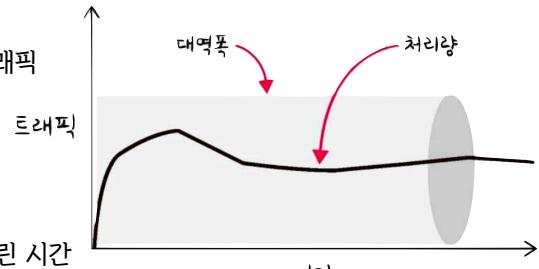
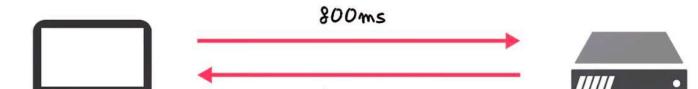
## Chapter 2 네트워크

---

## 2.1 네트워크의 기초

### Section 2.1 네트워크의 기초

#### 2.1.1 처리량과 지연 시간

네트워크	컴퓨터 등의 장치들이 통신 기술을 이용해 구축하는 연결망
노드	서버, 라우터, 스위치 등의 네트워크 장치
링크 link	유선/무선  [좋은 네트워크란?] <ul style="list-style-type: none"><li>- 많은 처리량 처리 가능</li><li>- 지연 시간 ↓</li><li>- 장애 빈도 ↓</li><li>- 좋은 보안을 갖춤</li></ul>
처리량 throughput	링크를 통해 전달되는 단위 시간당 데이터양  단위: bps(bits per second)-초당 전송, 초당 수신되는 비트 수  [처리량에 영향을 주는 요소] <ul style="list-style-type: none"><li>- 사용자들이 많이 접속할 때마다 커지는 트래픽</li><li>- 네트워크 장치 간의 대역폭</li><li>- 네트워크 중간에 발생하는 에러</li><li>- 장치의 하드웨어 스펙</li></ul>
지연시간 latency	요청이 처리되는 시간 어떤 메세지가 두 장치 사이를 왕복하는데 걸린 시간  [지연시간에 영향을 주는 요소] <ul style="list-style-type: none"><li>- 매체 타입(무/유선)</li><li>- 패킷 크기</li><li>- 라우터의 패킷 처리 시간</li></ul>   

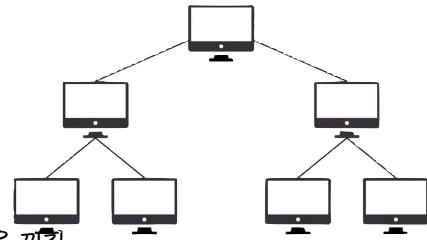
$$\text{지연 시간} = 800\text{ms} + 900\text{ms} = 1.7\text{s}$$

## 2.1.2 네트워크 토플로지와 병목 현상

### 네트워크 토플로지 network topology

#### - 트리 토플로지 tree topology

노드와 링크가 배치되어 있는 방식, 연결 형태



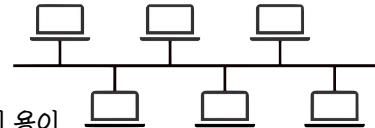
계층형 토플로지  
트리 형태

[장점] 노드의 추가 및 삭제 용이

[단점] 특정 노드에 트래픽 집중 시 하위 노드에 영향을 끼침

#### - 버스 토플로지 bus topology

중앙 통신 회선 하나에 여러 개의 노드가 연결되어 공유



[사용처] 근거리 통신망(LAN)

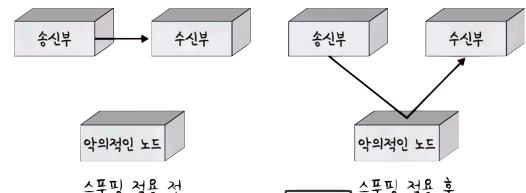
[장점] 설치 비용 ↓, 신뢰성 ↑, 통신 회선에 노드 추가 및 삭제 용이

[단점] 스폐핑 가능

#### ·스포핑 spoofing

LAN 상에서 송신부의 패킷을 송신과 관련 없는 다른 호스트에 가지 않도록 하는

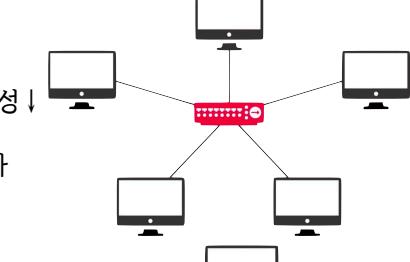
스위칭 기능을 마비시키거나 속여서  
특정 노드에 해당 패킷이 오도록 처리



#### - 스타 토플로지 star topology

성형 토플로지

중앙에 있는 노드에 모두 연결



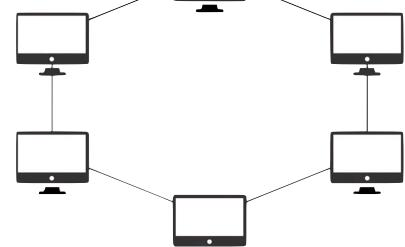
[장점] 노드 추가 및 에러 탐지 용이, 패킷 충돌 발생 가능성 ↓  
노드에 장애가 발생해도 다른 노드에 영향 ↓

[단점] 중앙 노드에 장애 발생 시 전체 네트워크 사용 불가  
설치 비용이 고가

#### - 링형 토플로지 ring topology

각각의 노드가 양 옆의 두 노드와 연결

전체적으로 고리처럼 하나의 연속된 길을 통해 통신



[장점] 노드 수가 증가되어도 네트워크상의 손실 거의 X  
충돌 발생 가능성 ↓, 노드 고장 발견 용이

[단점] 네트워크 구성 변경 복잡  
회선에 장애 발생 → 전체 네트워크에 영향 BIG

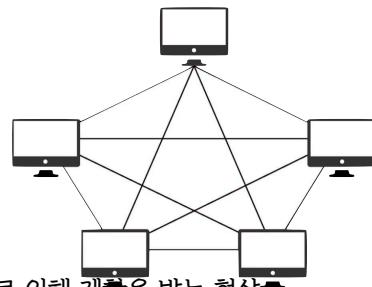
표로 토플로지 특징, 장  
점, 그림 나열

- 메시 토플로지  
mesh topology

## 망형 토플로지

- [장점] 여러 개의 경로 존재
  - 한 단말 장치에 장애가 발생해도 네트워크를 계속 사용 가능
  - 트래픽의 분산 처리 가능

[단점] 노드 추가 복잡, 구축 비용 & 운용 비용 ↑

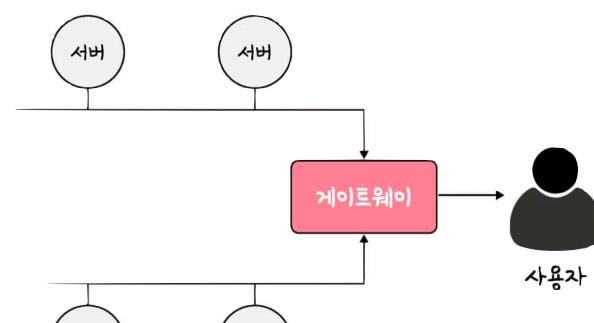


병목 현상  
bottleneck

전체 시스템의 성능이나 용량이 하나의 구성요소로 인해 제한을 받는 현상

Ex) 서비스에서 이벤트를 열었을 때 트래픽이 많이 생기고

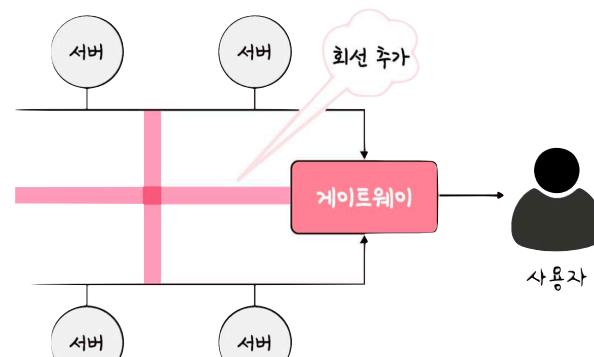
그 트래픽을 잘 관리하지 못해 병목 현상이 생겨 사용자가 웹 사이트로 들어가지 못함



병목현상이 일어나, 사용자가 서비스를 이용할 때 지연시간이 길게 발생하고 있다고 가정

관리자가 지연 시간을 줄이기 위해 대역폭을 크게 설정했음에서 성능이 개선되지 않았음

네트워크의 토플로지 확인 ↓



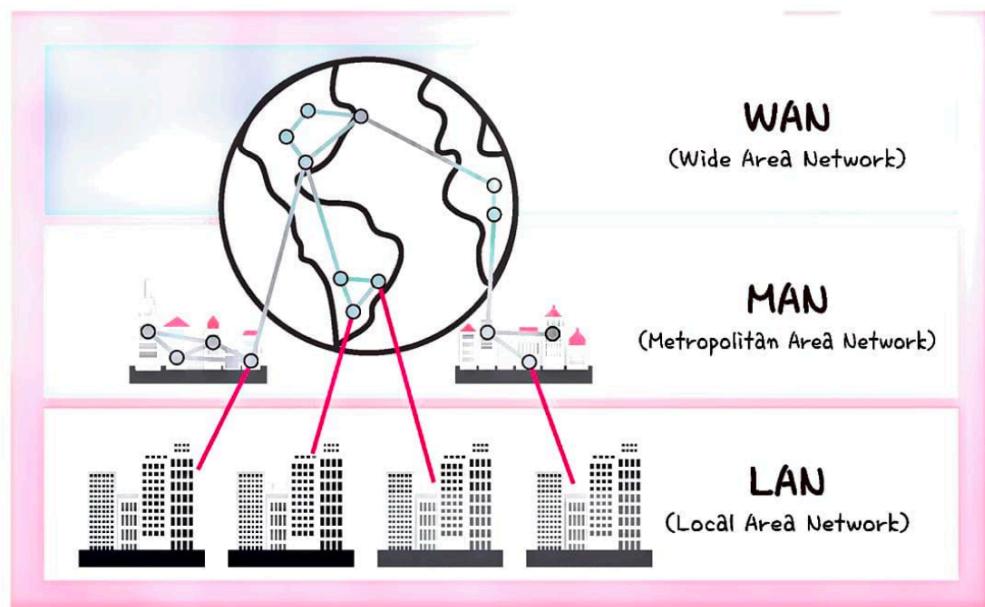
서버와 서버 간 · 게이트웨이로 이어지는 회선을 추가해 병목현상을 해결

↑ 어떤 경로로 이루어져 있는지 알아야 병목현상을 올바르게 해결 가능

.: 토플로지 → 병목 현상을 찾을 때의 중요한 기준

### 2.1.3 네트워크 분류

규모를 기반으로 분류



**LAN**  
Local Area Network

근거리 통신망: 좁은 공간에서 운영  
Ex) 같은 건물, 캠퍼스, 사무실 등

[전송 속도] 빠름  
[혼잡도] 혼잡하지 않음

**MAN**  
Metropolitan Area Network

대도시 지역 네트워크: 서울 등 도시 같은 넓은 지역에서 운영  
[전송속도] 보통  
[혼잡도] LAN 보다는 더 많이 혼잡

**WAN**  
Wide Area Network

광역 네트워크: 국가 또는 대륙 같은 더 넓은 지역에서 운영  
[전송속도] 낮음  
[혼잡도] MAN보다 더 혼잡

#### 2.1.4 네트워크 성능 분석 명령어

<b>netstat</b> <i>Packet INternet Groper</i>	<p>코드상 문제가 없는데 사용자가 서비스로부터 데이터를 가져오지 못하는 상황 → 병목현상 가능성</p> <p>![병목현상의 주된 원인]</p> <ul style="list-style-type: none"> <li>- 네트워크 대역폭</li> <li>- 네트워크 토플로지</li> <li>- 서버 CPU, 메모리 사용량</li> <li>- 비효율적인 네트워크 구성</li> </ul>
<b>ping</b> <i>Packet INternet Groper</i>	<p>네트워크로부터 발생한 문제점인지 확인하는 방법: 네트워크 성능 분석 명령어를 통해 네트워크와 관련된 테스트 및 관련 없는 테스트 시행</p>
<b>nslookup</b>	<p>네트워크 상태를 확인하려는 대상 노드를 향해 일정 크기의 패킷을 전송하는 명령어</p> <p>[획득 정보]</p> <ul style="list-style-type: none"> <li>- 해당 노드의 패킷 수신 상태</li> <li>- 노드에 도달하기까지의 시간</li> <li>- 노드의 네트워크 연결 상태</li> </ul> <p>[테스팅 불가능한 경우]</p> <ul style="list-style-type: none"> <li>- ICMP 프로토콜을 지원하지 않는 기기(ping은 TCP/IP 프로토콜 중 ICMP 프로토콜을 통해 동작)</li> <li>- 네트워크 정책상 ICMP나 traceroute를 차단하는 대상</li> </ul> <p>[실행 예시]</p> <pre>-n 12: 12번의 패킷을 보내고 받음 접속 중인 서비스들의 네트워크 상태를 표시</pre> <div style="border: 1px solid black; padding: 10px; margin-top: 10px;"> <pre>C:\Users\jhc&gt;ping www.google.com -n 12  Ping www.google.com [172.217.26.228] 32바이트 데이터 사용: 172.217.26.228의 응답: 바이트=32 시간=56ms TTL=117 172.217.26.228의 응답: 바이트=32 시간=56ms TTL=117 172.217.26.228의 응답: 바이트=32 시간=57ms TTL=117 172.217.26.228의 응답: 바이트=32 시간=56ms TTL=117 172.217.26.228의 응답: 바이트=32 시간=57ms TTL=117 172.217.26.228의 응답: 바이트=32 시간=57ms TTL=117 172.217.26.228의 응답: 바이트=32 시간=56ms TTL=117  172.217.26.228에 대한 Ping 통계:   패킷: 보냄 = 12, 받음 = 12, 손실 = 0 (0% 손실),   왕복 시간(밀리초):     최소 = 56ms, 최대 = 57ms, 평균 = 56ms</pre> </div>

[실행 예시]]

```
C:\Users\jhc>netstat
```

활성 연결

프로토콜	로컬 주소	외부 주소	상태
TCP	121.165.224.223:6881	220.118.188.195:41519	TIME_WAIT
TCP	121.165.224.223:49245	211.115.106.72:http	CLOSE_WAIT
TCP	121.165.224.223:50124	nrt12s51-in-f19:https	ESTABLISHED
TCP	121.165.224.223:50278	118.223.101.233:56517	ESTABLISHED
TCP	121.165.224.223:52025	211.115.106.207:http	CLOSE_WAIT
TCP	121.165.224.223:52042	211.115.106.207:http	CLOSE_WAIT
TCP	121.165.224.223:52043	211.115.106.207:http	CLOSE_WAIT
TCP	121.165.224.223:52220	211.249.220.83:https	ESTABLISHED
TCP	121.165.224.223:52221	104.21.37.168:http	ESTABLISHED
TCP	121.165.224.223:52243	a104-74-192-17:http	TIME_WAIT

DNS에 관련된 내용 확인

특정 도메인에 매핑된 IP 확인

[실행 예시]]

```
C:\Users\jhc>nslookup
기본 서버:  kms.kornet.net
Address:  168.126.63.1

> google.com
서버:  kms.kornet.net
Address:  168.126.63.1

권한 없는 응답:
이름:  google.com
Addresses:  2404:6800:4004:820::200e
           172.217.31.174
```

tracert 리눅스의 traceroute	목적지 노드까지 네트워크 경로 확인: 어느 구간에서 응답 시간이 느려지는지 확인 가능 [실행 예시]  C:\Users\jhc>tracert www.google.com 최대 30홉 이상의 www.google.com [142.250.199.100](으)로 가는 경로 추적:  1 1 ms * * 121.165.224.254 2 1 ms 1 ms 1 ms 61.78.42.172 3 2 ms 2 ms 1 ms 112.189.31.209 4 * * * 요청 시간이 만료되었습니다. 5 1 ms 2 ms 1 ms 112.174.47.102 6 41 ms 40 ms 41 ms 72.14.209.102 7 36 ms 36 ms 37 ms 108.170.241.80 8 43 ms 43 ms 41 ms 216.239.62.240 9 53 ms 53 ms 53 ms 172.253.50.221 10 56 ms 56 ms 56 ms 216.239.49.194
ftp	대형 파일을 전송해 테스팅
tcpdump	노드로 오고 가는 패킷 캡처
네트워크 분석 프로그램	wireshark, netmon

## 2.1.5 네트워크 프로토콜 표준화

### 네트워크 프로토콜

다른 장치들끼리 데이터를 교환하기 위해 설정된 공통 인터페이스

개인이나 기업이 아닌 표준화 단체(IEEE 또는 IETF)가 결정

#### - IEEE802.3



#### IEEE 802.3 ETHERNET WORKING GROUP

- The IEEE 802.3 Working Group develops standards for Ethernet networks. We have a number of active projects, study groups, and ad hoc as listed below:
  - IEEE P802.3ck 100 Gb/s, 200 Gb/s, and 400 Gb/s Electrical Interfaces Task Force.
  - IEEE P802.3cs Increased-reach Ethernet optical subscriber access (Super-PON) Task Force.
  - IEEE P802.3cw 400 Gb/s over DWDM systems Task Force.
  - IEEE P802.3cx Improved PTP Timestamping Accuracy Task Force.
  - IEEE P802.3cy Greater than 10 Gb/s Electrical Automotive Ethernet Task Force.
  - IEEE P802.3cz Multi-Gigabit Optical Automotive Ethernet Task Force.
  - IEEE P802.3da 10 Mb/s Single Pair Multidrop Segments Enhancement Task Force.
  - IEEE P802.3db 100 Gb/s, 200 Gb/s, and 400 Gb/s Short Reach Fiber Task Force.
  - IEEE P802.3 (IEEE 802.3dc) Revision to IEEE Std 802.3-2018 Maintenance #16 Task Force.

유선 LAN 프로토콜: 유선으로 LAN을 구축할 때 쓰이는 프로토콜

어떤 기업이 이것을 기반으로 만들면 다른 장치라도 서로 데이터 수신 가능

Ex) HTTP

웹 접속 시 사용

HTTP라는 프로토콜을 통해 노드들이 웹 서비스를 기반으로 데이터 교환 가능

## 2.2 TCP/IP 4계층 모델

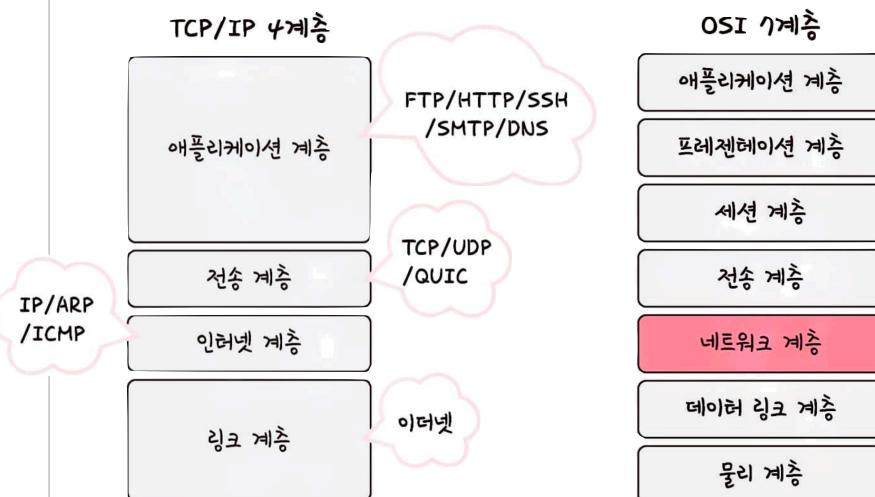
### Section 2.2 TCP/IP 4계층 모델

#### 2.2.1 계층 구조

##### 인터넷 프로토콜 스위트 internet protocol suite

인터넷에서 컴퓨터들이 정보를 교환하는데 사용되는 프로토콜 집합

[TCP/IP 4계층 및 각 계층을 대표하는 스택과 OSI 7계층 비교]



TCP/IP 계층과 달리 OSI 계층은 애플리케이션 계층을 세 개로 쪼개고 링크 계층을 데이터 링크 계층, 물리 계층으로 나누어 표현

[계층] 특정 계층이 변경되었을 때 다른 계층이 영향을 받지 않도록 설계됨

Ex) 전송 계층에서 TCP를 UDP로 변경한다고 인터넷 웹 브라우저를 다시 설치하는 것은 아님

##### 애플리케이션 계층 application

웹 서비스, 이메일 등 실질적으로 사람들에게 제공하는 층

[사용되는 응용프로그램]

- FTP: 장치 상호간 파일 전송 시 사용되는 표준 통신 프로토콜
- HTTP: World Wide Web을 위한 데이터 통신의 기초이자 웹 사이트 이용 시 사용되는 프로토콜
- SSH: 보안X 네트워크에서 네트워크 서비스를 안전하게 운영하기 위한 암호화 네트워크 프로토콜
- SMTP: 전자 메일 전송을 위한 인터넷 표준 통신 프로토콜
- DNS: 도메인 이름 및 IP 주소 매핑 서버

Ex) www.naver.com에 DNS 쿼리 발생

[Root DNS] → [.com DNS] → [.naver DNS] → [.www DNS] 과정으로

완벽한 주소를 찾아 IP 주소 매핑

→ IP 주소가 바뀌어도 사용자들에게 똑같은 도메인 주소로 서비스 가능

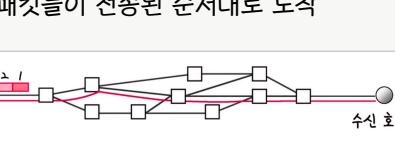
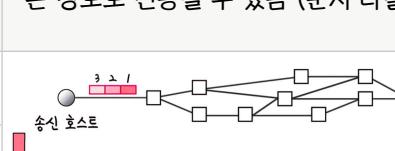
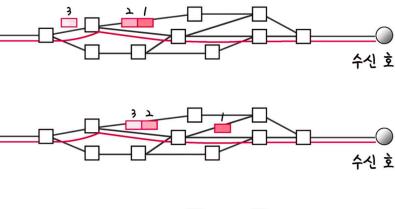
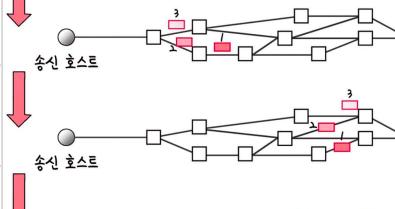
Ex) www.naver.com의 IP 주소가 222.111.222.111에서 222.111.222.122로 변경  
되어도 똑같은 www.naver.com라는 주소로 서비스 가능

송신자와 수신자를 연결

연결 지향 데이터 스트림 지원  
신뢰성, 흐름제어 제공

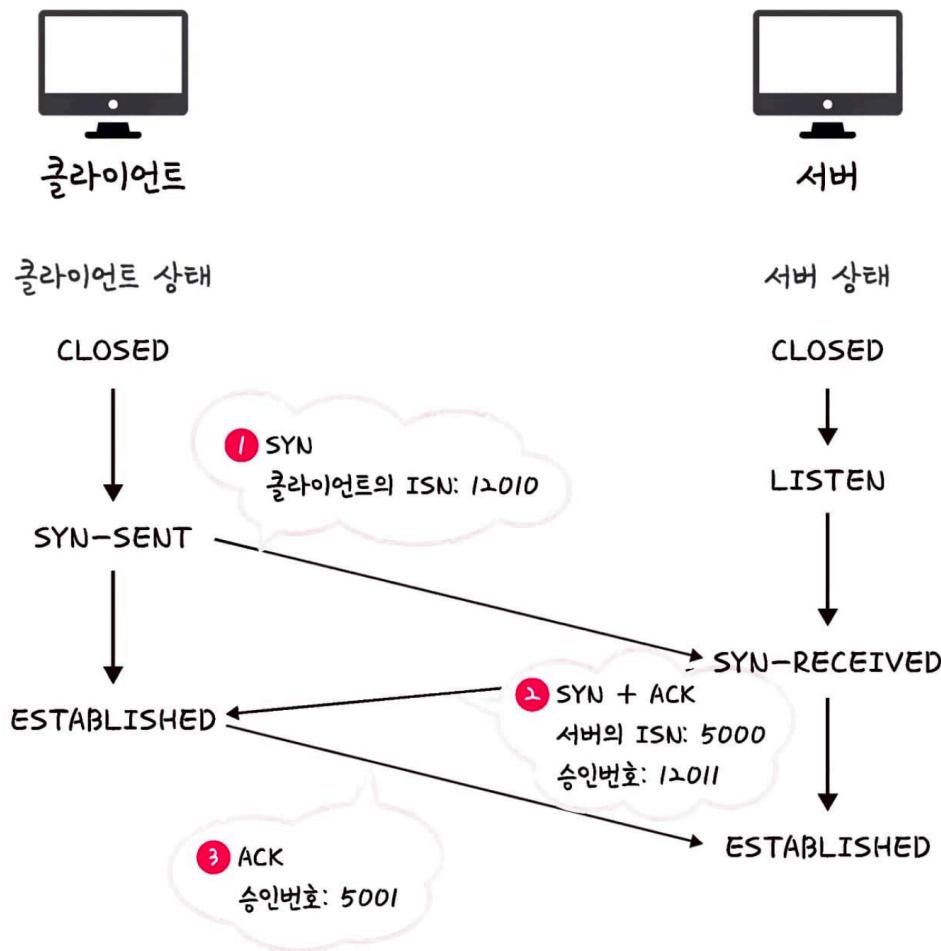
##### 전송계층 transport

	패킷 사이의 순서 보장	수신 여부 확인	패킷 교환 방식
TCP	O	O → 연결지향 프로토콜 사용해 신뢰성 확보	가상 회선
UDP	X	X	데이터그램

가상회선 패킷 교환 방식	데이터그램 패킷 교환 방식
<p>각 패킷에 가상회선 식별자 존재 모든 패킷을 전송하면 가상회선 해제 패킷들이 전송된 순서대로 도착</p> 	<p>패킷이 독립적으로 존재 → 최적의 경로 선택 하나의 메시지에서 분할된 여러 패킷은 서로 다른 경로로 전송될 수 있음 (순서 다를 수 있음)</p> 
	

- TCP 연결 성립 과정

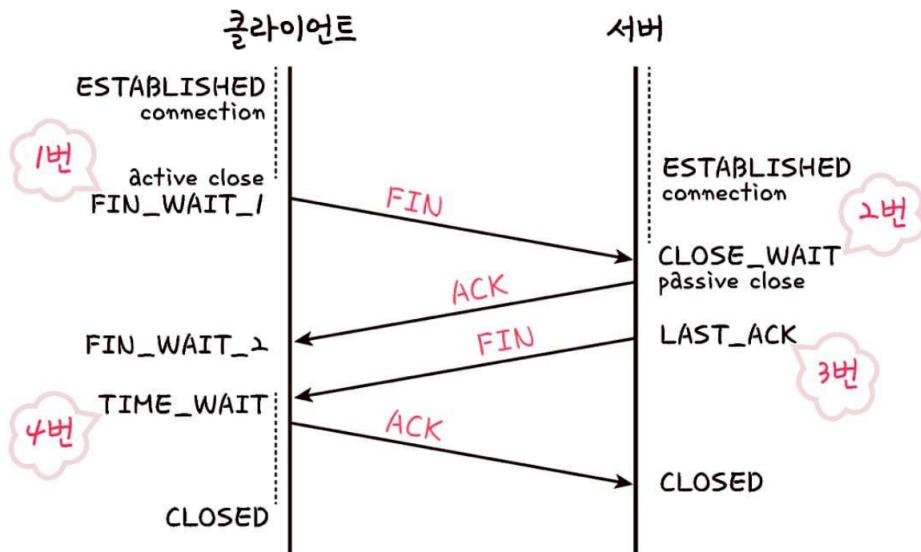
[신뢰성 확보 작업] 3-웨이 핸드셰이크 3-way handshake



1. SYN 단계: 클라이언트가 서버에 클라이언트의 ISN을 담아 SYN을 보냄
2. SYN + ACK 단계: 서버가 클라이언트의 SYN 수신 → 서버의 ISN을 보냄(승인번호로 클라이언트의 ISN + 1을 보냄)
3. ACK 단계: 클라이언트가 서버의 ISN + 1한 값인 승인번호를 담아 ACK를 서버에 보냄

- TCP 연결 해제 과정

4-웨이 핸드쉐이크 4-way handshake 과정 발생



1번: 클라이언트가 연결을 닫으려고 할 때 FIN으로 설정된 세그먼트 전송  
클라이언트: FIN\_WAIT\_1 상태 (서버 응답 대기)

2번: 서버가 클라이언트로 ACK라는 승인 세그먼트 전송  
서버: CLOSE\_WAIT 상태  
클라이언트: 세그먼트를 받으면 FIN\_WAIT\_2 상태

3번: 서버가 ACK 전송 → 클라이언트에 FIN이라는 세그먼트 전송

4번: 클라이언트: TIME\_WAIT 상태, 서버로 ACK 전송  
서버: CLOSED 상태  
클라이언트가 일정 시간 대기 후 연결이 닫히고 클라이언트-서버의 모든 자원 연결 해제

[TIME\_WAIT의 의의] 일정 시간을 두고 연결을 닫는 이유

1. 지연 패킷의 가능성에 대비: 패킷이 뒤늦게 도달하고 처리하지 못하면 데이터 무결성 문제 발생
2. 두 장치의 연결이 닫혔는지 확인: LAST\_ACK 상태에서 닫히면 새로운 연결을 시도 할때도 장치는 여전히 LAST\_ACK 상태가 되어 접속오류 발생

## 인터넷 계층 internet

장치로부터 받은 네트워크 패킷을 IP 주소로 지정된 목적지로 전송하기 위해 사용되는 계층

패킷을 수신할 상대의 주소를 지정해 데이터 전달

비연결형적 특징: 상대방이 제대로 받았는지에 대해 보장하지 않음

[종류] IP, ARP, ICMP

전선, 광섬유, 무선 등으로 실질적으로 데이터를 전달 및 장치 간 신호 교환 규칙을 정하는 계층

## 링크 계층 = 네트워크 접근 계층

물리 계층: 무선 LAN과 유선 LAN을 통해 0과 1로 이루어진 데이터를 보내는 계층

데이터 링크 계층: 이더넷 프레임을 통해 에러 확인, 흐름 제어, 접근 제어 담당 계층

유선 LAN을 이루는 이더넷은 IEEE802.3이라는 프로토콜을 따름  
전이중화 통신 사용

### - 유선 LAN(IEEE802.3)

[전이중화 통신full duplex]

양쪽 장치가 동시에 송수신 가능

송신로와 수신로로 나눠서 데이터 교환

현대의 고속 이더넷이 통신하는 방식

[CSMA/CDCarrier Sense Multiple Access with Collision Detection]

유선 LAN 반이중화 통신 방법 중 하나

데이터를 보낸 이후에 충돌 발생 시 일정 시간 이후 재전송

수신로와 송신로가 나뉘어져 있는 것이 아니기 때문에 충돌 발생 시 처리할 수 있어야 함

[TP 케이블twisted pair cable]

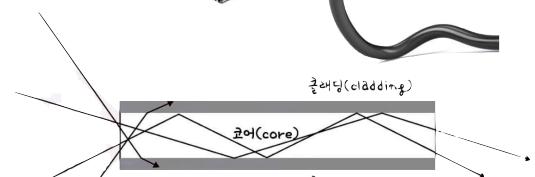
하나의 케이블처럼 보이지만

실제로는 여덟 개의 구리선을 두 개씩 꼬아서 묶은 케이블



구리선의 실드처리 O - UTP; LAN 케이블

구리선의 실드처리 X - STP



[광섬유 케이블]

레이저를 이용해 통신 →

구리선과 비교도 되지 않는 장거리 및 고속 통신 가능

외부와 내부가 다른 밀도를 가지고 제작(유리·플라스틱)

한 번 들어간 빛이 계속적으로 반사하며 전진하여 끝까지 도달

- 무선 LAN(IEEE802.11)  
WLAN

수신과 송신에 같은 채널 사용 - 반이중화 통신

[반이중화 통신 half duplex]

- 양쪽 장치가 동시에 송수신 불가능; 한 번에 한 방향만 통신 가능
- 장치가 신호를 수신하기 시작하면 응답하기 전에 전송이 완료될 때까지 대기
- 둘 이상의 장치가 동시에 전송 → 충돌 → 메세지 손실·왜곡: 충돌방지 시스템 필요

[CSMA/CA]

무선 LAN 반이중화 통신 방법 중 하나

데이터 송신 전에 캐리어 감지 등으로 충돌 방지

〈과정〉

1. 데이터 송신 전, 무선 매체를 살핌
2. 캐리어 감지: 회선이 비어있는지 판단
3. IFS Inter Frame Space: 랜덤 값을 기반으로 일정 시간 대기  
무선 매체 사용 중 → 대기 시간을 늘려가며 대기
4. 데이터 송신

- 무선 LAN을 이루는 주파수

무선 LAN은 무선 신호 전달 방식을 이용해 2대 이상의 장치를 연결하는 기술  
비유도 매체인 공기애 주파수를 쏘아 무선 통신망 구축

〈주파수 대역〉

- 2.4GHz: 장애물에 강하지만 전자레인지 등 전파 간섭 ↑
- 5GHz: 이용 가능한 채널 수 ↑, 동시에 사용 가능 → 상대적으로 깨끗한 전파 환경 구축 가능

[와이파이 wifi]

전자기기들이 무선 LAN 신호에 연결할 수 있게 하는 기술

무선 접속 장치(AP Access Point) 필요; 공유기

유선 LAN에 흐르는 신호 → 무선 LAN 신호; 신호가 닿는 범위 내에서 무선 인터넷 사용 가능  
이 외에도 지그비, 블루투스가 있음

[BSS Basic Service Set] 기본 서비스 집합

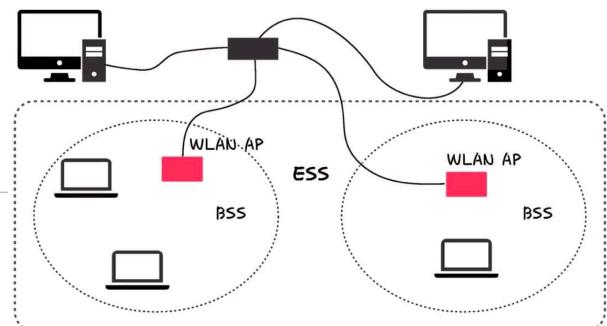
공유기를 통해 네트워크에 접속 + 근거리 무선 통신 제공

하나의 AP 기반: 사용자가 한 곳에서 다른 곳으로 자유롭게 이동하며 네트워크 접속 불가

[ESS Extended Service Set] 하나 이상의 연결된 BSS 그룹

공유기를 통해 네트워크에 접속 + 장거리 무선 통신 제공

BSS보다 더 많은 가용성과 이동성 제공: 사용자가 장소를 이동하며 중단 없이 네트워크에 연결



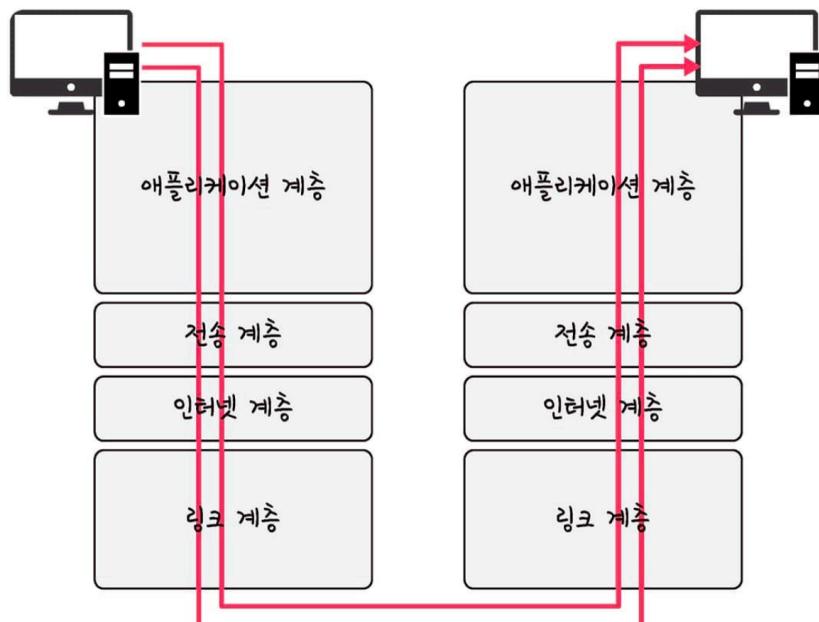
- 이더넷 프레임 데이터 링크 계층은 이더넷 프레임을 통해 전달받은 데이터의 에러 검출 및 캡슐화

[구조]



- Preamble: 이더넷 프레임 시작 알림
- SFD Start Frame Delimiter: 다음 바이트부터 MAC 주소 필드 시작임을 알림
- DMAC, SMAC: 수신, 송신 MAC 주소
- EtherType: 데이터 계층 위의 계층인 IP 프로토콜 정의 (Ex) IPv4·IPv6 )
- Payload: 전달받은 데이터
- CRC: 에러 확인 비트

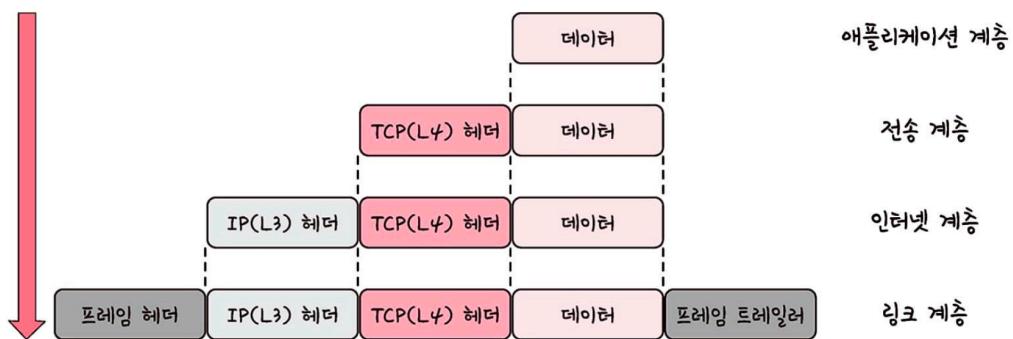
계층 간  
데이터 송수신 과정



한 컴퓨터에서 캡슐화 과정을 거쳐 다른 컴퓨터로 통신을 하고  
해당 컴퓨터에서 비캡슐화를 거쳐 데이터를 전송받음

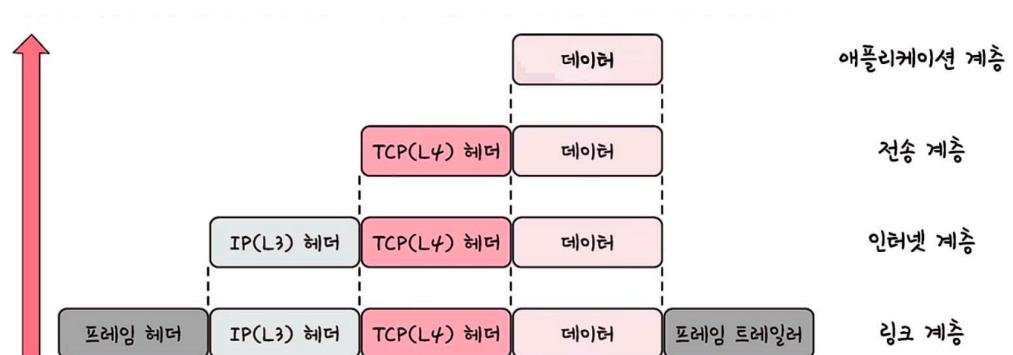
## 캡슐화 과정

상위 계층의 헤더와 데이터를 하위 계층의 데이터 부분에 포함시키고 해당 계층의 헤더 삽입



애플리케이션 계층 → 전송 계층: '세그먼트' 또는 '데이터그램'화 + TCP(L4) 헤더 추가  
 전송 계층 → 인터넷 계층: IP(L3) 헤더 추가 → '패킷'화  
 인터넷 계층 → 링크 계층: 프레임 헤더 + 프레임 트레일러 추가 → '프레임'화

## 비캡슐화 과정



캡슐화된 데이터를 받아 프레임화된 데이터 → 패킷화 → 세그먼트, 데이터그램화 → 메세지화  
 최종적으로 사용자에게 애플리케이션의 PDU인 메세지로 전달

## 2.2.2 PDU Protocol Data Unit

네트워크의 어떠한 계층에서 데이터가 전달될 때 한 덩어리의 단위

[구성]

- 헤더: 제어 관련 정보를 포함
- 페이로드: 데이터

[계층마다 부르는 명칭]

- 애플리케이션 계층: 메세지
- 전송 계층: 세그먼트(TCP), 데이터그램(UDP)
- 인터넷 계층: 패킷
- 링크 계층: 프레임(데이터 링크 계층), 비트(물리 계층)

cf. 비트로 송수신하는 것이 모든 PDU 중 가장 속도와 효율 ↑

but 애플리케이션 계층에서는 문자열 기반으로 송수신  
; 헤더에 authorization 값 등 다른 값들을 넣는 확장이 쉽기 때문

## 2.3 네트워크 기기

### Section 2.3 네트워크 기기

#### 2.3.1 네트워크 기기의 처리 범위

##### 계층별 처리 기기

네트워크 기기는 계층별로 처리 범위가 나누어져 있음

- 애플리케이션 계층: L7 스위치
- 인터넷 계층: 라우터, L3 스위치
- 데이터 링크 계층: L2 스위치, 브리지
- 물리 계층: NIC, 리피터, AP

상위 계층을 처리하는 기기는 하위 계층 처리 가능 (반대는 불가능)

Ex. L7 스위치는 모든 계층의 프로토콜을 처리할 수 있지만, AP는 물리 계층만 처리 가능

### 2.3.2 애플리케이션 계층을 처리하는 기기

L7 스위치  
= 로드밸런서

[스위치란?]  
네트워크 장비  
여러 장비를 연결 및 데이터 통신 중재  
목적지가 연결된 포트로만 전기 신호를 보내 데이터 전송

서버의 부하를 분산하는 기기: 클라이언트로부터 오는 요청들을 뒤쪽의 여러 서버로 분배

[목표] 시스템이 처리할 수 있는 트래픽 증가

[기능] URL, 서버, 캐시 쿠키를 기반으로 트래픽 분산  
바이러스 및 불필요한 외부 데이터 필터링  
응용 프로그램 수준의 트래픽 모니터링  
정기적인 헬스 체크<sup>health check</sup>  
: 장애 서버 발생 시 해당 서버를 트래픽 분산 대상에서 제외

[L4 스위치와 L7 스위치의 차이점]

스위치 종류	특이사항	트래피 분산 기반	클라우드서비스에서의 로드밸런싱
L7	상단 참조	IP, 포트, URL, HTTP 헤더, 쿠키	ALB 컴포넌트
L4	전송 계층 처리 기기 스트리밍 관련 서비스 에서는 사용 불가	IP와 포트 메세지 기반 인식 X	NLB 컴포넌트

[헬스 체크] 정상적인 서버 또는 비정상적인 서버 판별  
전송 주기 및 재전송 횟수 설정 후 반복적으로 서버에 요청을 보냄  
요청 횟수: 서버에 부하가 되지 않을 만큼 적절히  
요청 방법: TCP, HTTP  
판별 방법: 요청이 정상적으로 이루어졌을 경우 정상 서버로 판단  
Ex. TCP 요청 → 3-웨이 핸드셰이크가 정상적으로 시행됨 → 정상 서버

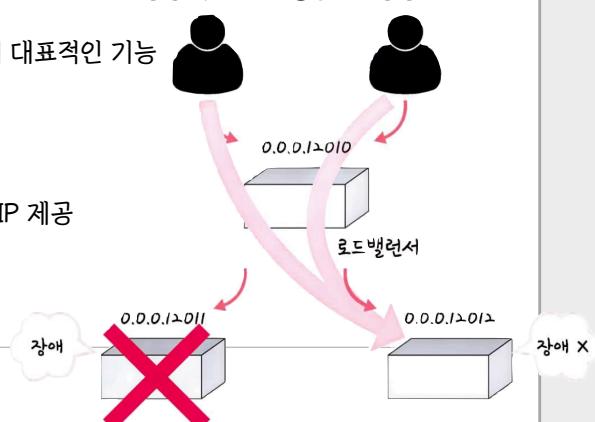
[로드밸런서를 이용한 서버 이중화] 로드밸런서의 대표적인 기능  
안정적인 서버 운용을 위해

2대 이상의 서버 구비 필수

: 1대의 서버의 에러 발생 시에도  
서비스를 안정적으로 운영하기 위함

로드밸런서 → 2대 이상의 서버 기반으로 가상 IP 제공  
→ 안정적인 서비스 제공

0.0.0.12010: 가상 IP



### 2.3.3 인터넷 계층을 처리하는 기기

라우터 router	여러 개의 네트워크를 연결·분할·구분  [라우팅] 다른 네트워크에 존재하는 장치끼리 서로 데이터를 주고받을 때 패킷 소모 최소화 및 경로 최적화 → 최소 경로로 패킷을 포워딩	
L3 스위치	L2 스위치 기능 + 라우터 기능 라우터라고 칭해도 무방 <ul style="list-style-type: none"><li>라우터 → 소프트웨어 및 하드웨어 기반 라우팅</li><li>L3 스위치 → 하드웨어 기반 라우팅</li></ul>	

### 2.3.4 데이터 링크 계층을 처리하는 기기

#### L2 스위치

장치들의 MAC 주소를 MAC 주소 테이블을 통해 관리  
연결된 장치로부터 패킷이 왔을 때 패킷 전송

IP 주소 이해 불가 → 패킷의 MAC 주소를 읽어 스위칭  
목적지가 MAC 주소 테이블에 없으면 전체 포트에 전달  
MAC 주소 테이블의 주소 → 일정 시간 이후 삭제 가능 O

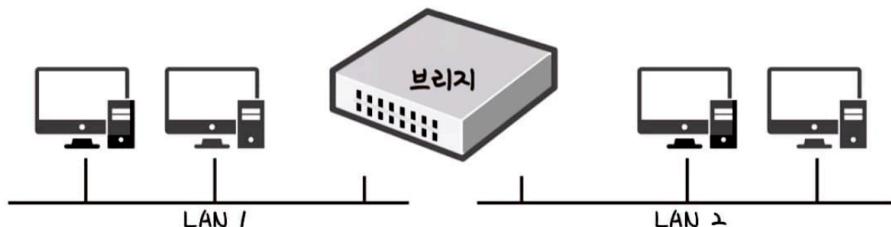
구분	L2 스위치	L3 스위치
참조 테이블	MAC 주소 테이블	라우팅 테이블
참조 PDU	이더넷 프레임	IP 패킷
참조 주소	MAC 주소	IP 주소

#### 브리지 bridge

두 개의 근거리 통신망(LAN)을 상호 접속시키는 통신망 연결 장치  
각 포트 사이의 다리 역할

장치에서 받아온 MAC 주소를 MAC 주소 테이블로 관리

[용도] 통신망 범위 확장  
서로 다른 LAN 등으로 이루어진 하나의 통신망 구축



### 2.3.5 물리 계층을 처리하는 기기

**NIC**  
Network Interface Card

= LAN 카드  
컴퓨터 내에 설치하는 확장 카드



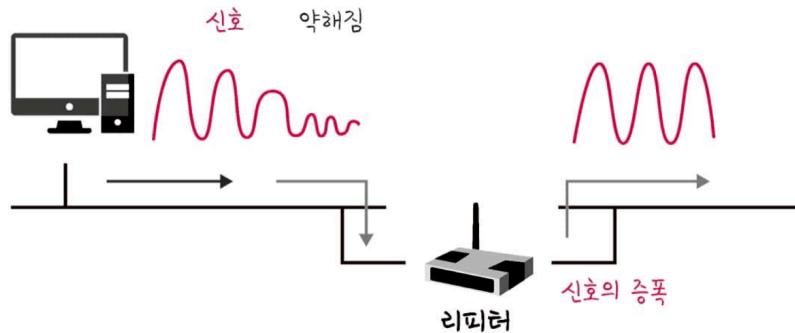
[용도] 2대 이상의 컴퓨터 네트워크 구성  
네트워크와 빠른 속도로 데이터 송수신

각 LAN 카드에는 고유의 식별번호인 MAC 주소가 있음

**리피터**  
repeater

들어오는 약해진 신호를 증폭시켜 패킷이 더 멀리 갈 수 있도록 만들어 전달하는 장치

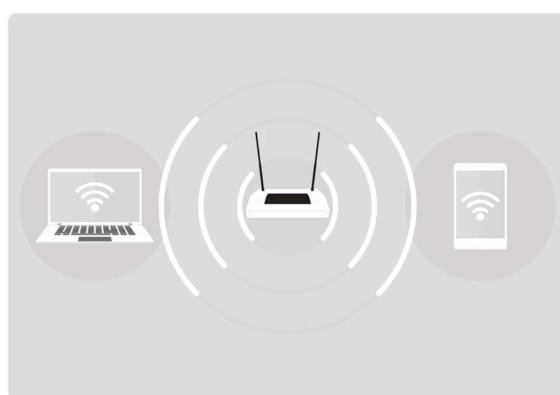
광케이블 보급에 따라 현재는 잘 쓰이지 않음



**AP**  
Access Point

패킷 복사 기기

AP에 유선 LAN 연결 → 다른 장치에서 무선 LAN 기술(wifi 등)을 사용해 무선 네트워크 연결



## 2.4 IP 주소

### Section 2.4 IP 주소

#### 2.4.1 ARPAddress Resolution Protocol

[컴퓨터와 컴퓨터 간의 통신]

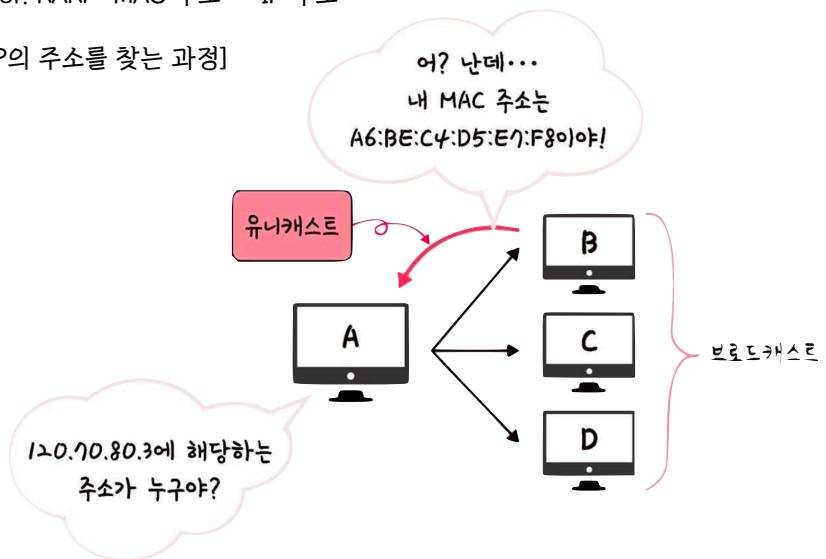
흔히 IP 주소기반 통신이라고 알려져있으나

정확히 이야기 하자면 IP 주소에서 ARP를 통해 MAC 주소를 찾아 MAC 주소를 기반으로 통신

[ARP] IP 주소로부터 MAC 주소를 구하는 다리역할 프로토콜

cf. RARP: MAC 주소 → IP 주소

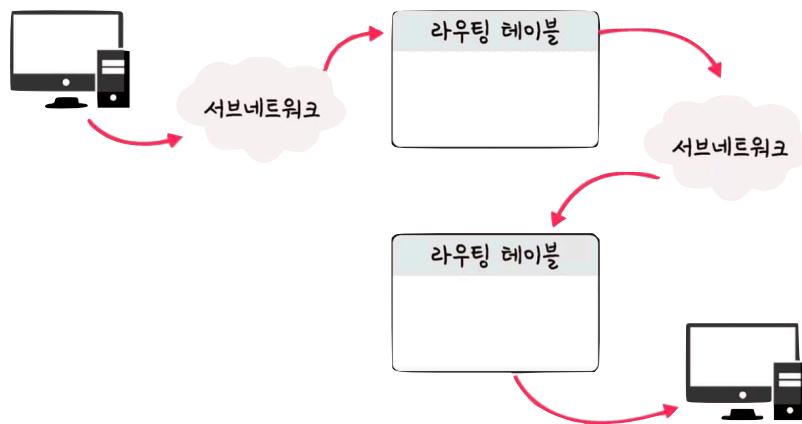
[ARP의 주소를 찾는 과정]



1. 장치 A가 ARP Request 브로드캐스트를 보내 IP 주소인 120.70.80.3를 통해 MAC 주소를 찾음
2. 해당 주소를 갖는 장치 B가 ARP reply 유니캐스트를 통해 MAC 주소 반환

## 2.4.2 홉바이홉 hop by hop 통신

IP 주소를 통해 통신하는 과정



수많은 서브네트워크 안에 있는 라우터의 라우팅 테이블 IP를 기반  
패킷 전달을 여러번 거쳐 라우팅을 수행하며 최종 목적지에 전달

라우팅 테이블  
routing table

송신지에서 수신지까지 도달하기 위해 사용

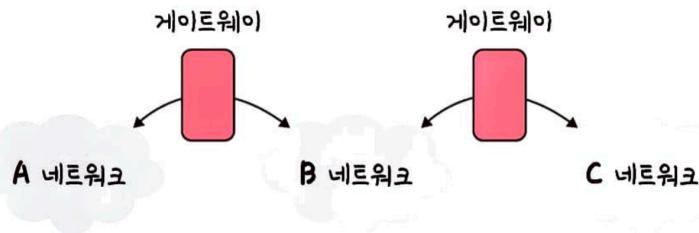
[알 수 있는 정보]

라우터에 들어가 있는 목적지 정보 & 목적지로 가기 위한 방법

게이트웨이 & 모든 목적지에 대해 해당 목적지에 도달하기 위해 거쳐야 할 다음 라우터 정보

게이트웨이  
gateway

서로 다른 통신망 및 프로토콜을 사용하는 네트워크 간의 통신을 가능하게 하는 컴퓨터 및 SW



[역할] 인터넷에 접속하기 위해 거치는 수많은 툴게이트와 같음  
서로 다른 네트워크 상의 통신 프로토콜 변환

[게이트웨이 확인 방법] → 라우팅 테이블을 통해 확인 가능 (윈도우 cmd: netstat -r)

### 2.4.3 IP 주소 체계

		IPv4	IPv6														
클래스 기반 할당 방식 classful network addressing	IP 주소 표현 방법	32비트를 8비트 단위로 점을 찍어 표기 (ex. 123.45.67.89)	64비트를 16비트 단위로 점을 찍어 표기 (ex. 2001:db8:100:42:8329)														
	사용 추세	현재 가장 많이 사용	점점 늘어나는 중														
	<p>첫 번째 바이트      두 번째 바이트      세 번째 바이트      네 번째 바이트</p> <table border="1"> <tr> <td>클래스 A</td> <td>네트워크 주소</td> <td>호스트 주소</td> </tr> <tr> <td>클래스 B</td> <td>네트워크 주소</td> <td>호스트 주소</td> </tr> <tr> <td>클래스 C</td> <td>네트워크 주소</td> <td>호스트 주소</td> </tr> <tr> <td>클래스 D</td> <td>브로드캐스트용 주소</td> <td></td> </tr> <tr> <td>클래스 E</td> <td>예비용 주소</td> <td></td> </tr> </table>	클래스 A	네트워크 주소	호스트 주소	클래스 B	네트워크 주소	호스트 주소	클래스 C	네트워크 주소	호스트 주소	클래스 D	브로드캐스트용 주소		클래스 E	예비용 주소		
클래스 A	네트워크 주소	호스트 주소															
클래스 B	네트워크 주소	호스트 주소															
클래스 C	네트워크 주소	호스트 주소															
클래스 D	브로드캐스트용 주소																
클래스 E	예비용 주소																

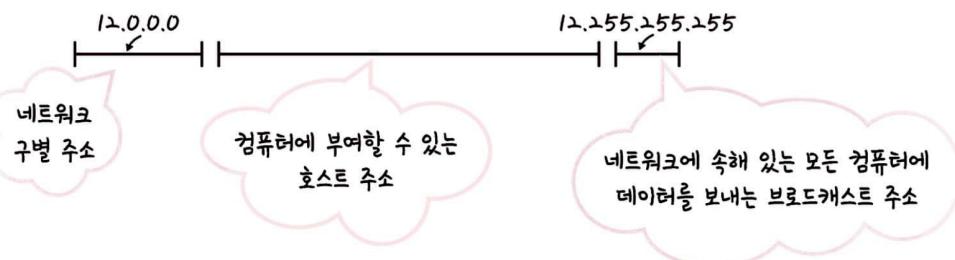
[클래스 기반 할당 방식 상세 내역]



- 구분 비트: 가장 왼쪽에 있는 비트

ex) 클래스 A - 0, 클래스 B - 10, 클래스 C - 110

클래스 A에서 가질 수 있는 IP 범위: 00000000.00000000.00000000.00000000  
~01111111.11111111.11111111.11111111



- 네트워크의 첫 번째 주소: 네트워크 구별 주소

- 네트워크의 마지막 주소: 브로드캐스트용 주소; 네트워크에 속해 있는 모든 컴퓨터에 데이터 전송

〈예시〉

클래스 A로 12.0.0.0이라는 네트워크를 부여받음

12.0.0은 네트워크 구별주소, 12.255.255.255는 브로드캐스트 주소이므로

12.0.0.1~12.255.255.254의 호스트 주소를 부여받음

[단점] 사용하는 주소보다 버리는 주소가 많음 → DHCP, IPv6, NAT 등장

**DHCP**  
Dynamic Host Configuration Protocol

IP 주소 및 기타 통신 매개변수를 자동 할당하기 위한 네트워크 관리 프로토콜  
네트워크 장치의 IP 주소를 수동으로 설정할 필요 X  
→ 인터넷에 접속할 때마다 자동으로 IP 주소 할당

[탑재된 장비] 라우터 및 게이트웨이

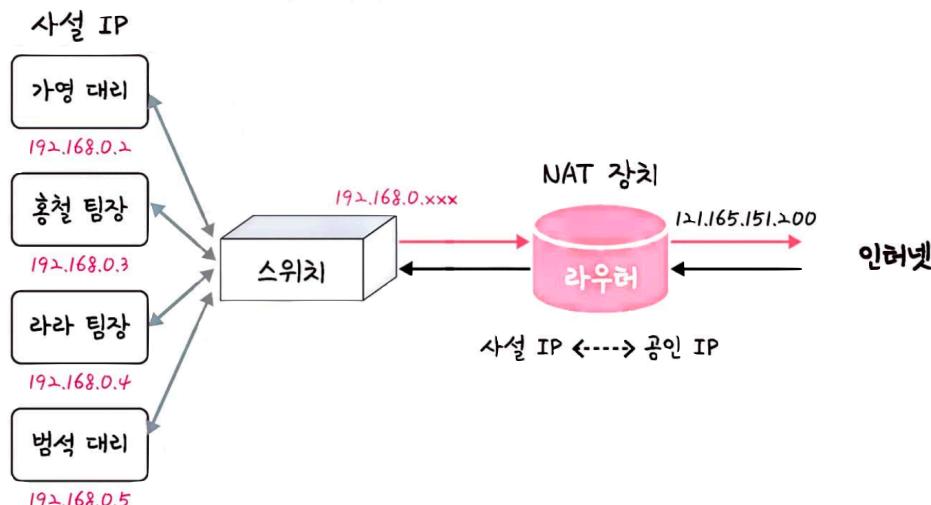
대부분의 가정용 네트워크에서 IP 주소 할당

**NAT**  
Network Address Translation

패킷이 라우팅 장치를 통해 전송되는 동안 패킷의 IP 주소 정보를 수정해 다른 주소로 매플  
IPv4 주소 체계만으로 많은 주소들을 모두 감당하지 못하는 단점 존재  
→ NAT로 공인 IP와 사설 IP를 나눠서 주소 처리: 사설 IP ←→ 공인 IP

[NAT 가능한 SW] ICS, RRAS, Netfilter

### 서비스 회사



### [공유기와 NAT]

- NAT 사용하는 이유: 여러 대의 호스트가 하나의 공인 IP 주소를 사용해 인터넷에 접속하기 위함  
ex) 인터넷 회선 하나를 개통 후 인터넷 공유기를 달아 여러 PC를 연결하여 사용  
→ 인터넷 공유기에 NAT가 탑재되어 있어 가능한 일

### [NAT를 이용한 보안]

NAT 이용 시 내부 네트워크에서 사용하는 IP 주소와 외부에 드러나는 IP 주소를 다르게 유지 가능  
→ 네트워크에 대한 보안 가능

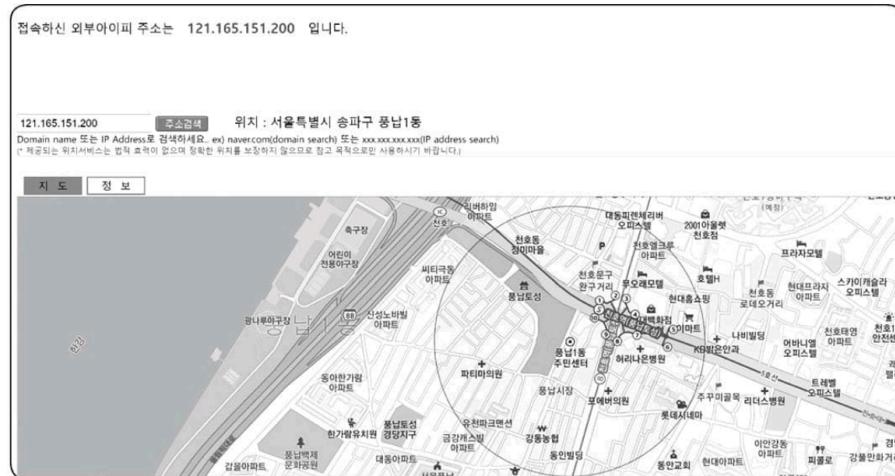
### [NAT의 단점]

여러 명이 동시에 인터넷 접속 → 접속하는 호스트 수에 따라 속도 저하

#### 2.4.4 IP 주소를 이용한 위치 정보

IP 주소 = 인터넷에서 사용하는 네트워크 주소 → 동 또는 구까지 위치 추적 가능

IP 주소를 기반으로 위치를 찾는 사이트: <https://mylocation.co.kr/>



## 2.5 HTTP

### Section 2.5 HTTP

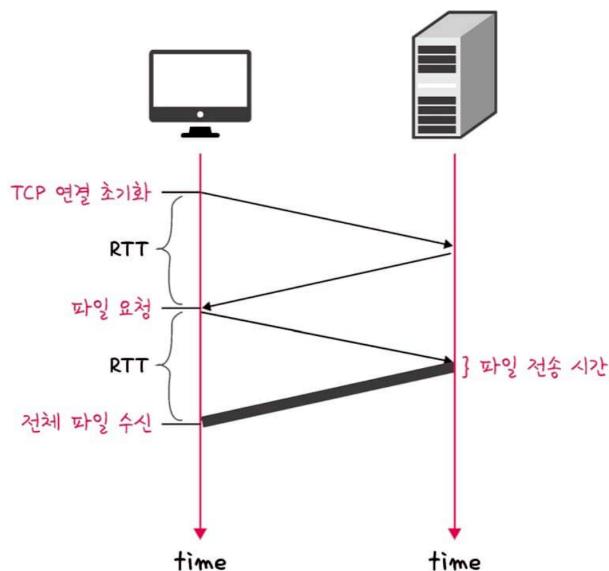
#### 2.5.1 HTTP/1.0

##### HTTP/1.0

한 연결당 하나의 요청 처리 → RTT 증가 초래

##### RTT 증가

서버로부터 파일을 가져올 때마다 TCP의 3-웨이 핸드쉐이크를 계속해서 열어야 함 → RTT 증가  
→ 서버에 부담 증가 → 응답시간 증가



##### RTT의 증가를 해결하기 위한 방법

###### - 이미지 스플리팅

많은 이미지 다운로드 시 과부하를 방지하기 위해 이미지가 합쳐져 있는 하나의 이미지를 다운받고 그것을 기반으로 background-image의 position을 이용해 이미지 표기

###### - 코드 압축

코드를 압축해서 개행 문자, 빈칸을 없애 코드 용량 최소화

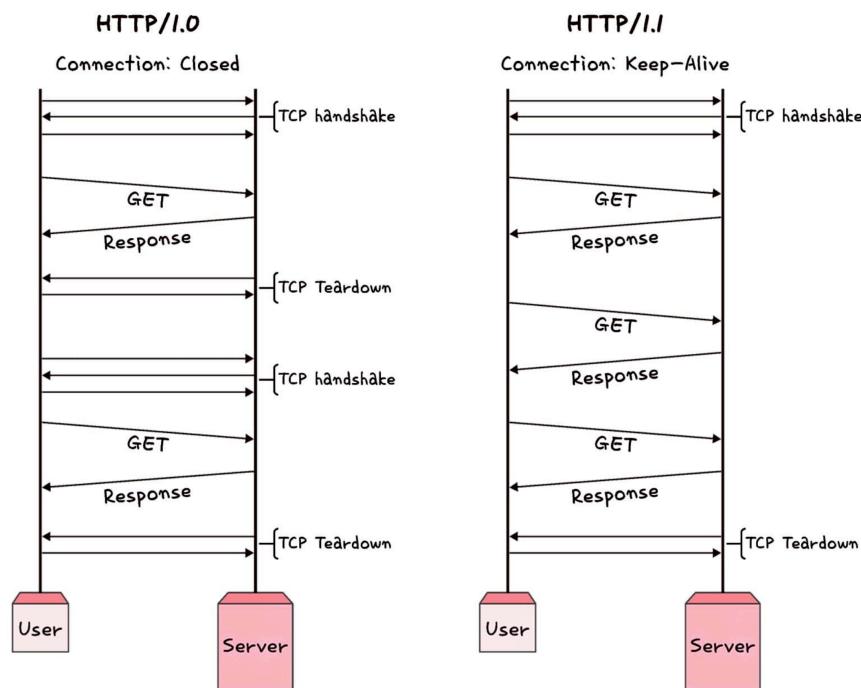
###### - 이미지 Base64 인코딩

이미지 파일을 64진법의 문자열로 인코딩  
[장점] 서버와의 연결을 열고 이미지에 대해 서버에 HTTP 요청 필요 X  
[단점] 약 37% 용량 증가

## 2.5.2 HTTP/1.1

HTTP/1.0에서 발전  
매번 TCP 연결 X → 한 번 TCP 초기화 후 keep-alive라는 옵션으로 여러 개의 파일을 송수신  
cf. HTTP/1.0에도 keep-alive 존재 but 표준화 X, HTTP/1.1에서 표준화되어 기본 옵션으로 설정

[HTTP/1.0과 HTTP/1.1의 비교]

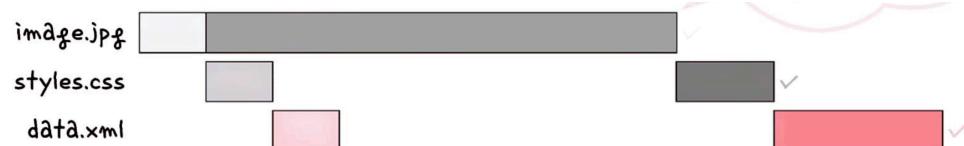


HTTP/1.1 : 한 번 TCP 3-웨이 핸드셰이크 발생 시 그다음부터는 발생 X

[단점] 문서 안에 포함된 다수의 리소스(이미지, css 파일, script 파일) 처리 시  
요청할 리소스 개수에 비해 대기 시간↑

HOL Blocking  
Head Of Line Blocking

네트워크에서 같은 큐에 있는 패킷이 그 첫 번째 패킷에 의해 지연될 때 발생하는 성능 저하 현상



무거운 헤더 구조

쿠키 등의 많은 메타데이터가 들어 있고 압축이 되지 않아 무거움

### 2.5.3 HTTP/2

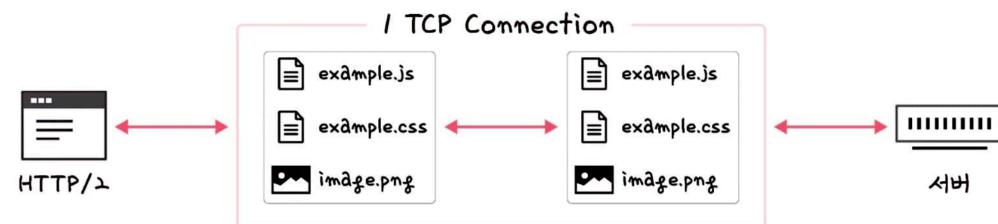
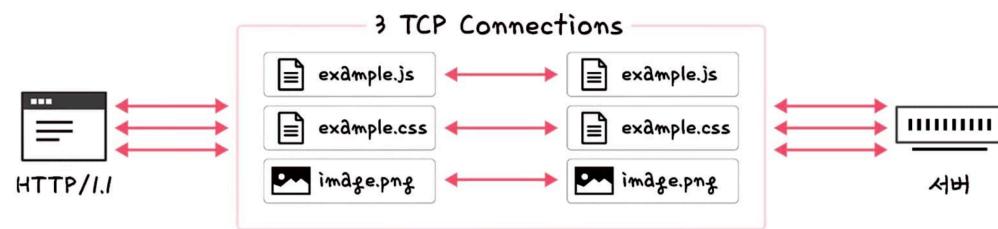
SPDY 프로토콜에서 파생된 HTTP/1.x보다  
지연시간 ↓, 응답시간 ↑  
멀티플렉싱, 헤더 압축, 서버 푸시, 요청의 우선순위 처리 지원

#### 멀티플렉싱

여러 개의 스트림을 사용하여 송수신 → 특정 스트림의 패킷 손실 시  
해당 스트림에만 영향, 나머지 스트림은 정상 작동



스트림 내의 데이터들도 쪼개어져 있음: 애플리케이션에서 받아온 메세지 → 독립된 프레임  
서로 송수신한 이후 다시 조립함으로써 데이터 교환



단일 연결을 사용해 병렬로 여러 요청 → 응답 ↓ → HOL Blocking 해결

## 헤더 압축

HTTP/1.1x의 문제점 중 하나: 크기가 큰 헤더 → HTTP/2에서는 헤더 압축을 통해 해결

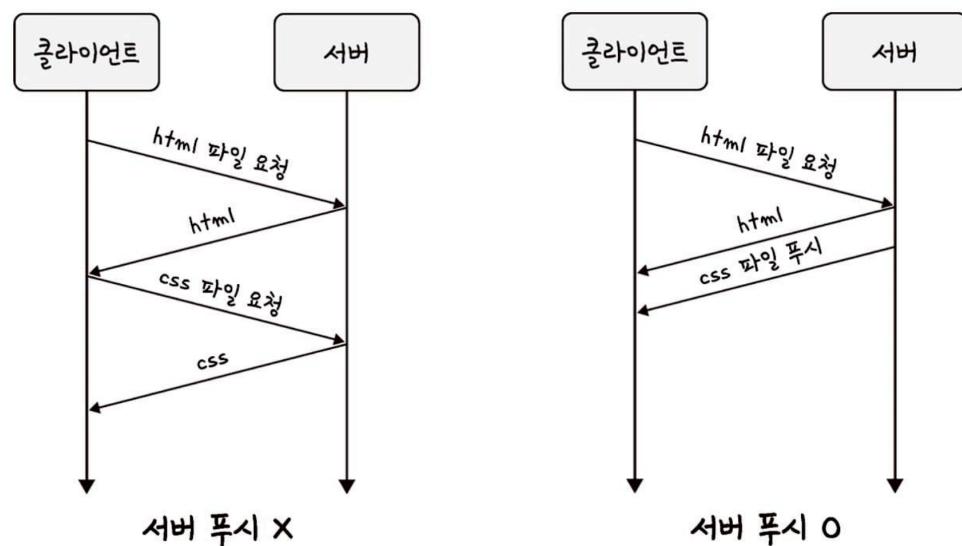
히프만 코딩 압축 알고리즘을 사용하는 HPACK 압축 형식

- 히프만 코딩  
huffman coding

문자열을 문자 단위로 조개 빈도수를 셈 → 빈도 높은 정보: 적은 비트 수 사용  
빈도 낮은 정보: 많은 비트 수 사용  
→ 전체 데이터 표현에 필요한 비트양 감소

## 서버 푸시

HTTP/1.1: 클라이언트가 서버에 요청을 해야만 파일 다운로드 가능  
HTTP/2: 클라이언트 요청 없이 서버가 바로 리소스 푸시 가능



html에는 css나 js 파일이 포함되어 있어,  
html을 읽으면서 그 안에 들어있던 css 파일을 서버에서 푸시하여 클라이언트에 먼저 제공 가능

## 2.5.4 HTTPS

**HTTPS**

HTTP/2는 HTTPS 위에서 동작

애플리케이션 계층과 전송계층 사이에 신뢰계층인 SSL/TLS계층을 넣은 신뢰할 수 있는 HTTP 요청  
→ 통신을 암호화

**SSL/TLS**

Secure Socket Layer/  
Transport Layer  
Security Protocol

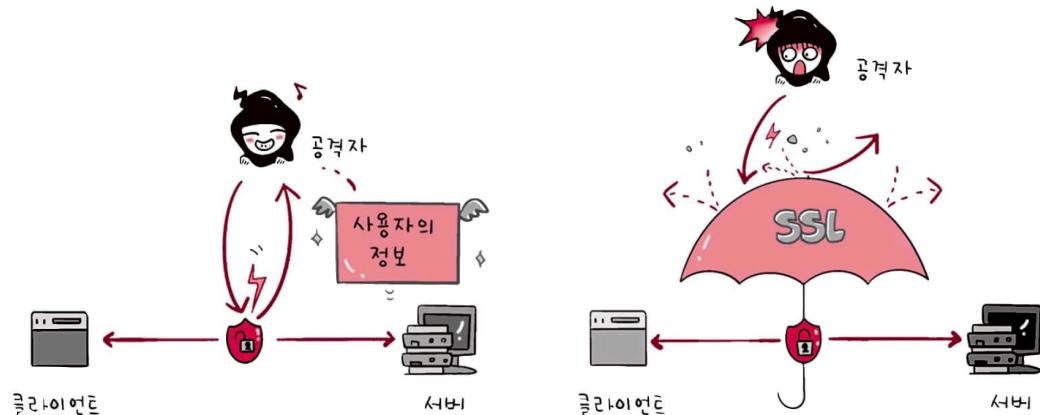
전송 계층에서 보안을 제공하는 프로토콜

클라이언트-서버 통신 시 SSL/TLS를 통해 제 3자의 메세지 도청 및 변조 방지

변천 과정: SSL(1.0 → 2.0 → 3.0) → TLS(1.0 → 1.3)

cf. TLS로 명칭이 변경되었으나 통칭 SSL/TLS로 사용

[SSL/TLS를 이용한 인터셉터 방지]



SSL/TLS는 보안 세션 기반으로 데이터 암호화

→ 보안 세션이 생성 시 사용되는 알고리즘: 인증 메커니즘, 키 교환 암호화 메커니즘, 해싱 알고리즘

- 보안 세션

보안이 시작되고 끝나는 동안 유지되는 세션

SSL/TLS는 핸드셰이크를 통해 보안 세션 생성 후 이를 기반으로 상태 정보 등을 공유

[TLS의 핸드셰이크]

클라이언트

서버

ClientHello

{+KeyShare}



ServerHello

{+KeyShare}

EncryptedExtensions

Certificate

CertificateVerify

Finished

[Application Data]



Finished

[Application Data]



Application Data

Application Data

클라이언트와 서버가 키를 공유

→ 인증, 인증 확인 등의 작업이 일어나는 단 한번의 1-RTT가 생긴 후 데이터 송수신

1. 클라이언트에서 사이퍼 슈트 cypher suites를 서버에 전달
2. 서버가 받은 사이퍼 슈트의 암호화 알고리즘 리스트를 제공할 수 있는지 확인
3. 제공 가능 → 클라이언트로 인증서를 보내는 인증 메커니즘 시작
4. 해싱 알고리즘 등으로 암호화된 데이터 송수신 시작

〈사이퍼 슈트〉 프로토콜, AEAD 사이퍼 모드, 해싱 알고리즘이 나열된 규약

- TLS\_AES\_128\_GCM\_SHA256
- TLS\_AES\_256\_GCM\_SHA384
- TLS\_CHACHA20\_POLY1305\_SHA256
- TLS\_AES\_128\_CCM\_SHA256
- TLS\_AES\_128\_CCM\_8\_SHA256

Ex. TLS\_AES\_128\_GCM\_SHA256에서

TLS는 프로토콜, AES\_128\_GCM은 AEAD 사이퍼 모드, SHA356은 해싱 알고리즘

- 인증 메커니즘

〈AEAD Authenticated Encryption with Associated Data 사이퍼 모드〉

AEAD: 데이터 암호화 알고리즘

Ex. AES\_128\_GCM: 128비트의 키를 사용하는 표준 블록 암호화 기술과 병렬 계산에 용이한 암호화 알고리즘 GCM이 결합된 알고리즘

- 암호화 알고리즘

CA Certificate Authorities에서 발급한 인증서 기반

CA에서 발급한 인증서: 안전한 연결을 시작할 때 필요한 '공개키'를 클라이언트에 제공  
사용자가 접속한 '서버가 신뢰'할 수 있는 서버임을 보장  
[구성] 서비스 정보, 공개키, 지문, 디지털 서명 등

Cf. 엄격하게 공인된 기업들만 참여 가능 Ex. Comodo, GoDaddy, GlobalSign

〈CA 발급 과정〉

자신의 사이트 정보와 공개키를 CA에 제출

→ 공개키를 해시한값인 지문 finger print을 사용하는 CA의 비밀키 등을 기반으로 CA 인증서 발급

= 키 교환 암호화 알고리즘

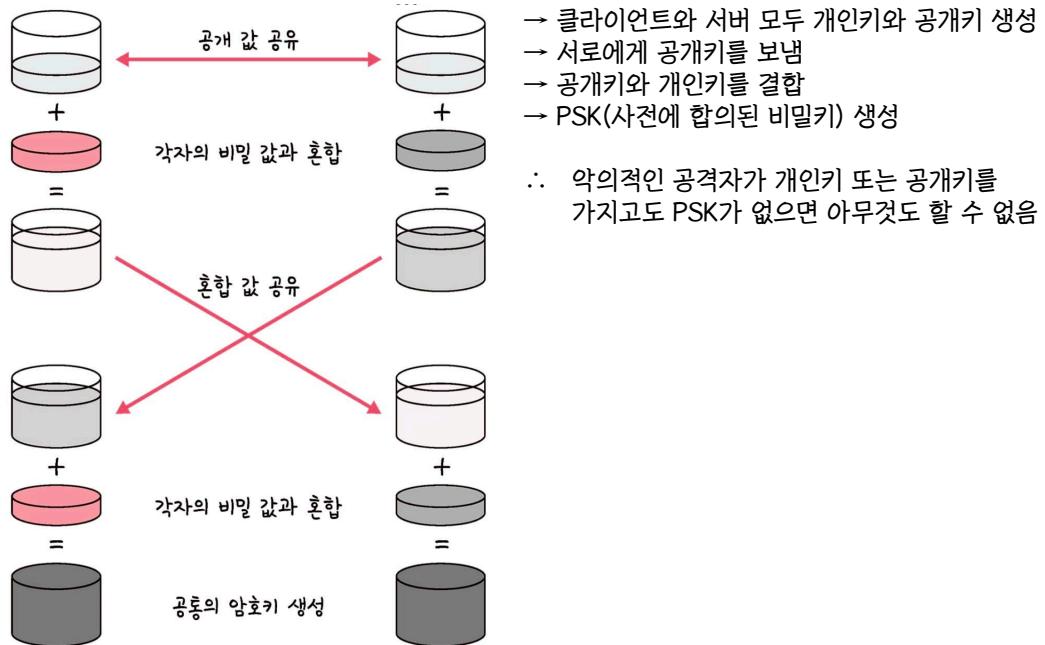
- 대수곡선 기반: ECDHE(Elliptic Curve Diffie-Hellman Ephemeral)

- 모듈식 시반: DHE(Diffie-Hellman Ephemeral)

〈디피-헬만 키 교환 Diffie-Hellman key exchange 암호화 알고리즘〉

$$y = g^x \bmod p$$

g와 x와 p를 알면 y는 구하기 쉽지만 g와 y와 p만 안다면 x를 구하기 어렵다는 원리에 기반



- 해싱 알고리즘

데이터를 추정하기 힘든 더 작고, 섞여 있는 조각으로 만드는 알고리즘  
SSL/TLS의 해싱 알고리즘: SHA-256, SHA-384

〈SHA-256 알고리즘〉

해시 함수의 결과값이 256비트인 알고리즘

비트코인 등의 많은 블록체인 시스템에서 사용

해싱을 해야 할 메세지에 1을 추가하는 등 전처리 → 전처리된 메세지 기반으로 해시 반환

## SHA256

SHA256 online hash function

나는 반드시 이번 면접에 합격하고 강원도 고성에 수영하러 간다!

Input type  Text ▾

Auto Update

08cc3029b838d4be3ed53ff e3bab5be2c2d44526218d365bfd15673e27838f

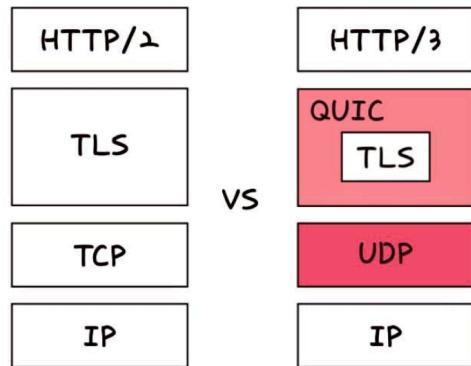
SHA-256 등 다양한 해싱 알고리즘을 테스팅할 수 있는 사이트 링크  
: <https://emn178.github.io/online-tools/sha256.html>

Cf. TLS 1.3은 사용자가 이전에 방문한 사이트로 다시 방문 시  
SSL/TLS에서 보안 세션을 만들 때 걸리는 통신을 하지 않아도 됨 → 0-RTT

<p>SEO에도 도움이 되는 HTTPS</p> <ul style="list-style-type: none"> <li>- 캐노니컬 설정</li> <li>- 메타 설정</li> <li>- 페이지 속도 개선</li> <li>- 사이트맵 관리</li> </ul> <p>HTTPS 구축 방법</p>	<p>구글은 SSL 인증서를 강조해왔고 사이트 내 모든 요소가 동일하다면 HTTPS 서비스를 하는 사이트가 그렇지 않은 사이트보다 SEO 순위가 높을 것이라고 밝힘</p> <p><b>SEO</b>Search Engine Optimization : 웹사이트 검색 시 그 결과를 페이지 상단에 노출시켜 많은 사람이 볼 수 있도록 최적화하는 방법</p> <p>서비스 운영 시 필수적인 SEO 관리: 캐노니컬 설정, 메타 설정 페이지 속도 개선, 사이트맵 관리 등</p> <p>사이트 link에 설정하는 것</p> <pre>&lt;link rel="canonical" href="https://example.com/page2.php" /&gt;</pre> <p>메타: html 파일의 가장 윗부분</p> <p>[애플의 메타]</p> <pre>&lt;meta property="analytics-track" content="Apple - Index/Tab"&gt; &lt;meta property="analytics-s-channel" content="homepage"&gt; &lt;meta property="analytics-s-bucket-0" content="applestoreww"&gt; &lt;meta property="analytics-s-bucket-1" content="applestoreww"&gt; &lt;meta property="analytics-s-bucket-2" content="applestoreww"&gt; &lt;meta name="Description" content="Discover the innovative world of Apple and shop everything iPhone, iPad, Apple Watch, Mac, and Apple TV, plus explore accessories, entertainment, and expert device support."&gt; &lt;meta property="og:title" content="Apple"&gt;</pre> <p>구글의 PageSpeedInsights에서 자신의 서비스에 대한 리포팅을 주기적으로 받으며 관리 필요 페이지 인사이트 링크: <a href="https://developers.google.com/speed/pagespeed/insights/">https://developers.google.com/speed/pagespeed/insights/</a></p> <p>사이트맵 sitemap.xml 을 관리하는 방법: 사이트맵 제너레이터 사용 / 직접 코드 구축 Ex.</p> <pre>&lt;?xml version="1.0" encoding="utf-8"?&gt; &lt;urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9"&gt; &lt;url&gt; &lt;loc&gt;http://kundol.co.kr/&lt;/loc&gt; &lt;lastmod&gt;수정날짜&lt;/lastmod&gt; &lt;changefreq&gt;daily&lt;/changefreq&gt; &lt;priority&gt;1.1&lt;/priority&gt; &lt;/url&gt; &lt;/urlset&gt;</pre> <ul style="list-style-type: none"> <li>직접 CA에서 구매한 인증키 기반으로 구축</li> <li>서버 앞단의 HTTPS를 제공하는 로드밸런서를 두어 구축</li> <li>서버 앞단에 HTTPS를 제공하는 CDN을 두어 구축</li> </ul>
---	--

## 2.5.5 HTTP/3

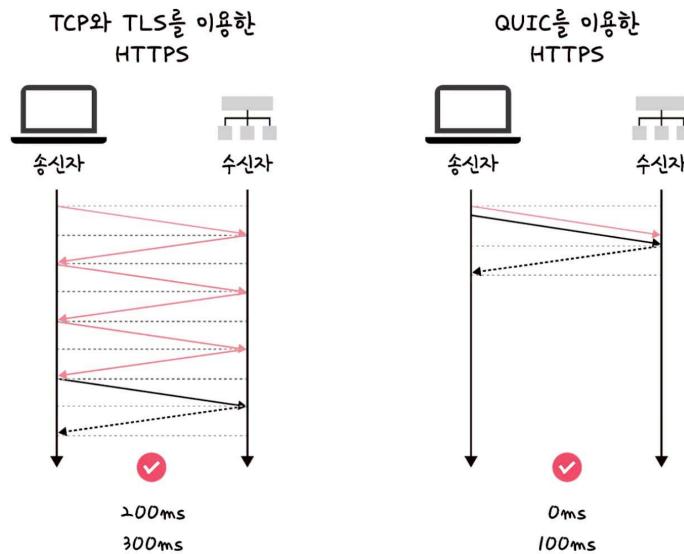
HTTP의 세 번째 버전  
TCP 위에서 돌아가는 HTTP/2와 달리 QUIC이라는 계층 위에서 돌아감  
TCP 기반이 아닌 UDP 기반



[장점] 멀티플렉싱, 초기 연결 설정 시 지연 시간 감소

QUIC은 TCP를 사용하지 않기 때문에 통신 시작 시 번거로운 3-웨이 핸드쉐이크 과정 X

초기 연결 설정 시  
지연 시간 감소



QUIC은 첫 연결 설정에 1-RTT만 소요: 클라이언트가 서버에 어떤 신호를 한 번 주고  
서버가 거기에 응답만 하면 바로 본 통신 시작

Cf. QUIC은 순방향 오류 수정 메커니즘 FEC Forward Error Correction이 적용됨

→ 전송한 패킷 손실 시 수신 측에서 여러 검출 및 수정

→ 열악한 네트워크 환경에서도 패킷 손실률 ↓

## Chapter 3 운영체제

---

# 3.1 운영체제와 컴퓨터

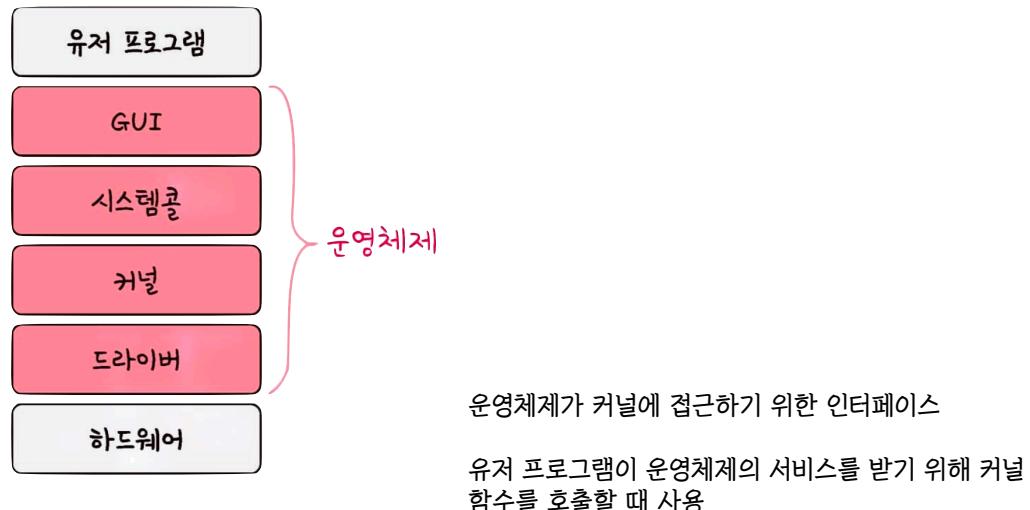
## 3.1 운영체제와 컴퓨터

### 3.1.1 운영체제의 역할과 구조

#### 운영체제의 역할

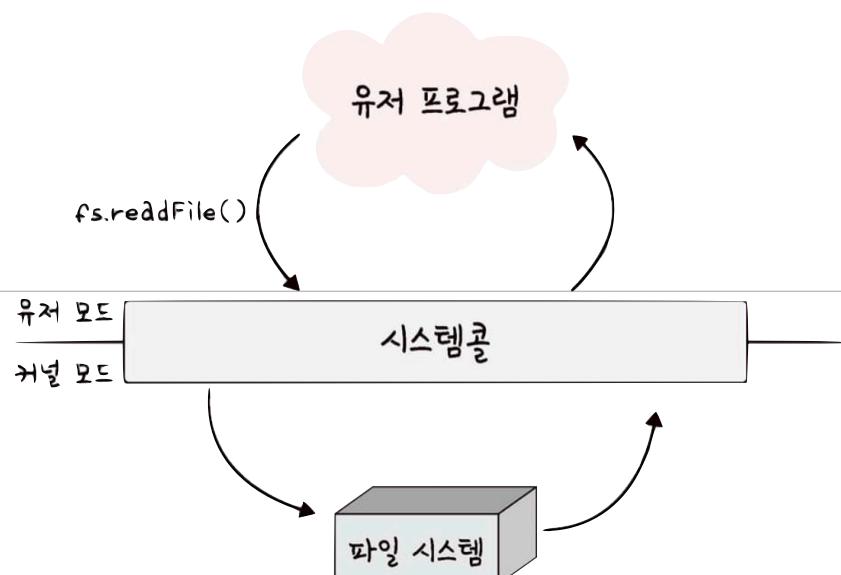
- CPU 스케줄링과 프로세스 관리  
CPU 소유권을 어떤 프로세스에 할당할지, 프로세스의 생성 및 삭제, 자원 할당 및 반환 관리
  - 메모리 관리: 한정된 메모리를 어떤 프로세스에 얼마나 할당해야 하는지 관리
  - 디스크 파일 관리: 디스크 파일 보관 방법 관리
  - I/O 디바이스 관리: 마우스, 키보드와 컴퓨터 간 데이터 교환 관리
- \* GUI가 없고 CUI만 있는 리눅스 서버도 존재

#### 운영체제의 구조

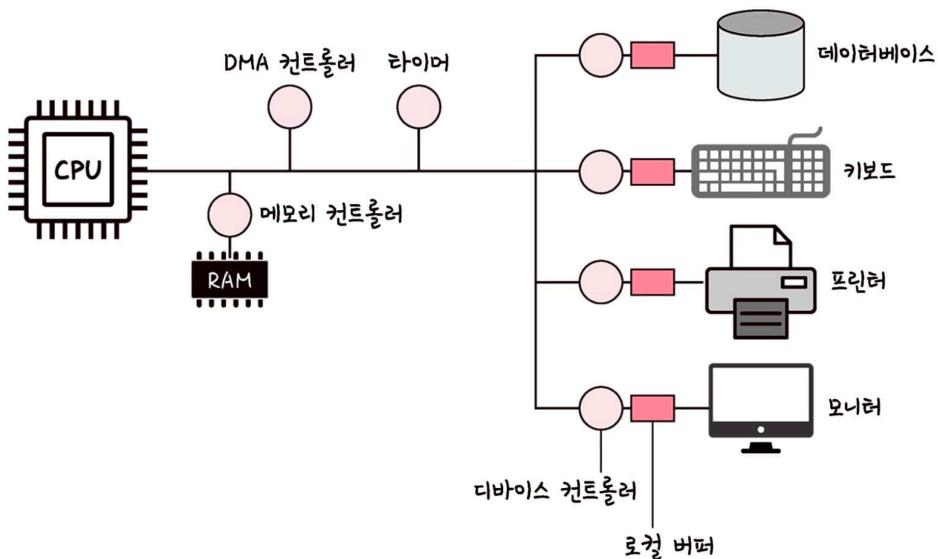


#### 시스템콜

- 유저 프로그램이 I/O 요청으로 트랩<sup>trap</sup> 발생 시  
올바른 I/O 요청인지 확인 후  
시스템콜을 통해 유저 모드가 커널 모드로 변환되어 실행  
→ 컴퓨터 자원에 대한 직접 접근 차단  
프로그램을 다른 프로그램으로부터 보호



### 3.1.2 컴퓨터의 요소



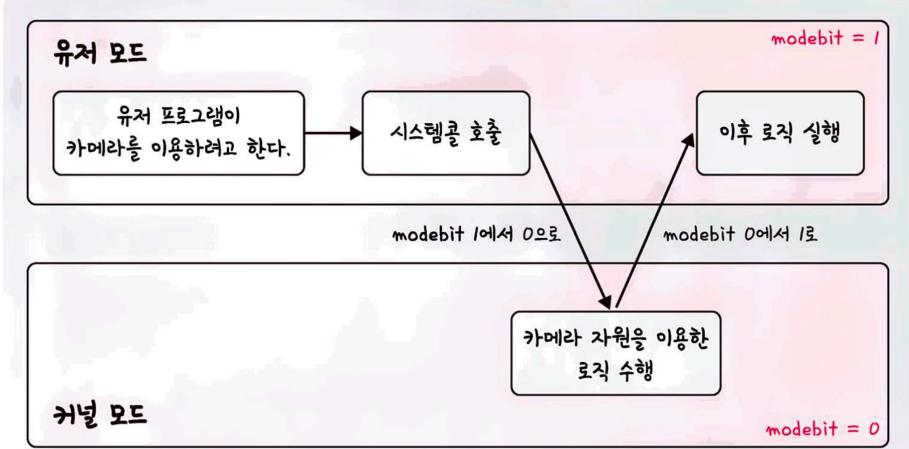
[시스템콜과 커널, 운영체제]

CPU  
Central Processing Unit

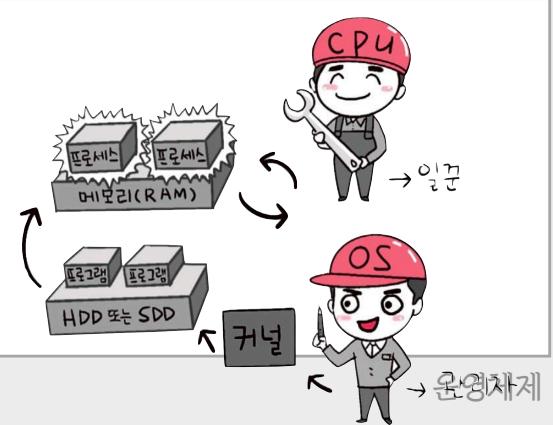
modebit  
- 레지스터  
Register

- 제어장치  
CU: Control Unit

- 산술논리연산장치  
ALU  
; Arithmetic Logic Unit



[장점] 낮은 단계의 영역 처리에 대한 부분을 신경쓰지 않고 프로그램 구현할 수 있음



유저 모드 기반으로 디바이스가 커지게 된다면 공격자가 디바이스 작동에 접근하기 쉬워지므로  
I/O 디바이스는 운영체제를 통해서만 작동  
(커널 모드를 거쳐 운영체제를 통해 작동하게 해야 막기 쉬움)

↓ 이를 위한 장치

시스템 콜 작동 시 유저 모드와 커널 모드를 구분할 때 참고하는 플래그 변수

인터럽트에 의해 단순히 메모리에 존재하는  
명령어를 해석해 실행하는 장치

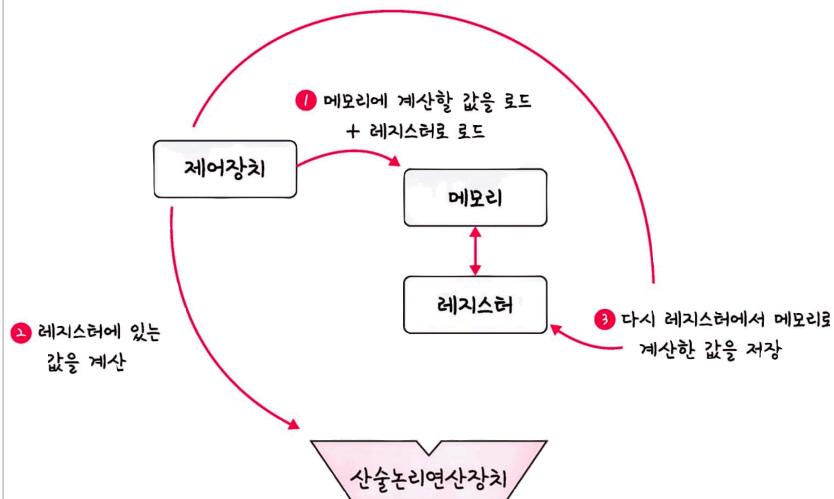
운영체제의 커널이 프로그램을 메모리에 옮겨  
프로세스로 만들면 그것을 CPU가 처리

프로세스 조작 지시  
입출력 간 통신 제어  
명령어 읽고 해석  
데이터 처리 순서 결정

매우 빠른 임시기억장치  
연산 속도가 메모리보다 수십~수백배 빠름 ; CPU와 직접 연결되어 있기 때문  
CPU는 자체적으로 데이터를 저장할 수 없기 때문에 레지스터를 거쳐 데이터 전달

산술 및 논리 연산을 계산하는 디지털 회로

## [CPU의 연산처리]



1. 제어장치 → 메모리와 레지스터에 계

- 산할 값 로드
- 제어장치 → 산술논리연산장치에게 레지스터에 있는 값을 계산하라고 명령
- 제어장치 → 레지스터에 저장된 계산된 값을 메모리에 저장

## - 인터럽트

어떤 신호가 들어왔을 때 CPU를 잠시 정지시킴

인터럽트 간에 우선순위 존재

## [하드웨어 인터럽트]

I/O 디바이스에서 발생하는 인터럽트: 키보드 및 마우스 연결

인터럽트 라인 설계 이후 순차적인 인터럽트 실행 중지

→ 운영체제에 시스템콜 요청

→ 원하는 디바이스에 있는 작은 로컬 버퍼에 접근해 작업 수행

## [소프트웨어 인터럽트] = 트랩 trap

프로세스 오류 등으로 프로세스가 시스템콜 호출 시 발생

[인터럽트 발생 시?] 인터럽트 벡터의 인터럽트 핸들러 함수 실행

인터럽트 핸들러 함수: 인터럽트 발생 시 핸들링하기 위한 함수

커널 내부의 IRQ를 통해 호출

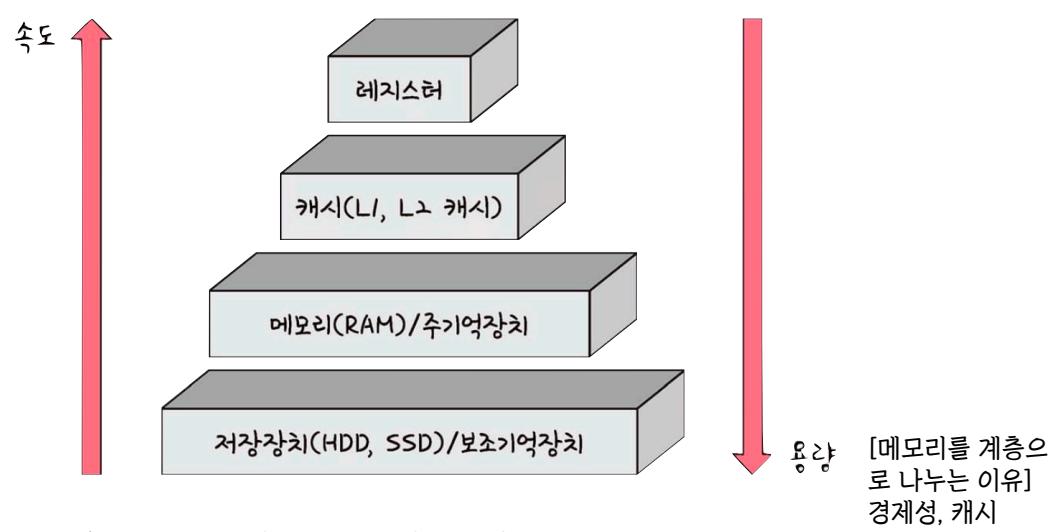
request\_irq()를 통해 인터럽트 핸들러 함수 등록

DMA 컨트롤러 <small>device controller</small>	I/O 디바이스가 메모리에 직접 접근할 수 있도록 하는 하드웨어 장치 [존재 의의] CPU에만 너무 많은 인터럽트 요청이 들어옴 → 작업 분담을 통해 CPU 부하 방지
메모리 <small>memory</small>	= RAM Random Access Memory 전자회로에서의 데이터, 상태, 명령어 <u>기록 장치</u>
타이머	특정 프로그램에 작업 시간 제한
디바이스 컨트롤러 <small>device controller</small>	I/O 디바이스들의 작은 CPU

## 3.2 메모리

### 3.2 메모리

#### 3.2.1 메모리 계층



Ex) 16GB RAM보다 16GB SSD가 훨씬 저렴함

[일상생활에서의 메모리 계층 경험] 로딩 중  
→ HDD 또는 인터넷에서 데이터를 읽어 RAM으로 전송하는 과정

데이터를 미리 복사해 놓는 임시 저장소  
병목 현상 완화시키는 메모리

→ 시간 절약: 데이터 접근 시간이 긴 경우를 해결하고 무언가를 다시 계산하는 시간 절약  
Ex) 메모리와 CPU

프로세스 내의 명령어 및 데이터에 대한 참조가 군집화 경향을 띨

- 지역성의 원리

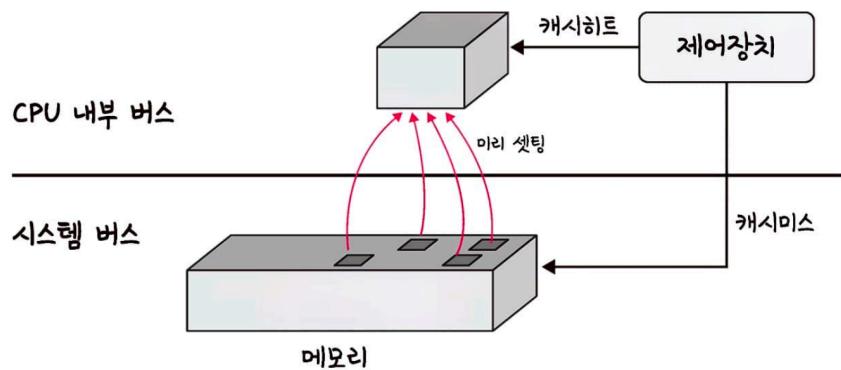
[시간 지역성 temporal locality]

최근 사용한 데이터에 다시 접근하는 특성

[공간 지역성 spatial locality]

최근 접근한 데이터를 이루고 있는 공간 및 가까운 공간에 접근하는 특성

## 캐시히트와 캐시미스



[캐시히트] 캐시에 찾는 데이터 O

→ 제어장치를 거쳐 가져옴: 빠름(CPU 내부 버스 기반)

[캐시미스] 캐시에 찾는 데이터 X → 주 메모리에서 데이터를 찾아옴: 느림(시스템버스 기반)

- 캐시매핑  
cache mapping

[직접 매핑directed mapping]

- 다대일 방식: 메모리의 여러 주소가 캐시 메모리의 한 주소에 대응  
Ex) 메모리: 1~100, 캐시: 1~10 → 1: 1~10, 2: 11~20, ...
- 처리가 빠르지만 충돌 발생↑

[연관 매핑associated mapping]

- 비어있는 캐시 메모리가 있으면 랜덤하게 주소 저장
- 저장 시에는 간단하지만 찾을 때는 모든 블록을 탐색해야 하므로 효율↓

[집합 연관 매핑set associated mapping]

- 직접 매핑 장점 + 연관 매핑 장점: 특정 행을 지정해 그 행 안의 어떤 열이든 비어있으면 저장  
Ex) 메모리: 1~100, 캐시: 1~10 → 1~5: 1~50 무작위로 저장
- 검색과 저장 속도: 직접 매핑과 연관 매핑의 중간 속도

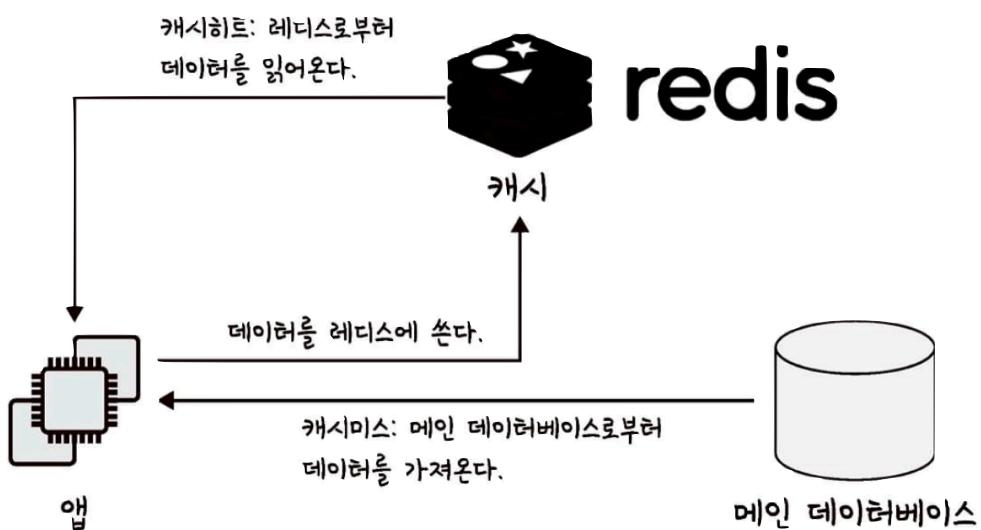
- 웹 브라우저의 캐시

사용자의 커스텀한 정보 및 인증 모듈 관련 사항 → 웹 브라우저에 저장: 추후 중복 요청 방지

	타입	만료기한	저장 용량	
쿠키	키-값	○	4KB	<ul style="list-style-type: none"> <li>- 다른 도메인에서 요청했을 때 자동 전송: same site 옵션을 strict로 설정하지 않은 경우</li> <li>- 설정 시 document.cookie로 httponly 옵션을 걸어 쿠키를 볼 수 없게 하는 것이 중요</li> <li>- 서버 또는 클라이언트에서 만료기한 지정</li> </ul>
로컬 스토리지	키-값	×	10MB	<ul style="list-style-type: none"> <li>- 웹 브라우저를 닫아도 유지</li> <li>- 도메인 단위로 저장·생성</li> <li>- HTML5 지원 웹 브라우저에서만 사용 가능</li> <li>- 클라이언트에서만 수정 가능</li> </ul>
세션 스토리지	키-값	×	5MB	<ul style="list-style-type: none"> <li>- 탭 단위로 저장·생성</li> <li>- HTML5 지원 웹 브라우저에서만 사용 가능</li> <li>- 클라이언트에서만 수정 가능</li> </ul>

- 데이터베이스의 캐싱 계층

데이터베이스 시스템 구축 시 메인 데이터베이스 위에  
레디스(redis) 데이터베이스 계층을 캐싱 계층으로 두고 성능 향상

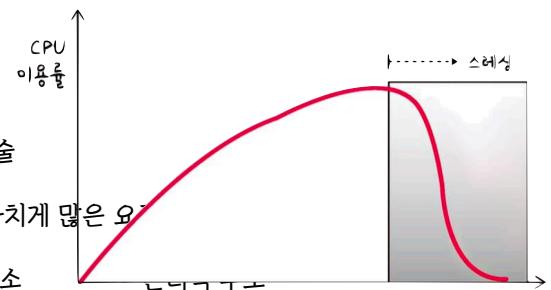


### 3.2.2 메모리 관리

#### 스레싱 thrashing

운영체제의 대표적인 업무

한정된 메모리를 극한으로 활용



#### 가상 메모리 virtual memory

메모리가 실제 메모리보다 많아 보이게 하는 기술

프로세스들이 존재하는 메모리가 여유 없이 지나치게 많은 요

[가상 주소logical address] 가상으로 주어진 주소

[실제 주소physical address] 실제 메모리상에 있는 주소 = 물리적 주소

메모리관리장치MMU: 가상 주소 → 실제 주소

가상 주소와 실제 주소가 매핑되어 있고, 페이지 테이블을 통해 관리

→ 속도 향상을 위해 TLB 사용

#### 스와핑 swapping

당장 사용하지 않는 영역을 하드디스크로 옮기고  
하드디스크의 일부분을 마치 메모리처럼 불러와 쓰는 것

[페이지 폴트page fault와 그로 인한 스와핑 과정]

1. CPU가 물리 메모리를 확인해 페이지 탐색 → 트랩 발생시켜 OS에 알림
2. OS가 CPU 동작 정지
3. OS가 페이지 테이블 확인해 가상 메모리에 페이지 존재 여부 확인  
→ 현재 물리 메모리에 빈 프레임 존재하는지 확인 → 스와핑
4. 비어있는 프레임에 해당 페이지 로드 → 페이지 테이블 최신화
5. 중단 종이었던 CPU 다시 실행

페이지 폴트율이 높은 상태  
컴퓨터의 심각한 성능 저하 초래

[발생 과정]

많은 프로세스가 동시에 메모리에 올라가 스와핑 ↑

→ 페이지 폴트 발생 → CPU 이용률 ↓

→ OS: CPU 가용성 ↑ 위해 더 많은 프로세스 적재

	<p>→ 반복 → 스레싱</p> <p>[해결 방법] 작업 세트, PFF</p> <p>지역성 <i>locality</i>를 통해 결정된 페이지 집합을 만들어 미리 메모리에 로드</p> <p>[장점] 탐색 비용·스와핑 ↓</p> <p>페이지 폴트 빈도 조절법 – 상한선, 하한선에 도달함에 따라 프레임을 늘리고 줄임</p>
- 작업 세트 working set  - PFF Page Fault Frequency	

<p><b>메모리 할당</b></p> <ul style="list-style-type: none"> <li>- 연속 할당</li> </ul> <p>[할당 기준] 시작 메모리 위치, 메모리의 할당 크기</p> <p>메모리에 연속적으로 공간 할당</p> <p>[고정 분할 방식fixed partition allocation] 메모리를 미리 나누어 관리 용통성X, 내부 단편화internal fragmentation 발생</p> <p>[가변 분할 방식variable partition allocation] 매 시점 크기에 맞게 동적으로 메모리를 나눔 외부 단편화external fragmentation 발생</p> <ul style="list-style-type: none"> <li>- 최초적합: 위쪽이나 아래쪽부터 시작해서 훌을 찾는 할당</li> <li>- 최적적합: 프로세스의 크기 이상인 공간 중 가장 작은 훌부터 할당</li> <li>- 최악적합: 프로세스의 크기 이상인 공간 중 가장 큰 훌부터 할당</li> </ul> <p>메모리를 연속적으로 할당하지 않음</p> <ul style="list-style-type: none"> <li>- 불연속 할당</li> </ul> <p>[페이지 paging 기법] 메모리를 동일한 크기의 페이지(보통 4KB)로 나누고 프로그램마다 페이지 테이블을 두어 이를 통해 메모리에 프로그램을 할당</p> <p>장점: 훌의 크기가 균일 단점: 주소 변환 복잡</p> <p>[세그멘테이션segmentation] 의미 단위인 세그먼트segment로 나누는 방식 : 프로세스의 코드와 데이터 등을 기반으로 또는 함수 단위로 나눔</p> <p>장점: 공유 및 보안 측면에서 좋음 단점: 훌 크기 불균일</p> <p>[페이지드 세그멘테이션paged segmentation] 공유나 보안 → 세그먼트 단위 물리적 메모리 → 페이지 단위</p>	
--	--

	<p>페이지 교체 알고리즘</p> <ul style="list-style-type: none"> <li>- 오프라인 알고리즘 offline algorithm</li> <li>- FIFO First In First Out</li> <li>- LRU Least Recently Used</li> <li>- NUR Not Used Recently</li> <li>- LFU Least Frequently Used</li> </ul> <p>스와핑이 많이 발생하지 않도록 설계하는 알고리즘</p> <p>먼 미래에 참조되는 페이지와 현재 할당하는 페이지를 바꾸는 알고리즘</p> <p>가장 좋은 방법이지만, 미래에 사용되는 프로세스를 알 수 없기 때문에 사용 불가</p> <p>[존재 의의] 다른 알고리즘과의 성능 비교에 대한 기준 제공</p> <p>가장 먼저 할당된 페이지를 가장 먼저 교체</p> <p>참조된지 가장 오래된 페이지를 먼저 교체 '오래됨'을 파악하기 위해 각 페이지마다 계수기, 스택을 둠</p> <p>[구현 자료 구조]</p> <ul style="list-style-type: none"> <li>- 해시 테이블: 이중 연결 리스트에서 빠르게 찾도록 사용</li> <li>- 이중 연결 리스트: 한정된 메모리</li> </ul> <p>LRU에서 발전한 알고리즘</p> <p>= clock 알고리즘 0과 1을 가지는 비트를 둠 (1: 최근에 참조됨, 0: 최근에 참조되지 않음) 시계방향으로 돌면서 0을 발견하면 해당 프로세스를 교체하고 비트를 1로 토글</p> <p>참조 빈도가 가장 낮은 페이지를 먼저 교체</p>
--	---

## 3.4 CPU 스케줄링 알고리즘

### 3.4 CPU 스케줄링 알고리즘

#### 3.4.1 비선점형 방식 non-preemptive

CPU 스케줄러

CPU 스케줄링 알고리즘에 따라 프로세스에서 해야하는 일을 스레드 단위로 CPU에 할당

CPU 스케줄링 알고리즘

프로그램 실행 시 어떤 프로그램에 CPU 소유권을 줄지 결정

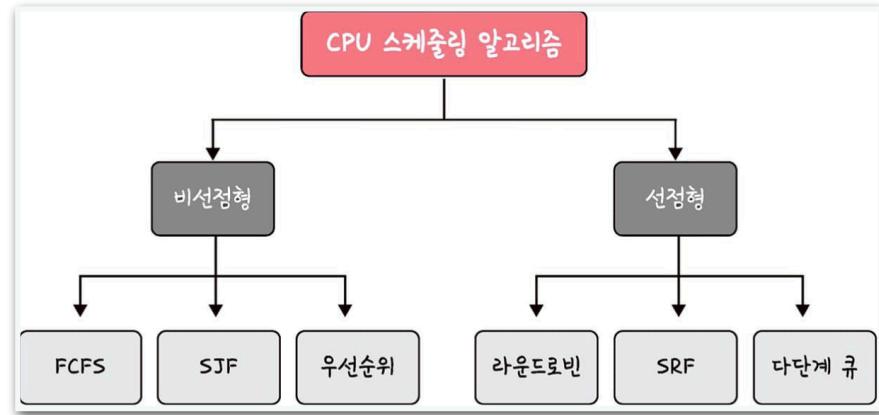
FCFS

First Come, First Served

SJF

Shortest Job First

우선순위



[목표]

CPU 이용률 높이기

주어진 시간에 많은 일 처리

준비 큐 ready queue에 있는 프로세스 수 ↓

응답 시간 ↓

프로세스가 스스로 CPU 소유권을 포기하는 방식

강제로 프로세스 중지X → 컨텍스트 스위칭으로 인한 부하 ↓

가장 먼저 온 것을 가장 먼저 처리

[단점] 길게 수행되는 프로세스로 인해 convoy effect(준비 큐에서 오래 기다리는 현상) 발생

실행 시간이 가장 짧은 프로세스를 가장 먼저 실행

실제로는 실행 시간을 알 수 없기 때문에 과거의 실행했던 시간을 토대로 추측해서 우선권 부여

[장점] 평균 대기 시간이 가장 짧음

[단점] starvation(긴 시간을 가진 프로세스가 실행되지 않는 현상) 발생

aging(오래된 작업일수록 우선순위를 높이는 작업)을 통해 SJF의 단점을 보완한 알고리즘

### 3.4.2 선점형 방식 preemptive

!라운드 로빈  
RR  
Round Robin

사용중인 프로세스를 알고리즘을 통해 중단시키고 강제로 다른 프로세스에 CPU 소유권 할당  
현대 운영체제가 사용하는 방식

현대 컴퓨터가 사용하는 스케줄링인 우선순위 스케줄링 priority scheduling의 일종  
각 프로세스가 동일한 할당시간을 부여받고,  
그 시간 안에 끝나지 않으면 다시 준비 큐 ready queue 뒤로 가는 알고리즘  
Ex) q만큼의 할당 시간이 부여되었고 N개의 프로세스 운영:  $(N-1)*q$  시간이 지나면 차례가 옴  
할당 시간이 너무 크면 FCFS가 되고 너무 짧으면 컨텍스트 스위칭이 잦아져 오버헤드(비용) 증가

[장점] 평균 응답 시간 ↓  
[단점] 전체 작업시간 ↑

로드밸런서에서 트래픽 분산 알고리즘으로도 사용

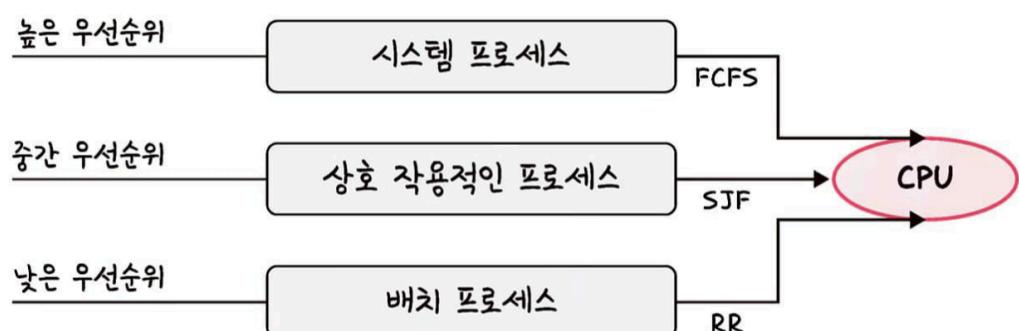
SJF + 중간에 더 짧은 작업이 들어오면 수행하던 프로세스를 중단하고 해당 프로세스 수행

SRF  
Shortest  
Remaining time First

다단계 큐

우선순위에 따른 준비 큐를 여러 개 사용  
큐마다 라운드 로빈이나 FCFS 등 다른 스케줄링 알고리즘을 적용

큐 간의 이동 X  
[장점] 스케줄링 부담 ↓  
[단점] 유연성 ↓



## Chapter 4 데이터베이스

---

## 4.1 데이터베이스의 기본

데이터베이스의 기본

### 4.1.1 엔터티

데이터베이스  
DB, DataBase

일정한 규칙, 혹은 규약을 통해 구조화되어 저장되는 데이터 모음  
실시간 접근과 동시 공유 가능

DBMS  
DataBase  
Management System

데이터베이스를 제어, 관리하는 통합 시스템

DB 내 데이터는 특정 DBMS마다 정의된  
쿼리 언어(query language)로 삽입, 삭제, 수정, 조회 가능

Ex)

MySQL이라는 DBMS에 그 위에 응용 프로그램에 속하는  
Node.js 및 php에서 해당 DB안의 데이터를 꺼내  
해당 데이터 관련 로직 구축

엔터티  
entity

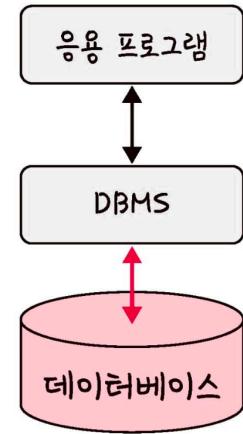
사람, 장소, 물건, 사건, 개념 등 여러 개의 속성을 지닌 명사

Ex) 회원이라는 엔터티는 이름, 아이디, 주소, 전화번호의 속성을 가짐

[약한 엔터티와 강한 엔터티]

- 약한 엔터티: 혼자서는 존재하지 못하고 다른 엔터티의 존재 여부에 따라 종속적
- 강한 엔터티: 약한 엔터티가 종속되어 있는 엔터티

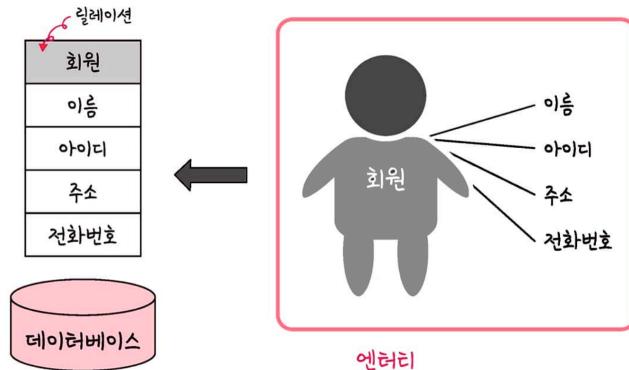
Ex) 방과 건물의 관계에서 방은 약한 엔터티, 건물은 강한 엔터티



### 4.1.2 릴레이션

#### 릴레이션 relation

데이터베이스에서 정보를 구분해 저장하는 기본 단위  
DB는 엔터티에 관한 데이터를 릴레이션 하나에 담아서 관리

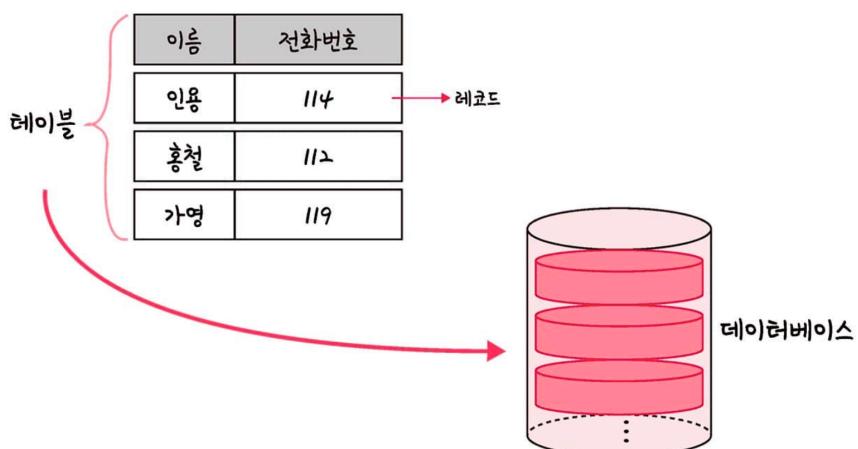


회원이라는 엔터티가 데이터베이스에서 관리될 때 릴레이션으로 변화됨

#### [테이블과 컬렉션]

관계형 데이터베이스에서는 '테이블', NoSQL 데이터베이스에서는 '컬렉션'이라고 함

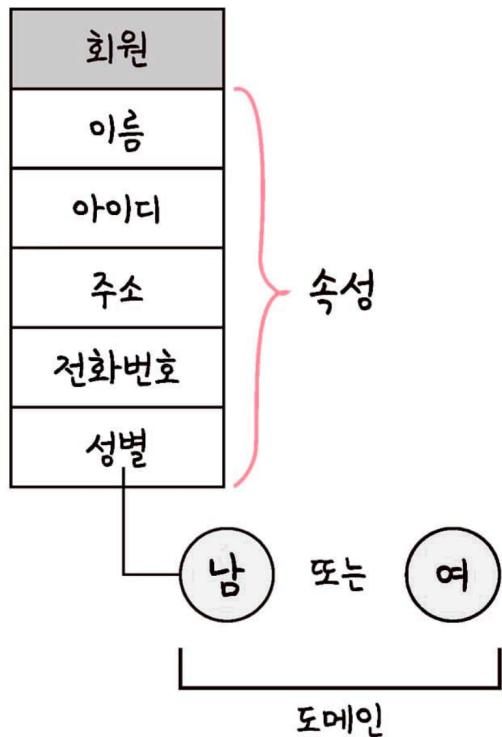
- 관계형 데이터베이스 MySQL의 구조: 레코드-테이블-데이터베이스
- NoSQL 데이터베이스 MongoDB의 구조: 도큐먼트-컬렉션-데이터베이스



레코드가 쌓여 테이블이 되고 테이블이 쌓여 데이터베이스가 된다.

### 4.1.3 속성 / 4.1.4 도메인

<b>속성 attribute</b>	릴레이션에서 관리하는 구체적이며 고유한 이름을 갖는 정보 Ex) '차'라는 엔터티의 속성으로 차 넘버, 바퀴 수, 차 색깔, 차종 등이 있을 때 서비스의 요구 사항을 기반으로 관리할 필요가 있는 속성들만 → 엔터티의 속성
<b>도메인 domain</b>	릴레이션에 포함된 각각의 속성들이 가질 수 있는 값의 집합 Ex) 성별이라는 속성이 가질 수 있는 값은 {남, 여} → 도메인



### 4.1.5 필드와 레코드

앞의 요소들을 기반으로 DB에서 필요한 필드와 레코드로 구성된 테이블을 구성

**member**

name	ID	address	phonenumbers
큰돌	kundol	서울	112
가영	kay	대전	114
빅뱅	big	카이루	119
:	:	:	:

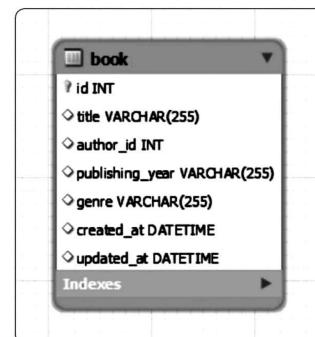
Ex) '책'이라는 엔터티를 정의하고 이를 기반으로 테이블 만들기

책의 속성: 제목, 가격, ISBN, 저자, 출판년도

→ 이름, 저자의 아이디, 출판년도, 장르, 생성 일시, 업데이트 일시를 엔터티의 속성으로 채택

테이블로 만들기 위해 속성에 맞는 타입 정의 (MySQL 기준)

- 책의 아이디: INT
- 책의 제목: VARCHAR(255)
- 책의 저자 아이디: INT
- 책의 출판년도: VARCHAR(255)
- 책의 장르: VARCHAR(255)
- 생성 일시: DATETIME
- 업데이트 일시: DATETIME



[테이블을 MySQL로 구현]

**SQL**

```
CREATE TABLE book(
    id INT NOT NULL AUTO_INCREMENT,
    title VARCHAR(255),
    author_id INT,
    publishing_year VARCHAR(255),
    genre VARCHAR(255),
    created_at DATETIME,
    updated_at DATETIME,
    PRIMARY KEY (id)
);
```

## 필드 타입

필드는 타입을 가짐  
Ex) 이름은 문자열, 전화번호는 숫자

## - 숫자 타입

MySQL의 숫자 타입

타입	용량(바이트)	최솟값(부호 있음)	최솟값(부호 없음)	최댓값(부호 없음)	최댓값(부호 있음)
TINYINT	1	-128	0	127	255
SMALLINT	2	-32768	0	32767	65535
MEDIUMINT	3	-8388608	0	8388607	16777215
INT	4	-2147483648	0	2147483647	4294967295
BIGINT	8	-263	0	263-1	264-1

## - 날짜 타입

**<DATE>**

날짜 부분은 있지만 시간 부분은 없는 값

지원 범위: 1000-01-01 ~ 9999-12-31

용량: 3 byte

**<DATETIME>**

날짜 및 시간 부분을 모두 포함하는 값

지원 범위: 1000-01-01 00:00:00 ~ 9999-12-31 23:59:59

용량: 8 byte

**<TIMESTAMP>**

날짜 및 시간 부분을 모두 포함하는 값

지원 범위: 1970-01-01 00:00:01 ~ 2038-01-19 03:14:07

용량: 4 byte

### - 문자 타입

#### 〈CHAR와 VARCHAR〉

그 안에 수를 입력해 몇 자까지 입력할지 정함  
Ex) CHAR(30): 최대 30글자까지 입력 가능

- CHAR: 테이블 생성 시 선언한 길이로 고정, 0~255
- VARCHAR: 가변 길이 문자열, 0~65,535 입력된 데이터에 따라 용량을 가변시켜 저장  
Ex.) 10의 이메일을 저장하는 경우 10글자에 해당하는 바이트 + 길이기록용 1바이트로 저장

#### 〈TEXT와 BLOB〉

큰 데이터를 저장할 때 사용하는 타입

- TEXT: 큰 문자열의 데이터 타입. 주로 게시판의 본문 저장 시 사용
- BLOB: 이미지, 동영상 등 큰 데이터의 데이터 타입 (그러나 보통은 아마존의 이미지 호스팅 서비스인 S3을 이용하는 등 서버에 파일을 올리고 파일에 관한 경로를 VARCHAR로 저장)

#### 〈ENUM와 SET〉

문자열을 열거한 타입

- ENUM: x-small, small, medium, large, x-large의 형태로 사용 (단일 선택)
  - ENUM 리스트에 없는 잘못된 값 삽입 시 빈 문자열을 대신 삽입
  - 최대 65535개의 요소 추가 가능
- SET: 비트 단위의 연산 가능, 최대 64개의 요소 추가 가능

## 4.1.6 관계

데이터베이스에는 여러 개의 테이블이 존재하고, 각 테이블끼리의 관계를 화살표로 나타냄

하나의 A는 하나의 B로 구성되어 있다:



하나의 A는 하나 이상의 B로 구성되어 있다:



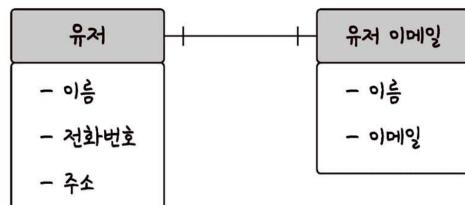
하나의 A는 하나 이하의 B로 구성되어 있다:



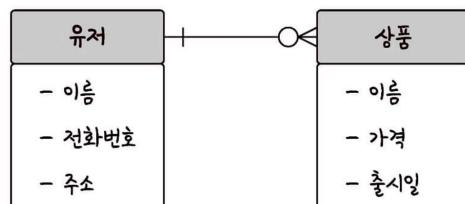
하나의 A는 0 또는 하나 이상의 B로 구성되어 있다:



### 1:1 관계

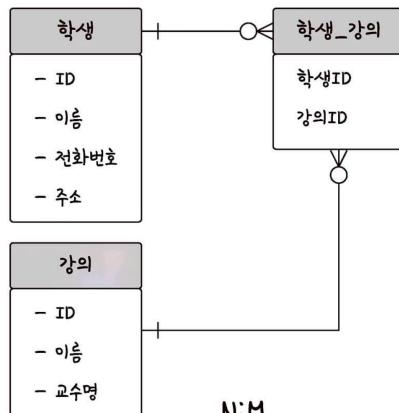


### 1:N 관계



Ex) 쇼핑몰의 유저가  
여러 개의 상품을 장바구니에 담는 경우

### N:M 관계



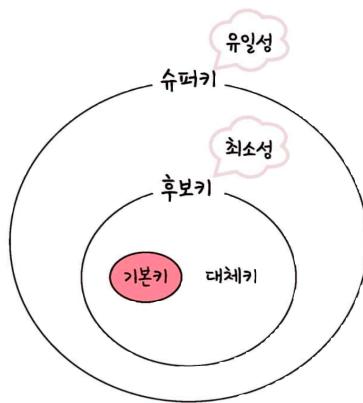
Ex) 학생과 강의의 관계  
학색이 강의를 많이 들을 수 있고  
강의도 여러 명의 학생을 포함

1:N, 1:M의 관계를 갖는 테이블 두 개로 나눠서 설정  
(테이블 두 개를 직접적으로 연결해 구축X)

### 4.1.7 키

테이블 간의 관계를 명확하게 하고 테이블 자체의 인덱스를 위해 설정된 장치

[키 간의 관계]



기본키  
PK, Primary Key

유일성과 최소성을 만족하는 키  
자연키 또는 인조키 중에 골라서 설정

ID	name
PDT-0001	홍철이의 땃스한 정퍼
PDT-0002	제호의 BMW
PDT-0002	제호의 BMW
PDT-0003	종선이의 벤츠

PDT-0002가 중복되기 때문에  
ID라는 필드는 기본키가 될 수 없다

- 자연키

유저 테이블을 만들다고 가정했을 때 속성으로 주민등록번호, 이름, 성별을 둠  
이 중 이름, 성별 등은 중복된 값이 들어올 수 있으므로 기본키로 부적절, 주민등록번호만 남음  
이렇게 자연스레 뽑다가 나오는 키를 자연키라고 함  
언젠가 변하는 속성

- 인조키

위 예시에서 인위적으로 유저 아이디를 부여하여 고유 식별자 생성  
자연키와 대조적으로 변하지 않으므로 일반적으로 기본키는 인조키로 설정

오라클은 sequence, MySQL은 auto increment 등으로 설정

외래키 FK, Foreign Key	<p>다른 테이블의 기본키를 그대로 참조하는 값 개체와의 관계 식별용 중복여부 관계 X</p> <p>Ex) client라는 테이블의 기본키인 ID가 product라는 테이블의 user_id라는 외래키로 설정됨</p> <p>user_id는 a_2라는 값이 중복됨</p> <table border="1"> <thead> <tr> <th colspan="3">client</th> </tr> <tr> <th>ID</th> <th>name</th> <th>contact</th> </tr> </thead> <tbody> <tr> <td>a_1</td> <td>주홍철</td> <td>112</td> </tr> <tr> <td>a_2</td> <td>연재호</td> <td>114</td> </tr> </tbody> </table> <table border="1"> <thead> <tr> <th colspan="3">product</th> </tr> <tr> <th>ID</th> <th>user_id</th> <th>name</th> </tr> </thead> <tbody> <tr> <td>PDT-0001</td> <td>a_2</td> <td>아우디</td> </tr> <tr> <td>PDT-0002</td> <td>a_2</td> <td>벤츠</td> </tr> <tr> <td>PDT-0003</td> <td>a_2</td> <td>BMW</td> </tr> </tbody> </table>	client			ID	name	contact	a_1	주홍철	112	a_2	연재호	114	product			ID	user_id	name	PDT-0001	a_2	아우디	PDT-0002	a_2	벤츠	PDT-0003	a_2	BMW
client																												
ID	name	contact																										
a_1	주홍철	112																										
a_2	연재호	114																										
product																												
ID	user_id	name																										
PDT-0001	a_2	아우디																										
PDT-0002	a_2	벤츠																										
PDT-0003	a_2	BMW																										
후보키 candidate key	기본키가 될 수 있는 후보들 유일성과 최소성을 동시에 만족하는 키																											
대체키 alternate key	후보키가 두 개 이상일 경우 어느 하나를 기본키로 지정하고 남은 후보키들																											
슈퍼키 Super key	각 레코드를 유일하게 식별할 수 있는 유일성을 갖춘 키																											

## 4.2 ERD와 정규화 과정

ERD와 정규화 과정

### 4.2.1 ERD의 중요성 / 4.2.2 예제로 배우는 ERD

ERD  
Entity Relationship  
Diagram

데이터베이스 구축에서 가장 기초적인 뼈대 역할: 서비스 구축 시 가장 먼저 신경 써야 할 부분  
릴레이션 간의 관계들을 정의한 것

시스템의 요구 사항을 기반으로 작성되고, 이 ERD를 기반으로 데이터베이스 구축  
데이터베이스 구축 이후엔 디버깅 또는 비즈니스 프로세스 재설계가 필요한 경우 설계도 역할 담당

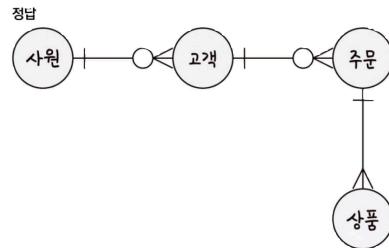
[장점] 관계형 구조로 표현 가능한 데이터를 구성하는데 유용  
[단점] 비정형 데이터를 충분히 표현할 수 없음

〈예제〉

[승원 영업부서의 ERD]

요구사항

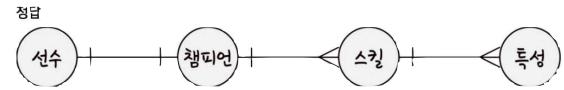
- 영업사원은 0~n명의 고객을 관리한다.
- 고객은 0~n개의 주문을 넣을 수 있다.
- 주문에는 1~n개의 상품이 들어간다.



[무무오브레전드의 ERD]

요구사항

- 선수들은 1명의 챔피언을 고를 수 있다.
- 챔피언은 한 개 이상의 스킬을 갖는다.
- 스킬은 한 개 이상의 특성을 갖는다.



### 4.2.3 정규화 과정

릴레이션 간의 잘못된 종속 관계로 인해 데이터베이스 이상 현상이 일어나서 이를 해결하거나, 저장 공간을 효율적으로 사용하기 위해 릴레이션을 여러 개로 분리하는 과정

정규형 원칙을 기반으로 정규형을 만들어가는 과정

정규화된 정도는 정규형(NF, Normal Form)으로 표현

- 기본 정규형: 제1정규형, 제2정규형, 제3정규형, 보이스/코드 정규형
- 고급 정규형: 제4정규형, 제5정규형

#### 정규형 원칙

같은 의미를 표현하는 릴레이션이지만 아래의 조건을 충족시키는 것

- 좀 더 좋은 구조
- 자료의 중복성 감소
- 독립적인 관계는 별개의 릴레이션으로 표현
- 각각의 릴레이션은 독립적인 표현 가능

#### 제1정규형

릴레이션의 모든 도메인이 더 이상 분해될 수 없는 원자 값(atomic value)만으로 구성

릴레이션의 속성 값 중, 한 개의 기본키에 대해 두 개 이상의 값을 가지는 반복 집합이 있으면 안됨  
반복 집합이 있다면 제거 필요

유저번호	유저ID	수강명	성취도
1	홍철	{C++코딩테스트, 프런트특강}	{90%, 10%}
2	범석	{코드포스특강, DS특강}	{7%, 8%}



유저번호	유저ID	수강명	성취도
1	홍철	C++코딩테스트	90%
1	홍철	프런트특강	10%
2	범석	코드포스특강	7%
2	범석	DS특강	8%

## 제2정규형

릴레이션이 제1정규형이며 부분 함수의 종속성을 제거한 형태

유저번호	유저ID	수강명	성취도
1	홍철	C++코딩테스트	90%
1	홍철	프런트특강	10%
2	범석	코드포스특강	7%
2	범석	DS특강	8%



유저번호	유저ID
1	홍철
2	범석

유저ID	수강명	성취도
홍철	C++코딩테스트	90%
홍철	프런트특강	10%
범석	코드포스특강	7%
범석	DS특강	8%

## ※주의점

- 릴레이션 분해 시 동등한 릴레이션으로 분해
- 정보 손실이 발생하지 않는 무손실 분해로 분해

제3정규형

제2정규형이고 기본키가 아닌 모든 속성이 이행적 함수 종속transitive FD를 만족하지 않는 상태

이행적 함수 종속

 $A \rightarrow B$ 와  $B \rightarrow C$ 가 존재하면 논리적으로  $A \rightarrow C$  성립: 집합 C가 집합 A에 이행적으로 함수 종속

유저ID	등급	할인율
홍철	플래티넘	30%
범수	다이아	50%
가영	마스터	70%



유저ID	등급
홍철	플래티넘
범수	다이아
가영	마스터

등급	할인율
플래티넘	30%
다이아	50%
마스터	70%

### 보이스/코드 정규형 BCNF

제3정규형이고, 결정자가 후보키가 아닌 함수 종속 관계를 제거하여  
릴레이션의 함수 종속 관계에서 모든 결정자가 후보키인 상태

#### 요구사항

- 각 수강명에 대해 한 학생은 오직 한 강사의 강의만 수강
- 각 강사는 한 수강명만 담당 가능
- 한 수강명은 여러 강사가 담당 가능

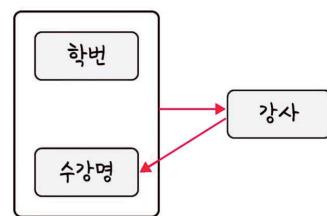
학번	수강명	강사
12010	코딩테스트	큰돌
12010	MEVN	재엽
12011	코딩테스트	큰돌
12011	MEVN	가영
NULL	롤	범석

#### 후보키

{학번, 수강명} 또는 {학번, 강사명}

범석이라는 강사가 '를'이라는 수강명을  
담당할 경우, 학번이 NULL이 되는 문제  
점 발생

#### [종속 다이어그램]



강사 속성이 결정자지만 후보키가 아니므로 분리 필요



학번	강사
12010	큰돌
12010	재엽
12011	큰돌
12011	가영

수강명	강사
코딩테스트	큰돌
MEVN	재엽
MEVN	가영
롤	범석

보이스/코드 정규형을 만족하기 때문에 룰-범석이 제대로 들어감

\*테이블을 나눈다고 성능이 100% 좋아지는 것은 아니기 때문에  
서비스에 따라 정규화 또는 비정규화 진행 필요

## 4.3 트랜잭션과 무결성

### 트랜잭션과 무결성

#### 4.3.1 트랜잭션

데이터베이스에서 하나의 논리적 기능을 수행하기 위한 작업의 단위  
여러 개의 쿼리들을 하나로 묶는 단위

[ACID 특징] 원자성, 일관성, 독립성, 지속성

원자성  
atomicity

“All or nothing”

트랜잭션과 관련된 일이 모두 수행되었거나 되지 않았거나를 보장하는 특징

예) 1000만원을 가진 흥철이가 0원을 가진 규영이에게 500만원을 이체  
결과적으로 흥철이는 500만원, 규영이는 500만원을 가지게 됨  
해당 결과는 다음과 같은 operation 단위들로 이루어진 과정을 거침

1. 흥철의 잔고 조회
2. 흥철에게서 500만원을 뺄
3. 규영에게 500만원을 넣음

데이터베이스 사용자는 이 세 가지 과정을 볼 수도, 참여할 수도 없으나  
이 과정이 모두 끝난 이후의 상황인 결과만 보게 됨

여기서 이 작업을 취소한다고 했을 때, 흥철이는 다시 1000만원, 규영이는 0원을 가져야 함  
**일부 operation만 적용된 흥철 500만원, 규영 0원이 되지 않는 것을 의미** → all or nothing

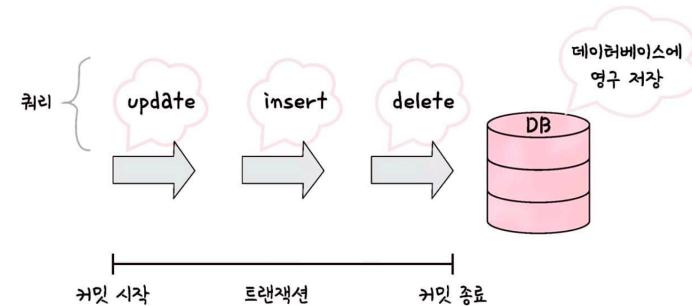
트랜잭션 단위로 여러 로직을 묶을 때 외부 API를 호출하는 것이 있으면 안됨  
만약 있다면? → 룰백이 일어났을 때 어떻게 할 것이니지에 대한 해결방법 필요  
트랜잭션 전파를 신경써서 관리해야함

- 커밋과 롤백

[커밋commit]

여러 쿼리가 성공적으로 처리되었다고 확정하는 명령어  
트랜잭션 단위로 수행되며 변경된 내용이 모두 영구적으로 저장됨

“커밋이 수행되었다” = “하나의 트랜잭션이 성공적으로 수행되었다”



## 4.4 데이터베이스의 종류

데이터베이스의 종류

### 4.4.1 관계형 데이터베이스 RDBMS

행과 열을 가지는 표 형식 데이터를 저장하는 형태의 데이터베이스

SQL라는 언어를 통해 조작: MySQL, PostgreSQL, 오라클, SQL Server, MSSQL 등

관계형 데이터베이스의 경우 표준 SQL을 지키기는 하지만 각각의 제품에 특화시킨 SQL 사용  
Ex) 오라클 - PL/SQL, SQL Server - T-SQL, MySQL - SQL

MySQL

대부분의 운영체제와 호환

현재 앱도적으로 가장 많이 사용되는 데이터베이스

C, C++ 기반

[제공 기술]

- MyISAM 인덱스 압축 기술
- B-트리 기반의 인덱스
- 스레드 기반의 메모리 할당 시스템
- 매우 빠른 조인
- 최대 64개의 인덱스 제공
- 쿼리 캐시

〈MySQL의 스토리지 엔진 아키텍처〉

모듈식 아키텍처로

쉽게 스토리지 엔진 교체 가능

[강점]

- 데이터 웨어하우징
- 트랜잭션 처리
- 고가용성 처리

스토리지 엔진 위에는 커넥터 API 및

서비스 계층을 통해 MySQL 데이터베이스와 쉽게 상호작용

PostgreSQL

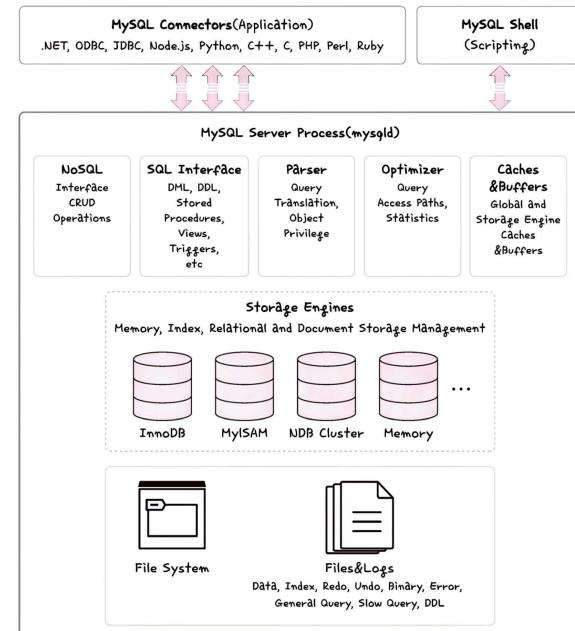
MySQL 다음으로 개발자들이 선호하는 데이터베이스 기술

VACUUM: 디스크 조각이 차지하는 영역을 회수할 수 있는 장치

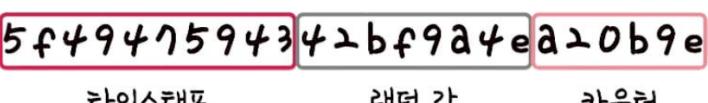
최대 테이블의 크기: 32TB

SQL뿐만 아니라 JSON을 이용해 데이터에 접근 가능

[지원 기능] 지정 시간에 복구, 로깅, 접근 제어, 중첩된 트랜잭션, 백업



#### 4.4.2 NoSQL 데이터베이스

	Not only SQL
MongoDB	<p>SQL을 사용하지 않는 데이터베이스</p> <p>JSON을 통해 데이터에 접근</p> <p>Binary JSON 형태(BSON)로 데이터 저장</p> <p>기본 스토리지 엔진: 와이드타이거 엔진</p> <p>키-값 데이터 모델에서 확장된 도큐먼트 기반의 데이터베이스</p>
	[장점]
	<ul style="list-style-type: none"> <li>뛰어난 확장성</li> <li>빅데이터 저장에 있어서 뛰어난 성능</li> <li>고가용성</li> <li>샤딩, 레플리카셋 지원</li> <li>스키마를 정해 놓지 않고 데이터를 삽입 가능 → 다양한 도메인의 데이터베이스를 기반으로 분석하거나 로깅 등의 구현 시 유용</li> </ul>
	도큐먼트 생성할 때마다 다른 컬렉션을 중복된 값을 지니기 힘든 유니크한 값인 ObjectID 생성
	 <p>타임스탬프      랜덤 값      카운터</p>
redis	<p>유닉스 시간 기반의 타임스탬프(4바이트), 랜덤 값(5바이트), 카운터(3바이트)로 구성</p> <p>인메모리 데이터베이스</p> <p>키-값 데이터 모델 기반</p> <p>[기본 데이터 타입] 문자열(string) [최대 저장 용량] 512MB</p> <p>셋(set), 해시(hash) 등 지원</p> <p>[pub/sub 기능을 통한 활용]</p> <ul style="list-style-type: none"> <li>채팅 시스템</li> <li>다른 데이터베이스 앞단에 두어 사용하는 캐싱 계층</li> <li>단순한 키-값이 필요한 세션 정보 관리</li> <li>정렬된 셋(sorted set) 자료 구조를 이용한 실시간 순위표 서비스</li> </ul>

## 4.5 인덱스

조인의 종류

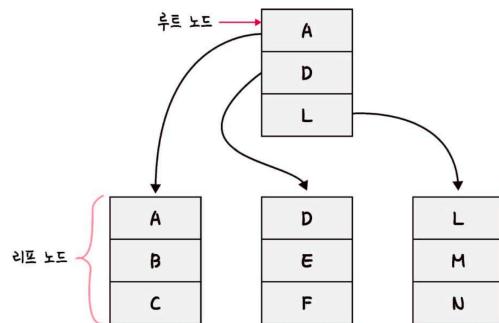
### 4.5.1 인덱스의 필요성 / 4.5.2 B-트리

#### 인덱스 index

데이터를 빠르게 찾을 수 있는 하나의 장치  
내가 찾고자 하는 테이블 내 데이터를 빠르게 탐색

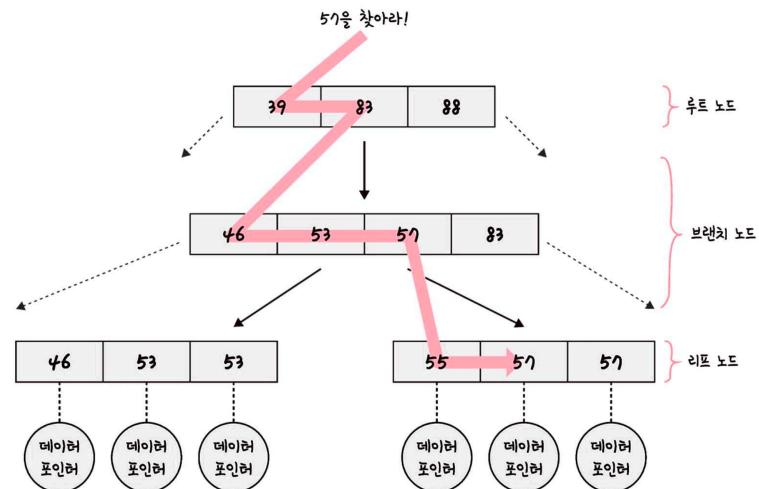
데이터베이스에서 인덱스는 일반적으로 B-트리라는 자료구조로 구성됨

[B-트리의 구성] 루트 노드, 브랜치 노드, 리프 노드



E 탐색 → 전체 테이블을 탐색하지 않고 루트 노드에서 E가 있을 만한 리프 노드로 들어가 찾음

[B-트리 예제] 57보다 같거나 클 때까지



[대수확장성]

트리 깊이가 리프 노드 수에 비해 매우 느리게 성장하는 것

기본적으로 인덱스가 한 깊이씩 증가할 때마다 최대 인덱스 항목의 수는 4배씩 증가

인덱스 항목의 수:  $3^n$  (단, n은 트리 깊이)

### 4.5.3 인덱스 만드는 방법

MySQL

[클러스터형 인덱스]

- 테이블당 하나를 설정 가능
- primary key 옵션으로 기본키로 만들면 클러스터형 인덱스 생성 가능
- 기본키로 만들지 않고 unique not null 옵션을 붙이면 클러스터형 인덱스 생성 가능

[세컨더리 인덱스]

- `create index...` 명령어 기반
- 보조 인덱스 개념: 여러 개의 필드 값을 기반으로 쿼리를 많이 보낼 때 생성하는 인덱스

Ex) `age`라는 하나의 필드만으로 쿼리를 보낸다면 클러스터 인덱스, `age`, `name`, `email` 등 다양한 필드를 기반으로 쿼리를 보낼 때는 세컨더리 인덱스 사용

하나의 인덱스만 생성하는 경우 세컨더리 보다 클러스터형 인덱스가 성능이 좋음

MongoDB

도큐먼트를 만들면 자동으로 `ObjectID` 형성, 해당 키가 자동으로 기본키로 설정됨

세컨더리키도 부가적으로 설정해 기본키와 세컨더리키를 같이 쓰는 복합 인덱스 설정 가능

#### 4.5.4 인덱스 최적화 기법

인덱스 최적화 기법은 데이터베이스마다 조금씩 다르지만 기본적인 골조는 똑같기 때문에 특정 데이터베이스를 기준으로 설명해도 무방

아래는 MongoDB 기준 인덱스 최적화 기법

##### 1. 인덱스는 비용이다

인덱스 리스트 - 컬렉션 순으로 탐색 → 두 번 탐색 강요 → 읽기 비용 발생

컬렉션 수정 시 인덱스도 수정 → B-트리의 높이를 균형있게 조절하는 비용,

데이터를 효율적으로 조회하도록 분산시키는 비용 발생

##### 2. 항상 테스팅하라

서비스에서 사용하는 객체의 깊이, 테이블의 양 등에 의해 서비스에 따라 최적화 기법이 달라짐

`explain()` 함수를 통해 인덱스를 만들고 쿼리를 보낸 후에 테스팅하며 시간 최소화

##### 3. 복합 인덱스는 같음, 정렬, 다중 값, 카디널리티 순

여러 필드 기반으로 조회 시 복합 인덱스를 생성

이 인덱스 생성 시에는 순서가 존재하며 생성 순서에 따라 인덱스 성능이 달라짐

생성 순서: 같음, 정렬, 다중 값, 카디널리티 순

1. ==이나 equal이라는 쿼리 존재 시 제일 먼저 인덱스로 설정
2. 정렬에 쓰는 필드는 그 다음 인덱스로 설정
3. 다중 값을 출력해야 하는 필드, 즉 쿼리 자체가 > 이거나 < 등 많은 값을 출력해야 하는 쿼리에 쓰는 필드라면 나중에 인덱스 설정
4. 카디널리티가 높은 순으로 생성 (ex. age와 email 중 email 필드에 대한 인덱스 우선 생성)

## 4.6 조인의 종류

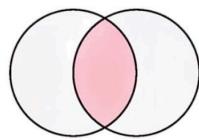
조인의 종류

조인  
join

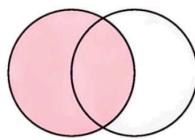
하나의 테이블이 아닌 두 개 이상의 테이블을 묶어서 하나의 결과물을 만드는 것  
MySQL에서는 JOIN이라는 쿼리로, MongoDB에서는 lookup이라는 쿼리로 처리  
그러나 MongoDB 사용 시 lookup 쿼리 사용은 가급적 자제: 성능 저하  
따라서 여러 테이블 조인 작업 시 MongoDB보다는 관계형 데이터베이스 사용

[조인의 종류]

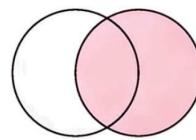
- 내부 조인 inner join: 두 테이블의 두 행이 모두 일치하는 행이 있는 부분만 표기
- 왼쪽 조인 left outer join: 왼쪽 테이블의 모든 행을 결과 테이블에 표기
- 오른쪽 조인 right outer join: 오른쪽 테이블의 모든 행을 결과 테이블에 표기
- 합집합 조인 full outer join: 두 개의 테이블 기반으로 조건을 만족시키지 않는 행까지 모두 표기



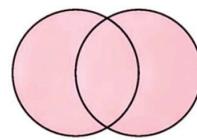
내부 조인



왼쪽 조인



오른쪽 조인



합집합 조인

## 4.7 조인의 원리

조인의 원리

### 4.7.1 중첩 루프 조인 / 4.7.2 정렬 병합 조인

#### 중첩 루프 조인

NLJ, Nested Loop Join

for문과 같은 원리로 조건에 맞는 조인을 하는 방법

랜덤 접근에 대한 비용이 크게 증가하므로 대용량 테이블에서는 사용 X

Ex) t1, t2 테이블 조인

첫 번째 테이블에서 행을 한 번에 하나씩 읽고

그다음 테이블에서도 행을 하나씩 읽어 조건에 맞는 레코드를 찾아 결과값 반환

중첩 루프 조인에서 발전한 형태 → 블록 중첩 루프 조인 BNL, Block Nested Loop

#### 정렬 병합 조인

각각의 테이블을 조인할 필드 기준으로 정렬 후 조인 작업 수행

[활용]

- 조인 시 쓸 적절한 인덱스가 없는 경우

- 대용량의 테이블 조인하고, 조인 조건으로 <, > 등 범위 비교 연산자가 있는 경우

### 4.7.3 해시 조인

#### MySQL의 해시 조인 단계

##### 해시 테이블을 기반으로 조인

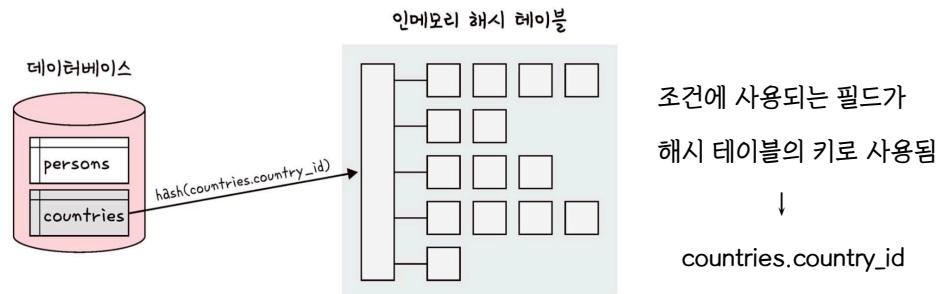
두 개의 테이블 조인 시, 하나의 테이블이 메모리에 온전히 들어갈 경우 중첩 루프 조인보다 효율적

##### [빌드 단계]

입력 테이블 중 하나를 기반으로 메모리 내 해시 테이블을 빌드

Ex) persons와 countries라는 테이블 조인

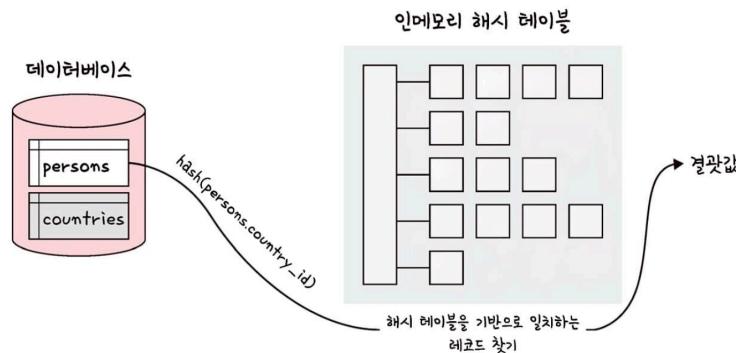
둘 중 바이트가 더 작은 테이블을 기반으로 테이블 빌드



##### [프로브 단계]

레코드를 읽기 시작하여

각 레코드에서 'persons.country\_id'에 일치하는 레코드를 찾아 결과값으로 반환



.. 각 테이블은 한 번씩만 읽게 되어 중첩해서 두 개의 테이블을 읽는 중첩 루프 조인보다 효율↑

[사용 가능한 메모리양] 시스템 변수 `join_buffer_size`에 의해 제어되며, 런타임 시 조정 가능

## Chapter 5 자료 구조

---

# 5.1 복잡도

비선형 자료 구조

## 5.1.1 시간 복잡도

자료 구조  
data structure

효율적으로 데이터를 관리하고 수정, 삭제, 탐색, 저장할 수 있는 데이터 집합

C++ → STL을 기반으로 전판적인 자료 구조를 잘 설명할 수 있는 언어

C++의 기본  
data structure

입력받은 문자열을 출력하는 프로그램 살펴보기

```
#include <bits/stdc++.h> // --- (1)
using namespace std; // --- (2)
string a; // --- (3)
int main()
{
    cin >> a; // --- (4)
    cout << a << "\n"; // --- (5)
    return 0; // --- (6)
}
```

C++는 main 함수를 중심으로 돌아가므로  
main 함수를 만들어야 한다.

이후 컴파일이 시작되면 전역변수 초기화,  
라이브러리 import 등의 작업이 일어나고,  
main 함수에 얹혀 있는 함수들이 작동된다.

(1) 헤더 파일: STL 라이브러리를 import

bits/stdc++.h는 모든 표준 라이브러리가 포함된 헤더

(2) std라는 네임스페이스를 사용한다는 의미

cin이나 cout 등을 사용할 때는 원래 std::cin처럼 네임스페이스를 달아서 호출해야 하는데,  
이를 기본으로 설정한다는 뜻

(3) 문자열 선언: <타입> <변수명> 과 같은 형태로 선언한다.

예를 들어 string a = “큰 돌” 일 때, a를 lvalue, 큰돌을 rvalue라고 한다.

lvalue는 추후 다시 사용될 수 있는 변수, rvalue는 한 번 쓰고 다시 사용되지 않는 변수

(4) 입력. 대표적으로 cin, scanf 사용

(5) 출력. 대표적으로 cout, printf 사용

(6) 프로세스가 정상적으로 마무리됨을 뜻함

## 빅오 표기법

입력 범위  $n$ 을 기준으로 해서 로직이 몇 번 반복되는지 나타내는 것

## [시간 복잡도]

문제를 해결하는 데 걸리는 시간과 입력의 함수 관계

알고리즘의 로직이 얼마나 오랜 시간이 걸리는지 나타내는데 사용하며 빅오 표기법으로 나타냄

Ex) 입력 크기  $n$ 의 모든 입력에 대한 알고리즘에 필요한 시간이  $10n^2 + n$  이라고 했을 때의 코드

```
for (int i = 0; i < 10; i++) {
    for (int j = 0; j < n; j++) {
        for (int k = 0; k < n; k++) {
            if (true) cout << k << '\n';
        }
    }
}
for (int i = 0; i < n; i++) {
    if (true) cout << i << '\n';
}
```

빅오 표기법으로 나타내면  $O(n^2)$ 이 된다.

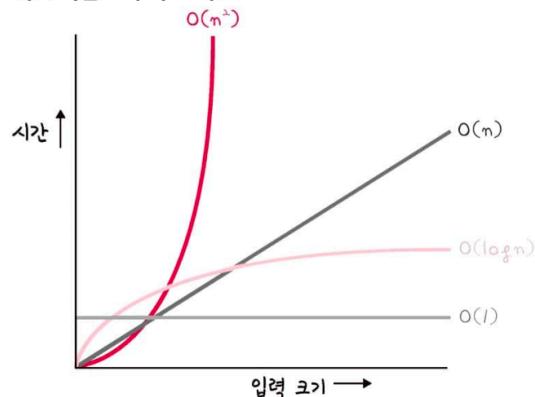
## 시간 복잡도의 존재 이유

효율적인 코드로 개선하는데 쓰이는 척도가 되기 때문

Ex) 로직이  $O(n^2)$ 의 시간 복잡도를 가지고 9초가 걸린다면

$O(n)$ 의 시간 복잡도를 가지는 알고리즘으로 개선된다면 3초가 걸리게 됨

시간 복잡도의 속도 비교



## 5.1.2 공간 복잡도 / 5.1.3 자료 구조에서의 시간 복잡도

프로그램을 실행시켰을 때 필요로 하는 자원 공간의 양

정적 변수로 선언된 것 외에 동적정적 변수로 선언된 것 외에 동적으로 재귀적인 함수로 인해 공간을 계속해서 필요로 할 경우도 포함

```
int a[1004];
```

배열 a는 1004×4바이트의 크기(공간)을 가진다

### [자료 구조의 평균 시간 복잡도]

자료 구조	접근	탐색	삽입	삭제
배열(array)	O(1)	O(n)	O(n)	O(n)
스택(stack)	O(n)	O(n)	O(1)	O(1)
큐(queue)	O(n)	O(n)	O(1)	O(1)
이중 연결 리스트(doubly linked list)	O(n)	O(n)	O(1)	O(1)
해시 테이블(hash table)	O(1)	O(1)	O(1)	O(1)
이진 탐색 트리(BST)	O(logn)	O(logn)	O(logn)	O(logn)
AVL 트리	O(logn)	O(logn)	O(logn)	O(logn)
레드 블랙 트리	O(logn)	O(logn)	O(logn)	O(logn)

### [자료 구조 최악의 시간 복잡도]

자료 구조	접근	탐색	삽입	삭제
배열(array)	O(1)	O(n)	O(n)	O(n)
스택(stack)	O(n)	O(n)	O(1)	O(1)
큐(queue)	O(n)	O(n)	O(1)	O(1)
이중 연결 리스트(doubly linked list)	O(n)	O(n)	O(1)	O(1)
해시 테이블(hash table)	O(n)	O(n)	O(n)	O(n)
이진 탐색 트리(BST)	O(n)	O(n)	O(n)	O(n)
AVL 트리	O(logn)	O(logn)	O(logn)	O(logn)
레드 블랙 트리	O(logn)	O(logn)	O(logn)	O(logn)

## 5.2 선형 자료 구조

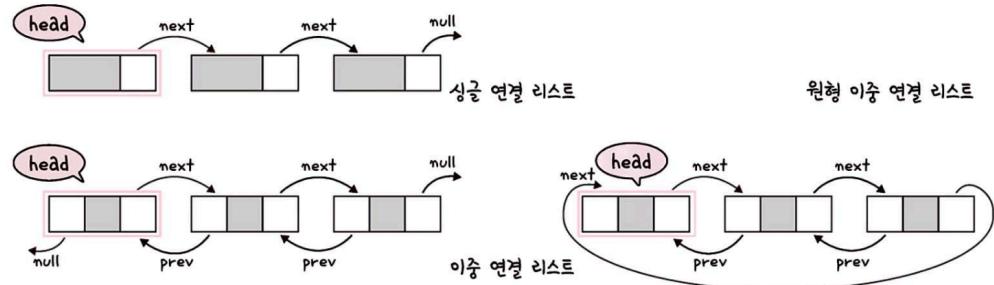
비선형 자료 구조

### 5.2.1 연결 리스트

#### 연결 리스트

데이터를 감싼 노드를 prev, next 포인터로 연결하여 공간적인 효율성 극대화시킨 자료 구조

삽입과 삭제가  $O(1)$ 이 걸리고, 탐색에는  $O(n)$ 이 걸린다



- 싱글 연결 리스트: `next` 포인터만 가짐
- 이중 연결 리스트: `next` 포인터와 `prev` 포인터를 가짐
- 원형 이중 연결 리스트: 이중 연결 리스트 + 마지막 노드의 `next` 포인터가 헤드 노드를 가리킴

<이중 연결 리스트>

```
#include <bits/stdc++.h>
using namespace std;
int main() {
    list<int> a;
    for (int i = 0; i < 10; i++) a.push_back(i);
    for (int i = 0; i < 10; i++) a.push_front(i);
    auto it = a.begin(); it++;
    a.insert(it, 1000);
    for (auto it : a) cout << it << " ";
    cout << '\n';
    a.pop_front();
    a.pop_back();
    for (auto it : a) cout << it << " ";
    cout << '\n';
    return 0;
}
/*
9 1000 8 7 6 5 4 3 2 1 0 0 1 2 3 4 5 6 7 8
1000 8 7 6 5 4 3 2 1 0 0 1 2 3 4 5 6 7 8
*/
```

`push_front()`: 앞에서부터 요소 추가

`push_back()`: 뒤에서부터 요소 추가

`insert()`: 중간에 요소 추가

## 5.2.2 배열

정적 배열 기반으로 설명

같은 타입의 변수들로 이루어져 있고, 크기가 정해져 있으며,  
인접한 메모리 위치에 있는 데이터를 모아놓은 집합  
중복을 허용하며 순서 존재

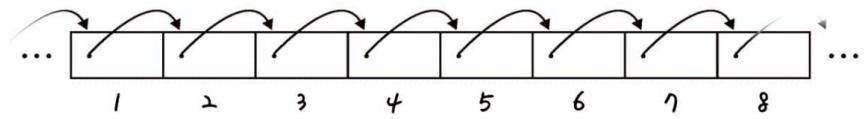
탐색에  $O(1)$ 이 되어 랜덤 접근 random access 가능  
삽입과 삭제에는  $O(n)$ 이 걸림  
∴ 데이터 추가와 삭제를 많이 하는 것은 연결 리스트, 탐색을 많이 하는 것은 배열 사용

랜덤 접근과 순차적 접근

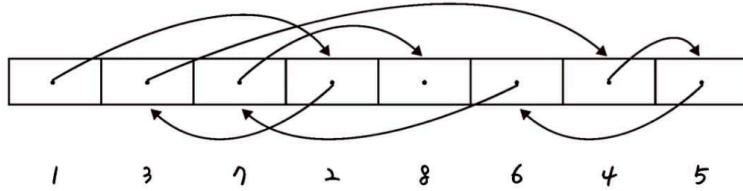
랜덤 접근: 동일한 시간에 배열과 같은 순차적인 데이터가 있을 때  
임의의 인덱스에 해당하는 데이터에 접근할 수 있는 기능

순차적 접근: 데이터를 저장된 순서대로 검색

### 순차적 접근



### 랜덤 접근



배열과 연결 리스트 비교

배열: 상자를 순서대로 나열한 데이터 구조  
몇 번째 상자인지만 알면 해당 상자의 요소 접근 가능

연결 리스트: 상자를 선으로 연결한 형태의 데이터 구조  
상자 안의 요소를 알기 위해서는 하나씩 내부를 확인해야 한다

### 5.2.3 벡터

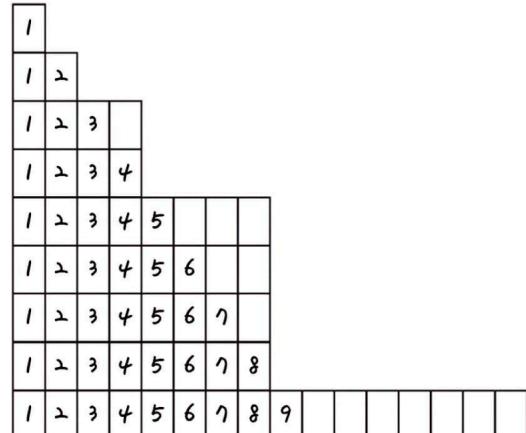
동적으로 요소를 할당할 수 있는 동적 배열  
컴파일 시점에 개수를 모른다면 벡터 사용  
증복을 허용하며, 순서 존재, 랜덤 접근 가능

탐색과 맨 뒤의 요소를 삭제하거나 삽입:  $O(1)$   
맨 뒤나 맨 앞이 아닌 요소를 삭제하고 삽입:  $O(n)$

뒤에서부터 삽입하는 `push_back()`의 경우  $O(1)$ 의 시간이 걸리는데,  
벡터의 크기가 증가되는 시간 복잡도가 amortized 복잡도,  
즉 상수 시간 복잡도  $O(1)$ 과 유사한 시간 복잡도를 가지기 때문

[`push_back()`을 할 때의 벡터 크기 증가]

함수	용량	비용
<code>push_back(1)</code>	1	1
<code>push_back(2)</code>	2	$1 + 1$
<code>push_back(3)</code>	4	$2 + 1$
<code>push_back(4)</code>	4	1
<code>push_back(5)</code>	8	$4 + 1$
<code>push_back(6)</code>	8	1
<code>push_back(7)</code>	8	1
<code>push_back(8)</code>	8	1
<code>push_back(9)</code>	16	$8 + 1$



`push_back()`을 할 때마다 매번 크기가 증가하는 것이 아닌

$2^2 + 1$ 마다 크기를 두배로 늘린다 (데이터 추가 시 남은 자리가 없으면 크기가 두 배로 증가)

`push_back()`을 할 때 드는 비용 cost:  $c_i = 1 + 2^k$

Ex) n번 `push_back()` 할 때 드는 비용  $T(n)$

$$T(n) = \sum_{i=0}^n c_i \leq n + \sum_{i=0}^{\log n} 2^i = n + 2n - 1 = 3n - 1$$

이것을 n으로 나누면 `push_back()`을 할 때 평균 비용이 되는데,

이것이 30이므로 1이라는 상수 시간보다는 크지만 상수 시간에 가까운 amortized 복잡도를 가짐

$\therefore$  `push_back()`은  $O(1)$ 의 시간 복잡도를 가진다

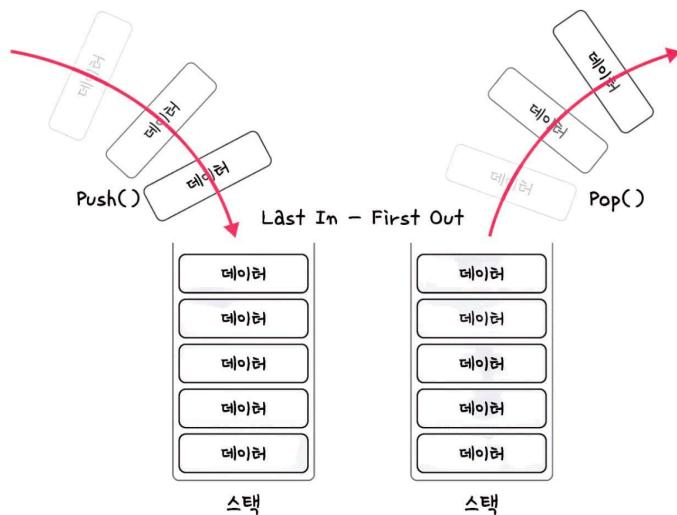
## 5.2.4 스택 / 5.2.5 큐

### -스택 stack-

가장 마지막에 들어간 데이터가 첫 번째로 나오는 성질 LIFO, Last In First Out을 가진 자료 구조

[사용] 재귀적인 함수, 알고리즘, 웹 브라우저 방문 기록 등

삽입 및 삭제에 O(1), 탐색에 O(n)이 걸린다



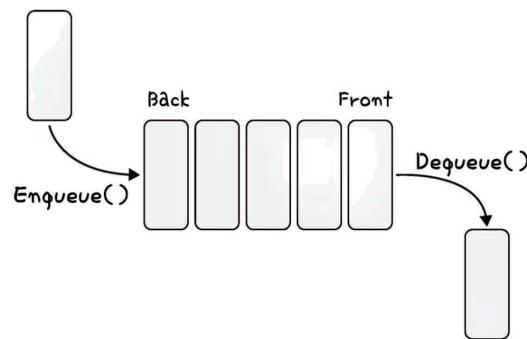
### -큐 queue-

먼저 집어넣은 데이터가 먼저 나오는 성질 FIFO, First In First Out을 가진 자료 구조

스택과 상반된 개념

[사용] CPU 작업을 기다리는 프로세스, 스레드 행렬 또는 네트워크 접속을 기다리는 행렬  
너비 우선 탐색, 캐시 등

삽입 및 삭제에 O(1), 탐색에 O(n)이 걸린다



## 5.3 비선형 자료 구조

비선형 자료 구조

### 5.3.1 그래프

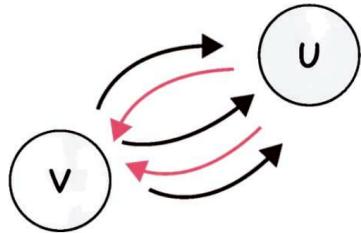
비선형 자료 구조

일렬로 나열하지 않고 자료 순서나 관계가 복잡한 구조 ex) 트리, 그래프

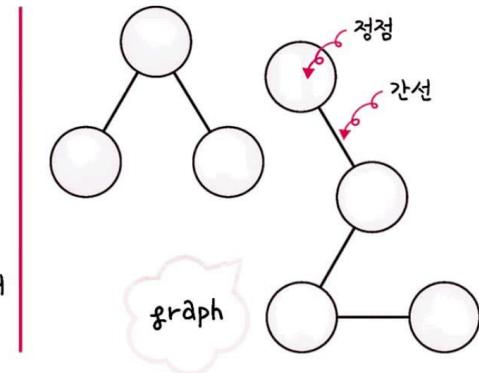
정점과 간선

어떠한 곳에서 어떠한 곳으로 무언가를 통해 간다고 했을 때

정점vertex = 어떠한 곳(장소), 간선edge = 무언가(길)



v로부터 나가는 간선: outdegree 세 개  
v로 들어오는 간선: indegree 두 개



그래프  
gragh

정점과 간선으로 이루어진 집합

가중치

간선과 정점 사이에 드는 비용

[예시]

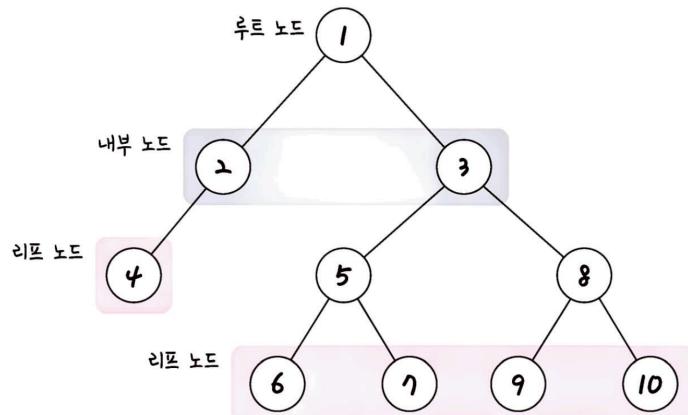
1번 노드에서 2번 노드까지 가는 비용이 한 칸이라면 1번 노드에서 2번 노드까지의 가중치: 한 칸  
성남이라는 정점에서 네이버라는 정점까지의 택시비가 13,000원일 때 가중치: 13,000원

### 5.3.2 트리

그래프 중 하나로, 정점과 간선으로 이루어져 있음

트리 구조로 배열된 일종의 계층적 데이터의 집합 (트리로 이루어진 집합 = 숲)

#### 트리의 특징



1. 부모, 자식 계층 구조  
2번 노드와 4번 노드는 각각 부모 노드, 자식 노드
2.  $V(\text{노드 수}) - 1 = E(\text{간선 수})$
3. 임의의 두 노드 사이의 경로는 '유일무이'하게 '존재'  
트리 내의 어떤 노드와 어떤 노드까지의 경로는 반드시 있음

#### 트리의 구성

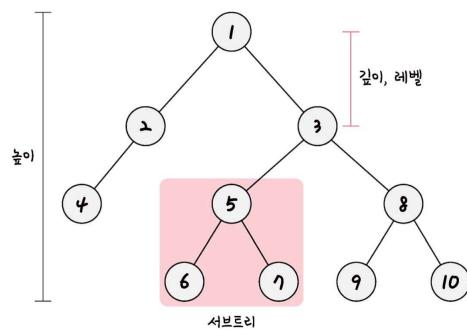
[루프 노드]: 가장 위에 있는 노드

트리 탐색 시 루트 노드 중심으로 탐색하면 문제가 쉽게 해결되는 경우가 많다

[내부 노드]: 루트 노드와 리프 노드 사이에 있는 노드

[리프 노드]: 자식 노드가 없는 노드

### - 트리의 높이와 레벨



[깊이]: 각 노드마다 다르며, 루트 노드부터 특정 노드까지 최단 거리로 갔을 때의 거리  
Ex) 4번 노드의 깊이 = 2

[높이]: 루트 노드부터 리프 노드까지 거리 중 가장 긴 거리

Ex) 트리 높이 = 3

[레벨]: 주어지는 문제마다 조금씩 상이하나, 보통 깊이와 같은 의미

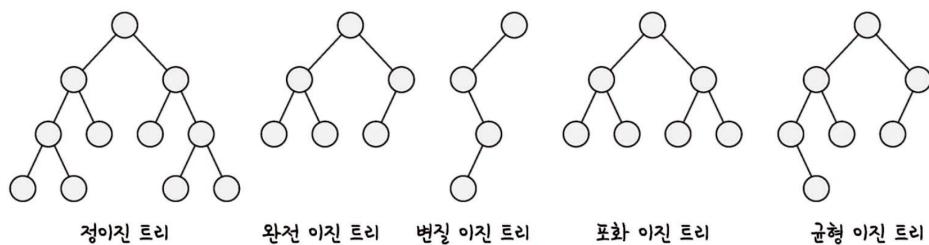
Ex) 1번 노드를 0레벨이라고 하고 2번 노드, 3번 노드까지의 레벨을 1레벨이라 할 수 있음  
1번 노드를 1레벨이라고 하면 2번 노드, 3번 노드는 2레벨

[서브트리]: 트리 내의 하위 집합, 트리 내에 있는 부분 집합

Ex) 5번, 6번, 7번 노드가 전체 트리의 서브트리

### 이진 트리

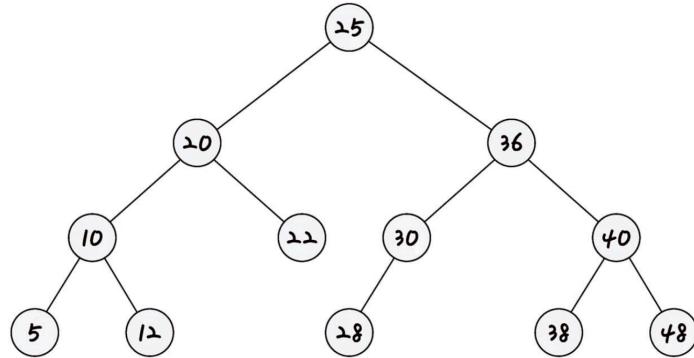
자식의 노드 수가 두 개 이하인 트리



- 정이진 트리 full binary tree: 자식 노드가 0 또는 두 개인 이진 트리
- 완전 이진 트리 complete binary tree: 왼쪽에서부터 채워져 있는 이진 트리
- 변질 이진 트리 degenerate binary tree: 자식 노드가 하나밖에 없는 이진 트리
- 포화 이진 트리 perfect binary tree: 모든 노드가 꽉 차있는 이진 트리
- 균형 이진 트리 balanced binary tree: 왼쪽과 오른쪽 노드의 높이 차이가 1 이하인 이진 트리

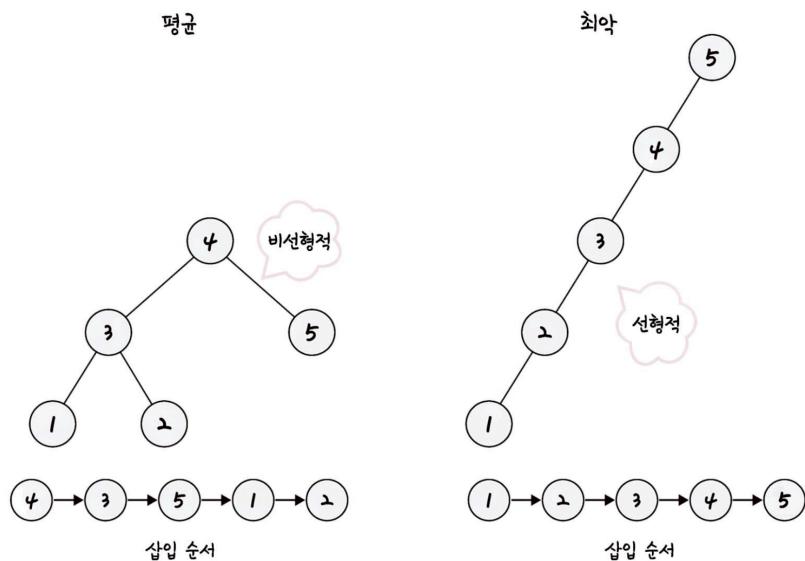
### 이진 탐색 트리 BST

노드의 오른쪽 하위 트리에는 ‘노드 값보다 큰 값’이 있는 노드만 포함,  
왼쪽 하위 트리에는 ‘노드 값보다 작은 값’이 들어 있는 트리



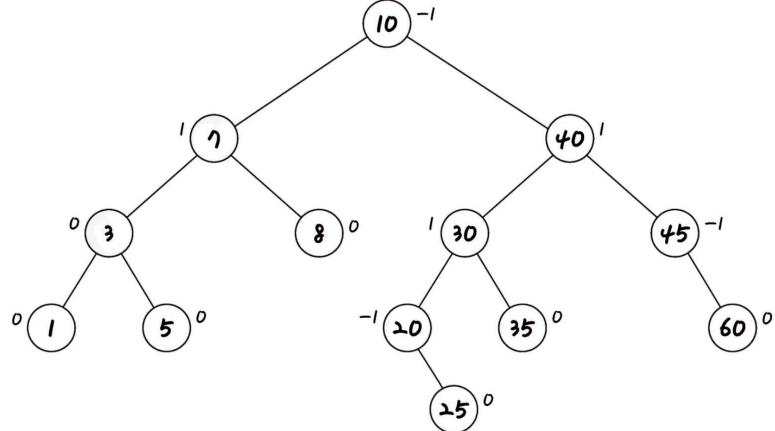
왼쪽 및 오른쪽 하위 트리도 해당 특성을 지님

검색에 용이: 탐색 시  $O(\log n)$ 이 걸리지만 최악의 경우  $O(n)$ 이 걸린다  
삽입 순서에 따라 선형적일 수 있기 때문



### AVL 트리 Adelson-Velsky and Landis tree

삽입 순서에 따른 최악의 경우 선형적 트리가 되는 것을 방지,  
스스로 균형을 잡는 이진 탐색 트리



탐색, 삽입, 삭제 모두 시간 복잡도가  $O(\log n)$

삽입, 삭제를 할 때마다 불균형 방지를 위해 트리 일부를 한 방향으로 회전시키며 균형을 잡는다

### 레드 블랙 트리

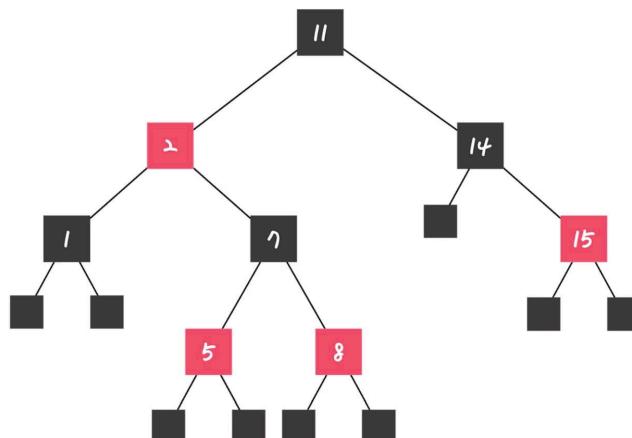
균형 이진 탐색 트리의 한 종류

탐색, 삽입, 삭제 모두 시간 복잡도가  $O(\log n)$

각 노드는 빨간색 또는 검은색의 색상을 나타내는 추가 비트 저장,  
삽입 및 삭제 중에 트리가 균형을 유지하도록 하는 데 사용

“모든 리프 노드와 루트 노드는 블랙이고 어떤 노드가 레드이면 그 노드의 자식은 반드시 블랙이다”

Ex) C++ STL의 set, multiset,  
map, multimap



### 5.3.3 힙

완전 이진 트리 기반의 자료 구조

[종류] 최소힙, 최대힙

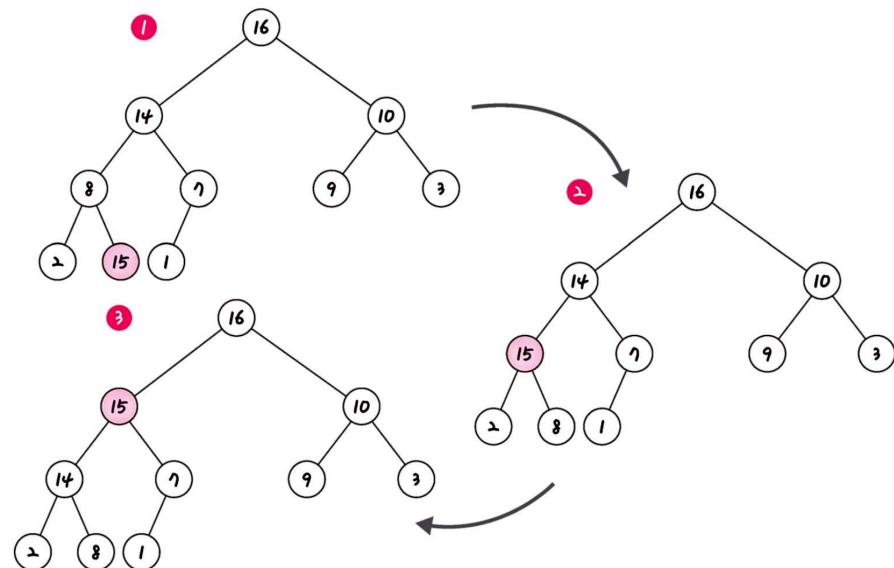
해당 힙에 따라 특정한 특징을 지킨 트리

- 최소힙: 루트 노드에 있는 키는 모든 자식에 있는 키 중에서 가장 작아야함
  - 최대힙: 루트 노드에 있는 키는 모든 자식에 있는 키 중에서 가장 커야함
- \* 각 노드의 자식 노드와의 관계도 이와 같은 특징이 재귀적으로 이루어짐

#### 최대힙의 삽입

새로운 노드를 힙의 마지막 노드에 이어서 삽입

그 새로운 노드를 부모 노드들과 크기를 비교하여 교환(스왑)해서 힙의 성질 만족시킴



#### 최대힙의 삭제

최대힙에서 최댓값은 루트 노드이므로 루트 노드가 삭제,

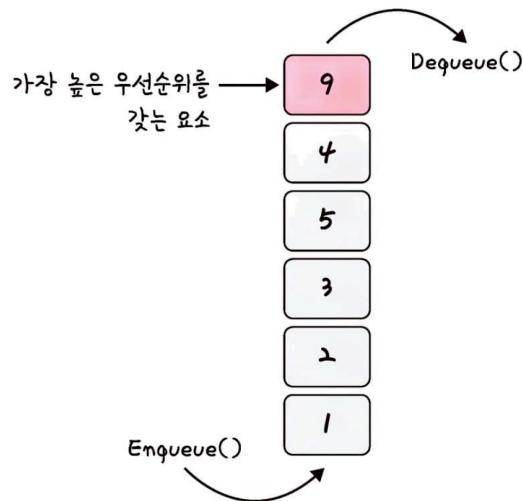
그 이후 마지막 노드와 루트 노드를 스왑하여 또다시 스왑 등의 과정을 거쳐 재구성

### 5.3.4 우선순위 큐

= 우선순위 대기열

대기열에서 우선순위가 높은 요소가 우선순위가 낮은 요소보다 먼저 제공되는 자료 구조

힙을 기반으로 구현



```
#include <bits/stdc++.h>
using namespace std;
priority_queue<int, vector<int>, greater<int> > pq; // 오름차순
//priority_queue<int, vector<int>, less<int> > pq; // 내림차순
int main() {
    pq.push(5);
    pq.push(4);
    pq.push(3);
    pq.push(2);
    pq.push(1);
    cout << pq.top() << "\n";
    return 0;
}
/*
1
*/
```

greater: 오름차순    less: 내림차순

오름차순으로 정렬하게 하고 5, 4, 3, 2, 1이 입력되었음에도 우선순위가 높은 1이 출력됨

### 5.3.4 맵 / 5.3.6 셋 / 5.3.7 해시 테이블

<b>맵 map</b> <b>셋 set</b> <b>해시 테이블 set</b>	<p>특정 순서에 따라 키와 매핑된 값의 조합으로 형성된 자료 구조</p> <p>Ex) “이승철” : 1, “박동영” : 2 처럼 string : int 값을 할당해야하는 경우 사용</p> <p>레드 블랙 트리 자료 구조 기반으로 형성</p> <p>삽입 시 자동 정렬</p> <p>해시 테이블 구현 시 사용</p> <p>정렬을 보장 유무에 따라 unordered_map, map 사용</p> <p>특정 순서에 따라 고유한 요소를 저장하는 컨테이너</p> <p>중복이 없으며 오로지 희소한 unique 값만 저장하는 자료 구조</p> <p>무한에 가까운 데이터들을 유한한 개수의 해시 값으로 매핑한 테이블</p> <p>삽입, 삭제, 탐색 시 평균적으로 O(1)의 시간 복잡도를 가짐</p> <p>unordered_map으로 구현</p>
---	--

# Chapter 1

## 디자인 패턴과 프로그래밍 패러다임

## 1.1 디자인 패턴

디자인 패턴

### 1.1.1 싱글톤 패턴

<p><b>디자인 패턴</b></p> <p><b>싱글톤 패턴</b> singleton pattern</p> <p>싱글톤 패턴의 단점</p> <p>의존성 주입 DI, Dependency Injection</p>	<p>프로그램을 설계할 때 발생했던 문제점들을 객체 간의 상호 관계 등을 이용하여 해결할 수 있도록 하나의 ‘규약’ 형태로 만들어 놓은 것</p> <p>하나의 클래스에 오직 하나의 인스턴스만 가지는 패턴</p> <p>일반적으로 데이터베이스 연결 모듈에 사용</p> <p>[장점] 사용하기 쉽고 실용적     인스턴스 생성 비용 감소     : 하나의 인스턴스를 만들어 놓고 해당 인스턴스를 다른 모듈들이 공유하여 사용</p> <p>[단점] 의존성 증가</p> <p>TDD Test Driven Development 를 할 때 걸림돌이 된다 TDD를 할 때 단위 테스트를 주로는데, 단위 테스트가 서로 독립적이어야하고 테스트를 어떤 순서로든 실행 가능해야 함</p> <p>싱글톤 패턴은 미리 생성된 하나의 인스턴스를 기반으로 구현하는 패턴이기 때문에 각 테스트마다 독립적인 인스턴스 생성 어려움</p> <p>모듈 간의 결합을 강하게 만들 수 있다는 단점을 의존성 주입을 통해 해결 : 모듈 간의 결합을 느슨하게 만듦</p> <p>메인 모듈 main module 이 직접 다른 하위 모듈에 대한 의존성을 주기보다, 의존성 주입자 dependency injector 를 이 부분을 가로채 메인 모듈이 간접적으로 의존성을 주입하는 방식 = 디커플링</p> <p>- 의존성 주입의 장점</p> <p>- 의존성 주입의 단점</p> <p>- 의존성 주입의 원칙</p>

## 1.1.2 팩토리 패턴 / 1.1.3 전략 패턴 / 1.1.4 옵저버 패턴

### 팩토리 패턴 factory pattern

객체를 사용하는 코드에서 객체 생성 부분을 떼어내 추상화한 패턴

상위 클래스: 중요한 빠대 결정

하위 클래스: 객체 생성에 관한 구체적인 내용 결정

상위 클래스와 하위 클래스가 분리되기 때문에 느슨한 결합

상위 클래스에서는 인스턴스 생성 방식에 대해 전혀 알 필요 X → 유연성 ↑

객체 생성 로직이 따로 떼어져 있어 코드를 리팩터링해도 한 곳만 고칠 수 있음 → 유지 보수성 ↑

### 전략 패턴 strategy pattern

= 정책 패턴 policy pattern

객체의 행위를 바꾸고 싶은 경우 직접 수정하지 않고

전략이라고 부르는 캡슐화한 알고리즘을 컨텍스트 안에서 바꿔주면서 상호 교체

### 옵저버 패턴 observer pattern

주체가 어떤 객체 subject의 상태 변화를 관찰하다가

상태 변화가 있을 때마다 메서드 등을 통해 목록에 있는 옵저버들에게 변화를 알려주는 패턴

주체와 객체를 따로 두지 않고 상태가 변경되는 객체를 기반으로 구축하기도 함

Ex) 트위터: 어떤 사람인 주체를 ‘팔로우’했을 때, 주체가 포스팅을 올리게 되면 ‘팔로워’에게 알림

주로 이벤트 기반 시스템에 사용

MVC Model–View–Controller 패턴에도 사용

Ex) 주체라고 볼 수 있는 모델 model에서 변경 사항이 생겨

update() 메서드로 옵저버인 뷰에 알려주고 이를 기반으로 컨트롤러 controller 등이 작동

### 1.1.5 프록시 패턴과 프록시 서버

#### 프록시 패턴 proxy pattern

대상 객체 subject에 접근하기 전 그 접근에 대한 흐름을 가로채 대상 객체 앞단의 인터페이스 역할



객체의 속성, 변환 등을 보완하여 보완, 데이터 검증, 캐싱, 로깅에 사용

프록시 객체로 쓰이기도 하지만 프록시 서버로도 활용

#### 프록시 서버 proxy server

서버와 클라이언트 사이에서 클라이언트가 자신을 통해

다른 네트워크 서비스에 간접적으로 접속할 수 있게 해주는 컴퓨터 시스템이나 응용 프로그램

##### - nginx

비동기 이벤트 기반의 구조

다수의 연결을 효과적으로 처리 가능한 웹 서버

주로 Node.js 서버 앞단의 프록시 서버로 활용

##### - CloudFlare

전 세계적으로 분산된 서버 존재

어떠한 시스템의 콘텐츠 전달을 빠르게 할 수 있는 CDN 서비스

[장점] ↓ 웹 서버 앞단에 두어 '프록시 서버'로 쓰기 때문에 가능

- DDOS 공격 방어: 의심스러운 트래픽, 특히 사용자가 접속하는 것이 아닌

시스템을 통해 오는 트래픽을 자동으로 차단

거대한 네트워크 용량과 캐싱 전략으로 소규모 DDOS 공격을 쉽게 방어

공격에 대한 방화벽 대시보드 제공

- HTTPS 구축: 서버에서 HTTPS 구축 시 인증서 기반으로 구축할 수 있으나,

CloudFlare 사용 시 별도의 인증서 설치 필요 X

### - CORS와 프런트엔드의 프록시 서버

CORS Cross-Origin Resource Sharing: 서버가 웹 브라우저에서 리소스를 로드할 때 다른 오리진을 통해 로드하지 못하게 하는 HTTP 헤더 기반 메커니즘

프런트엔드 개발 시 프런트엔드 서버를 만들어 백엔드 서버와 통신할 때 주로 CORS 에러를 마주함 이를 해결하기 위해 프런트엔드에서 프록시 서버 생성하는 경우도 있다.

Ex) 프런트엔드에서는 12.0.0.1:3000으로 테스팅하는데 백엔드 서버는 127.0.0.1:12010이면 포트 번호가 다르기 때문에 CORS 에러 발생  
해결방법 → 프록시 서버를 두어 프런트엔드 서버에서 요청되는 오리진을 127.0.0.1:12010으로 변경