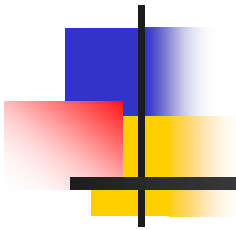
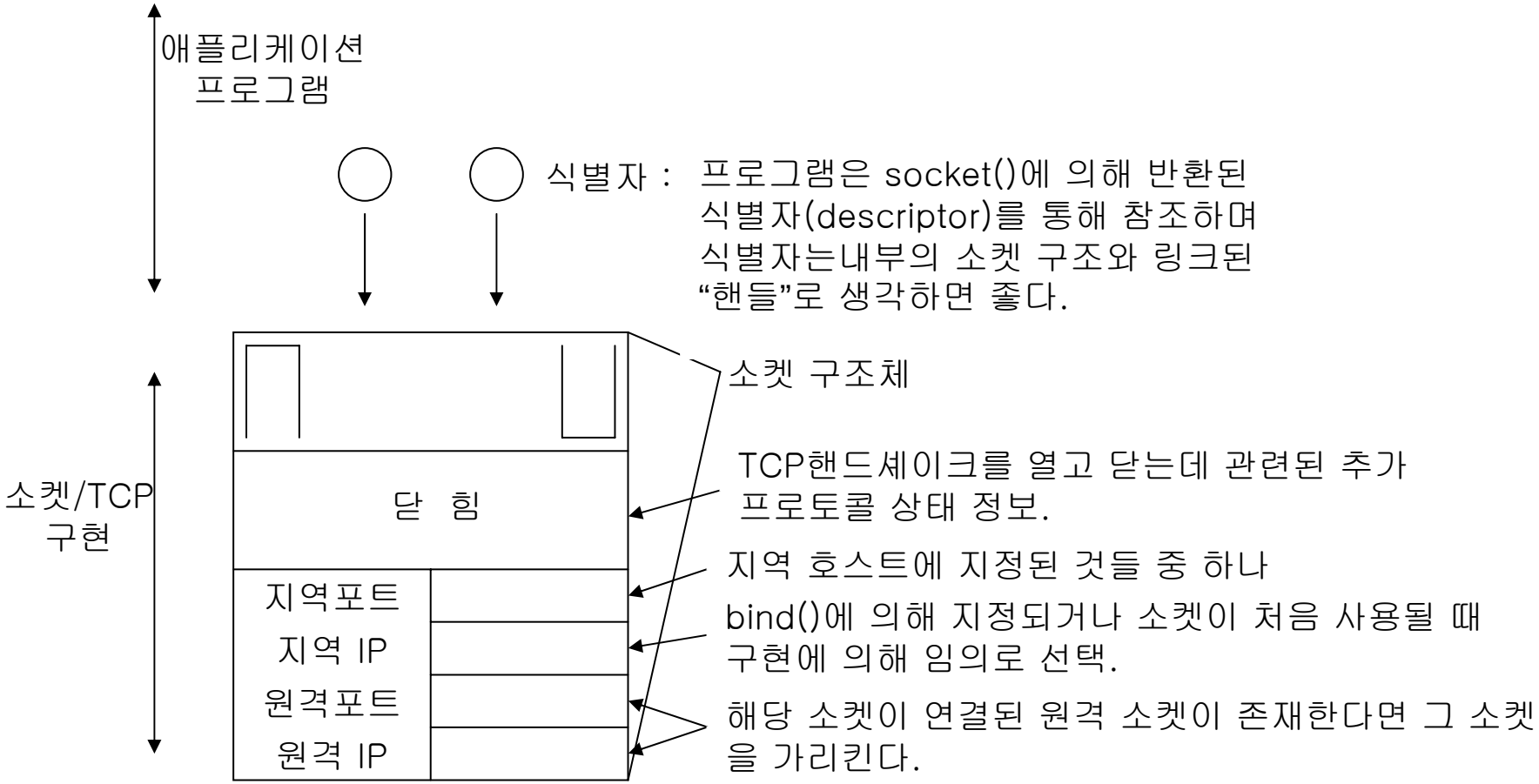


Chapter 6 내부구조

TCP 소켓 관련 자료 구조들

♬ 소켓 구조체는 송신 및 수신 큐와 다음을 포함하는 기타 정보를 가진다.



6.1 버퍼링(Buffering)과 TCP

- 시간축의 어떤 특정 순간까지 보내진 모든 바이트들의 순열을 다음 세 FIFO"큐들"로 나눈다.

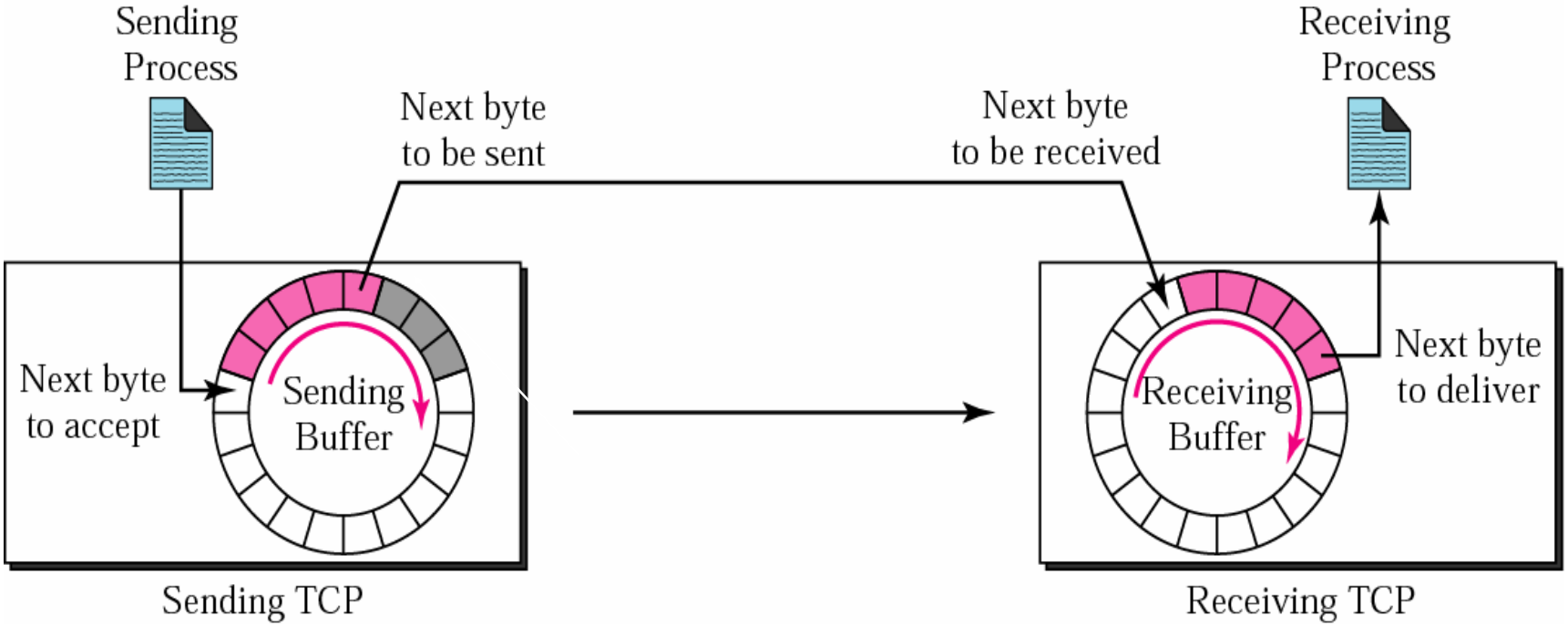
▷ SendQ : 송신자의 소켓 계층에서 버퍼링된, 받는 쪽 호스트로 아직 성공적으로 전송 되지 않은 바이트들.

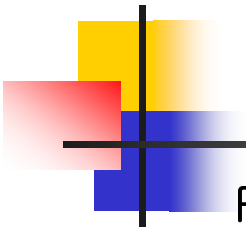
▷ RecvQ : 수신자의 소켓 계층에서 버퍼링된, 받는 프로그램으로 전달되기를 기다리는 즉 recv()에 의해 반환되기를 기다리는 바이트들.

▷ Delivered : recv()에 의해 받는 프로그램으로 반환된 바이트들.

TCP 서비스

- 송신버퍼와 수신 버퍼
  - TCP보관을 위한 버퍼 필요 – 동일속도로 데이터를 생성하거나 소비하지 않을 수 있다.
  - 한가지 구현방법은 순환 배열을 사용





```
Rv = connect(s, . . .);
.
.
.
Rv = send(s, buffer0, 1000, 0);
.
.
.
Rv = send(s, buffer1, 2000, 0);
.
.
.
Rv = send(s, buffer2, 5000, 0);
.
.
.
close(s);
```

이 TCP 연결은 수신자에게 8000바이트를 전송한다. 이들 8000바이트가 연결의 받는쪽 종단에서 전달을 위해 묶여지는 방법은 연결의 두 종단에서 send()와 recv()호출들의 발생 시각에 의존한다.

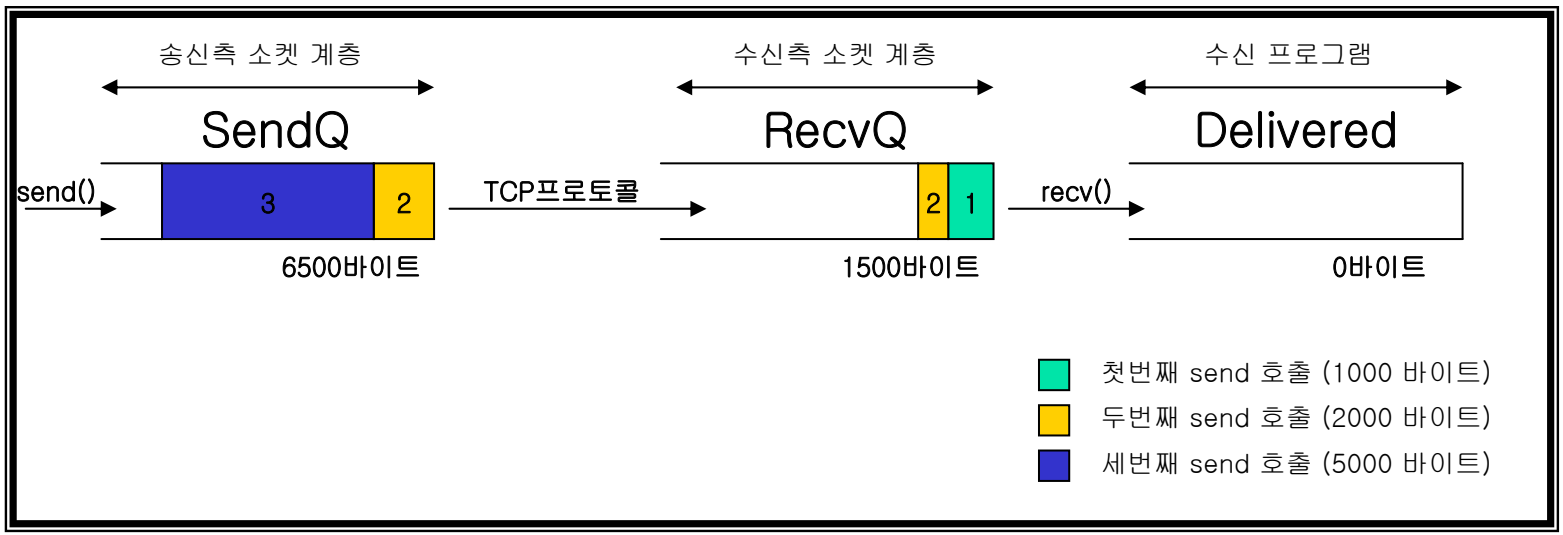
- *send() 호출은 SendQ에 바이트들을 추가하는 작업!*

  TCP프로토콜 하부구조가 SendQ로부터 RecvQ로 바이트들을 순서대로 옮긴다.

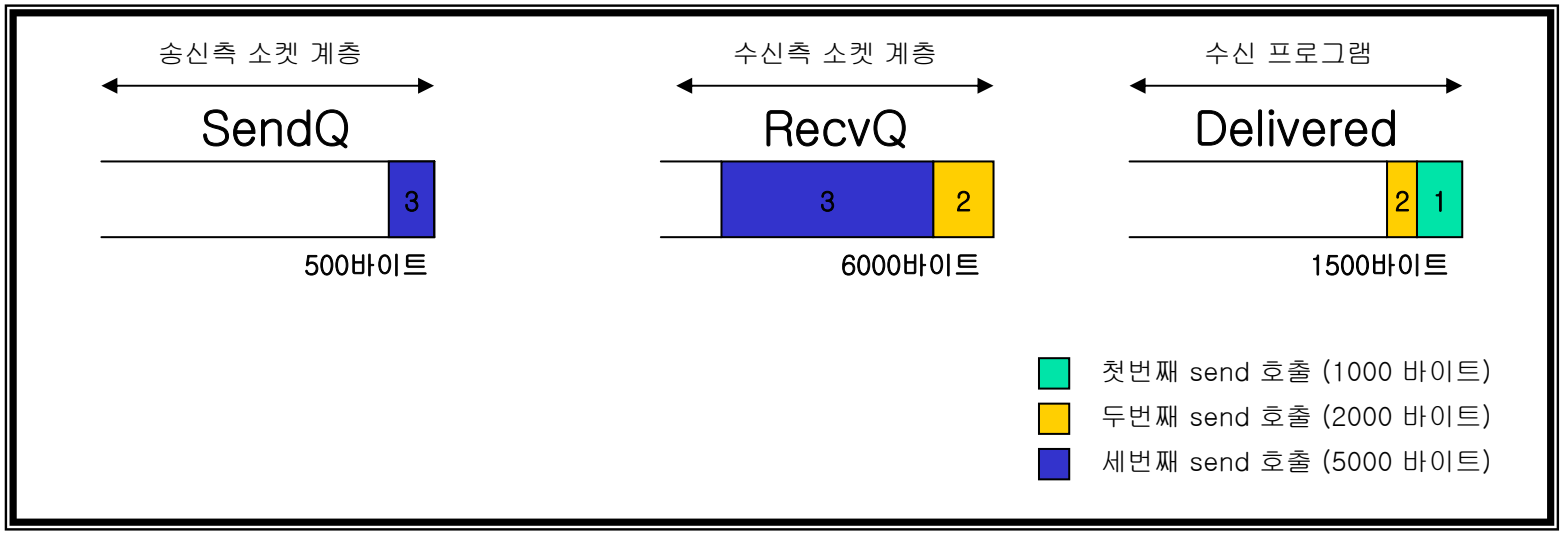
  이 이동은 사용자 프로그램에 의해 제어되거나 직접 관찰될 수 없다.

  옮겨지는 덩어리의 크기는 RecvQ의 데이터량과 recv()에 주어진 버퍼의 크기에 의존한다.

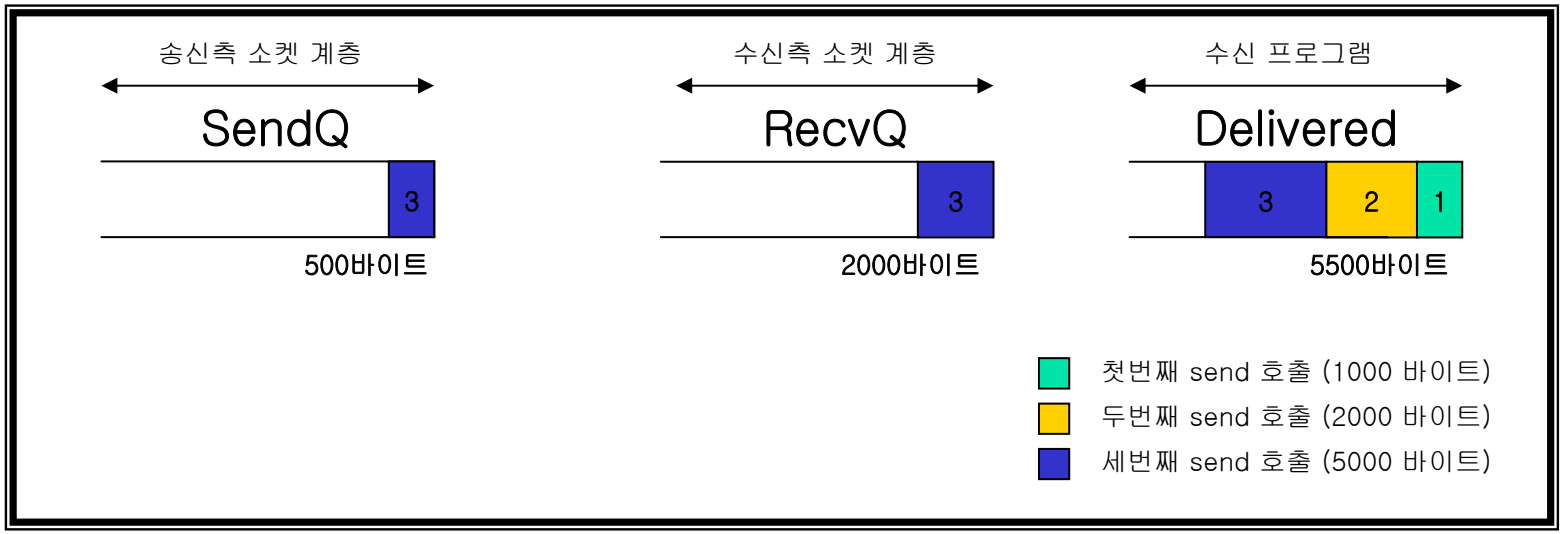
◆ send()의 세번 호출 후 각 큐들의 상태



◆ 첫번째 recv() 이후

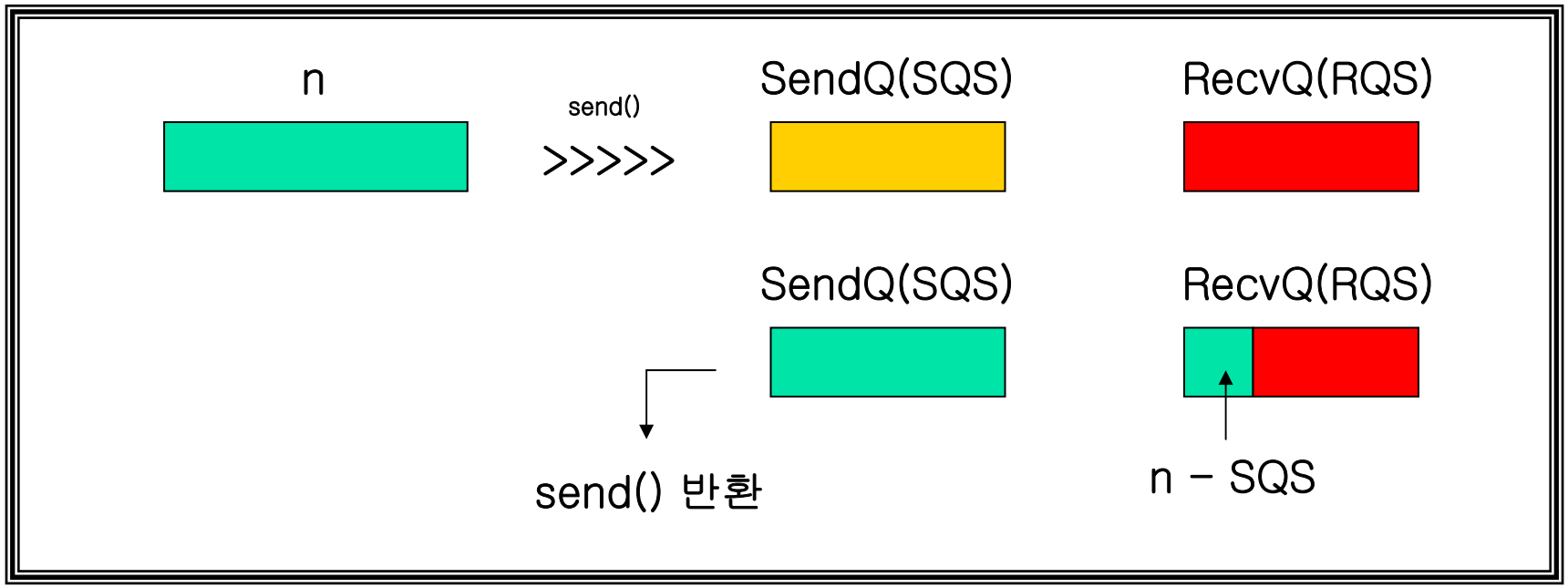


◆ 추가의 recv() 이후

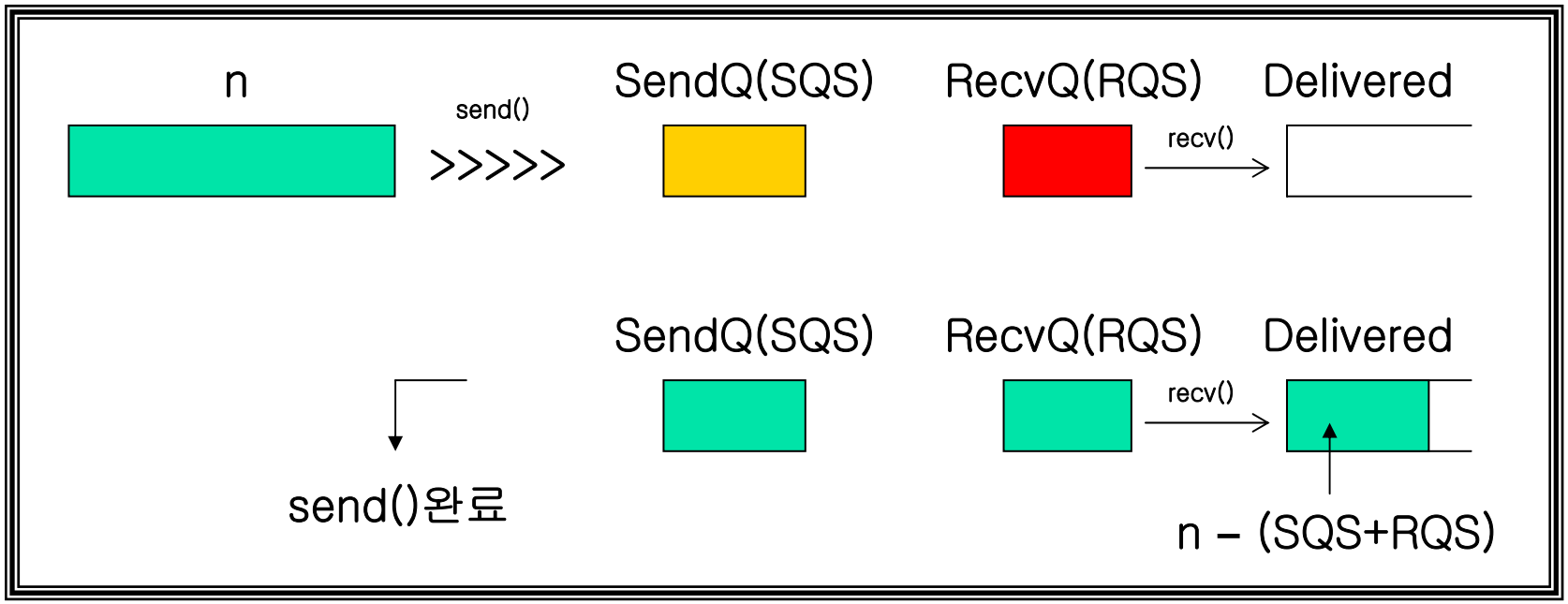


6.2 교착상태

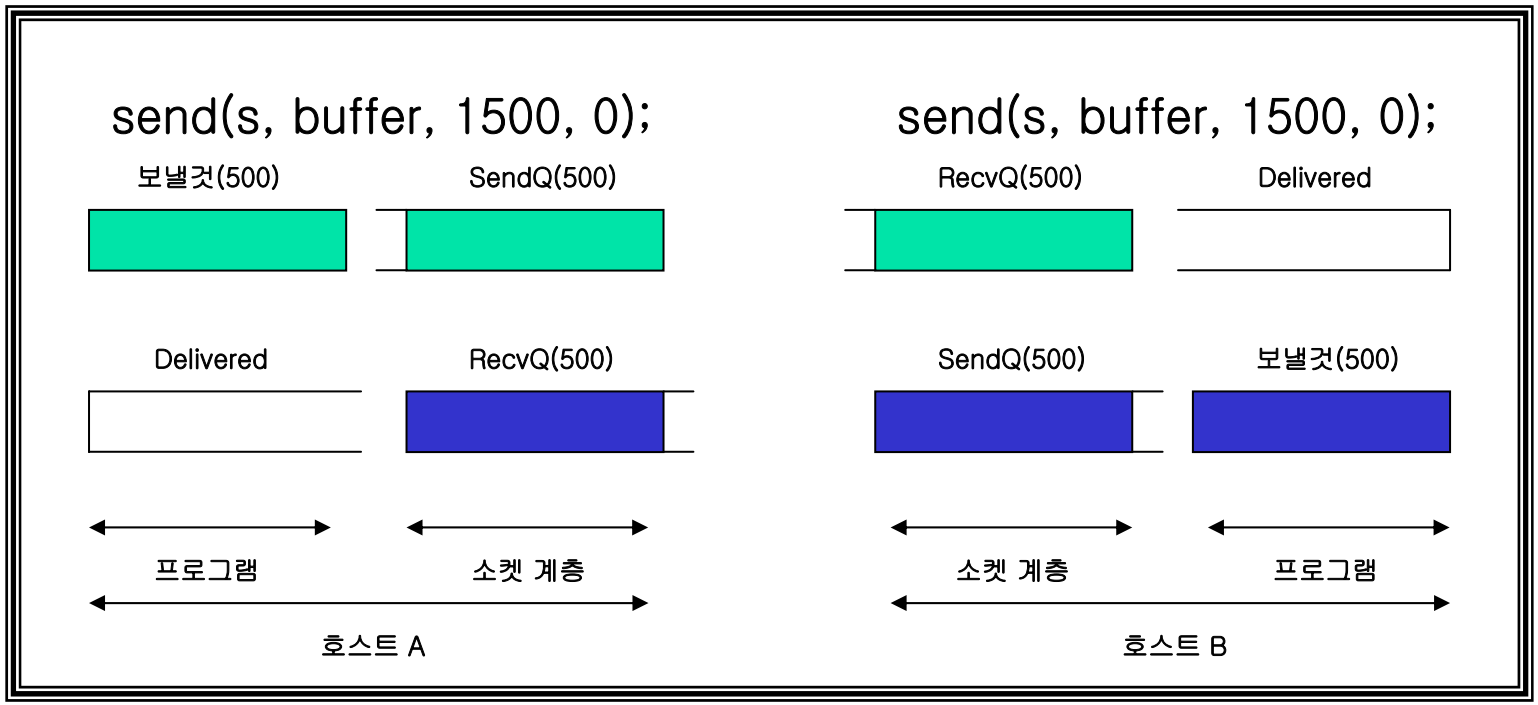
- SendQ와 RecvQ의 크기를 각각 SQS와 RQS라 하자.
  - ▶ 전송될 바이트의 수 n(n>SQS)을 가지는 send() 호출시
    - 수신 호스트에서는 적어도 n – SQS바이트가 RecvQ로 전달될 때까지 반환하지 않을 것이다.



- 전송될 바이트 수 n이 SQS + RQS를 초과시
  - send()는 수신 프로그램이 recv()를 여러 번 충분히 호출하거나 적어도 n – (SQS + RQS) 바이트의 데이터를 전달할 수 있는 충분한 크기의 버퍼를 가지고 호출하기 전까지는 반환할 수 없다.



- 만약 양 종단에서 SQS+RQS 보다 큰 크기의 파라미터들을 가지고 send()를 *동시에 호출* 한다면 교착상태가 생겨서 어떤 send()도 완료되지 않고 양 프로그램은 영원히 블록된 상태로 남게 될 것이다.



위에 그림이 교착상태의 모습을 보여주는 것으로 SQS와 RQS의 크기는 500이다. 이 상황에서는 어느쪽도 빈 공간이 생기기 전까지는 반환되어지지 않으므로 send()는 결코 완료되지 않는다.

6.3 성능 관련 사항.

- 사용자 데이터를 재전송에 대비하여 SendQ에 복사해 놓을 필요성은 성능의 문제를 유발시킨다.
  - 특히 SendQ 및 RecvQ 버퍼의 크기는 한 TCP 연결을 통해 달성할 수 있는 처리량(throughput)에 영향을 미친다.
  → 네트워크 용량이나 다른 제한이 없다면 더 큰 버퍼들이 일반적으로 더 높은 처리량을 가져온다.
  → 그 이유는 데이터를 커널 버퍼들의 내부로 혹은 외부로 이동하는데 드는 비용과 관계가 있다.

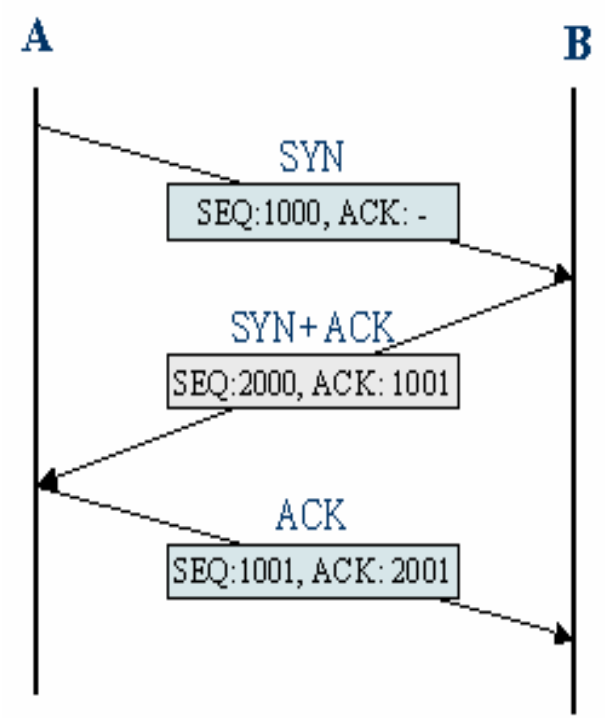
  ~ SQS보다 큰 n바이트를 send()로 호출한다면 시스템 데이터를 여러 단위로 나누어 전송해야 한다.
  매번 소켓 계층이 데이터가 SendQ로부터 이동하는 것을 기다릴 때마다 어느 정도의 시간이 부하로서 낭비된다.

※ send() 호출의 효율적인 크기는 실제 SQS에 의해 제한을 받고 recv() 호출의 효율적인 크기는 RQS에 의해 제한을 받게 된다.

6.4 TCP 소켓 생명 주기 (TCP Socket Life Cycle)

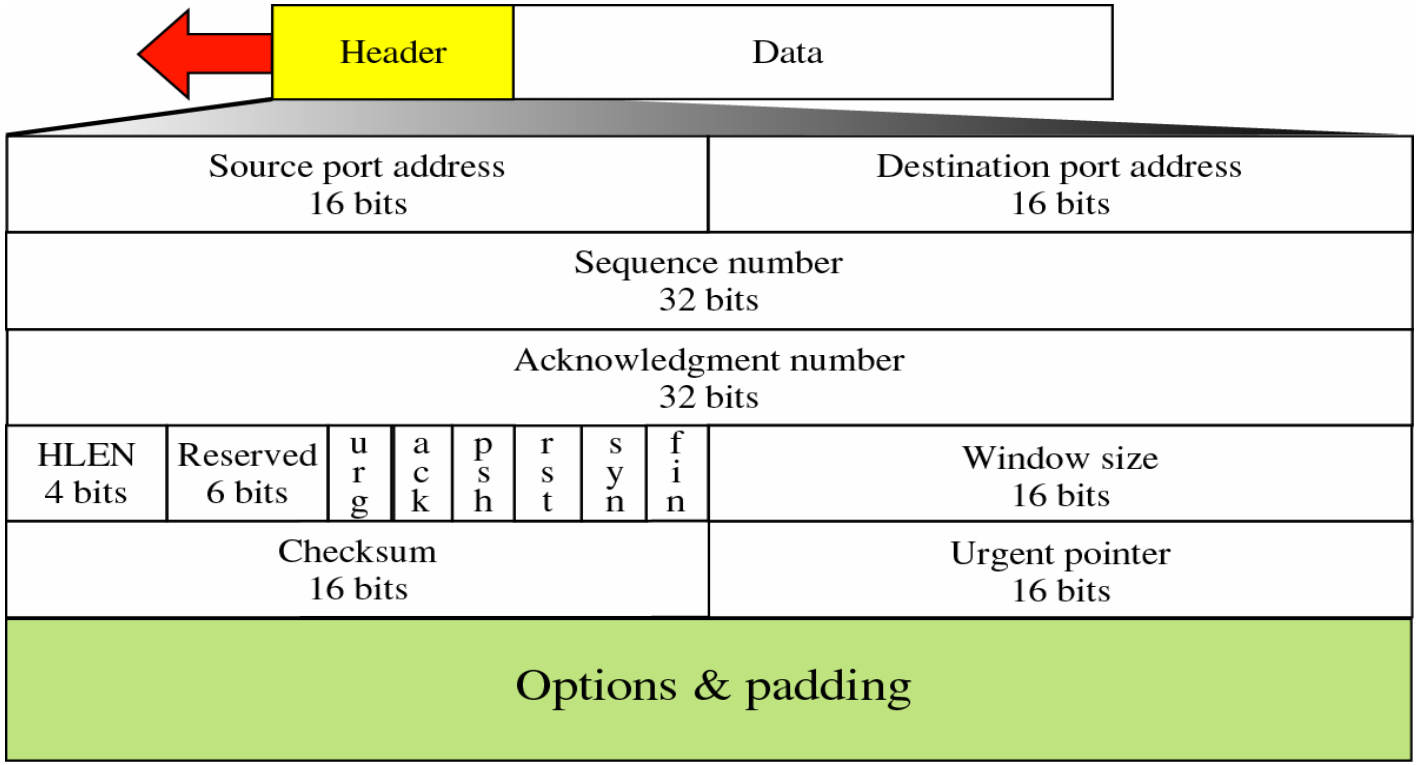
연결 설정 단계.

Three-way handshaking



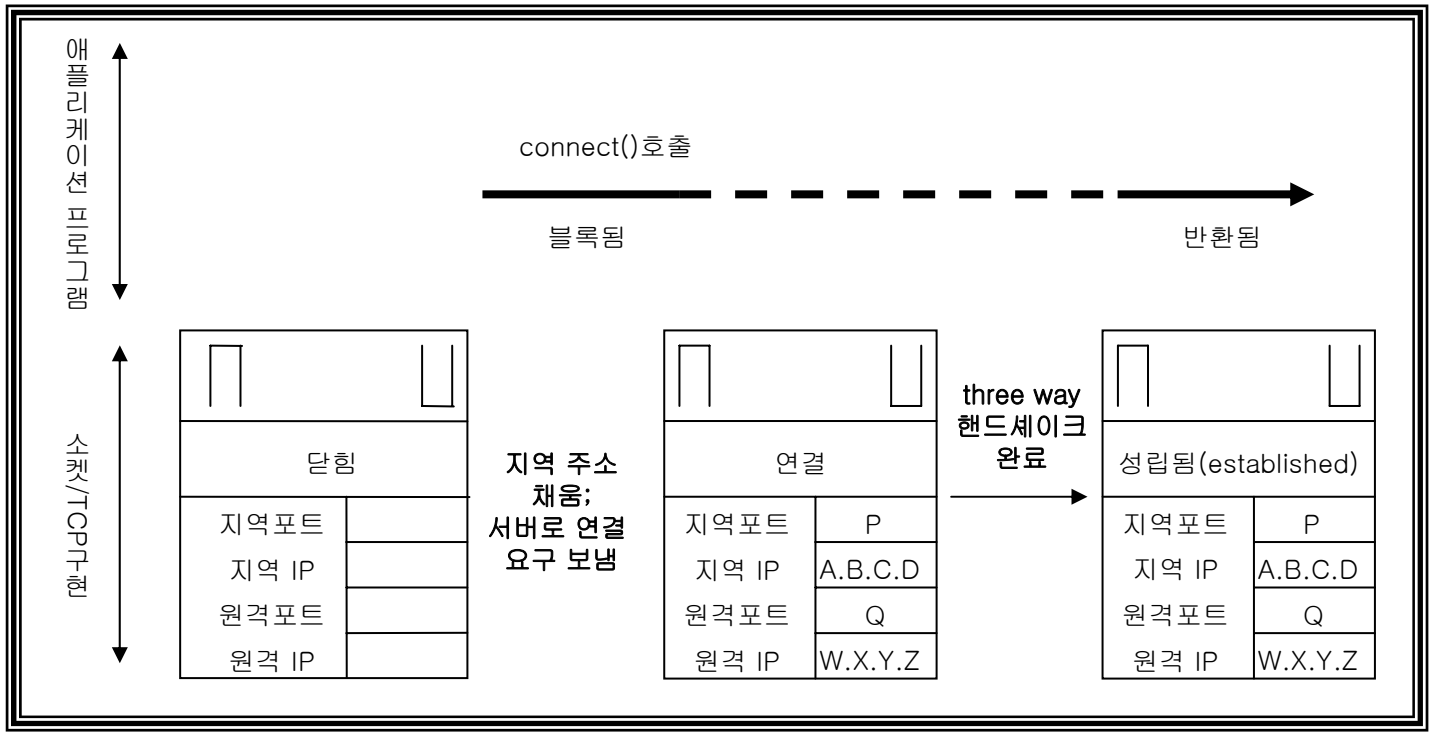
세그먼트

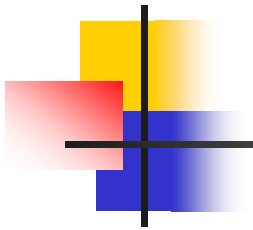
- TCP는 하나의 세그먼트를 이용하여 2개의 장치 사이에서 데이터를 전송
- 세그먼트의 형식



6.4.1 연결(connecting)

- 클라이언트측 연결 성립.
  - 클라이언트의 인터넷 주소는 A.B.C.D, 서버의 주소는 W.X.Y.Z이며, 서버의 포트는 Q이다.





- 클라이언트가 TCP소켓을 생성할 때 초기에는 닫힘 상태에 있다.

- 클라이언트가 W.X.Y.Z와 포트 Q로 connect()를 호출할 때, 시스템은 네 주소 필드들을 채운다.

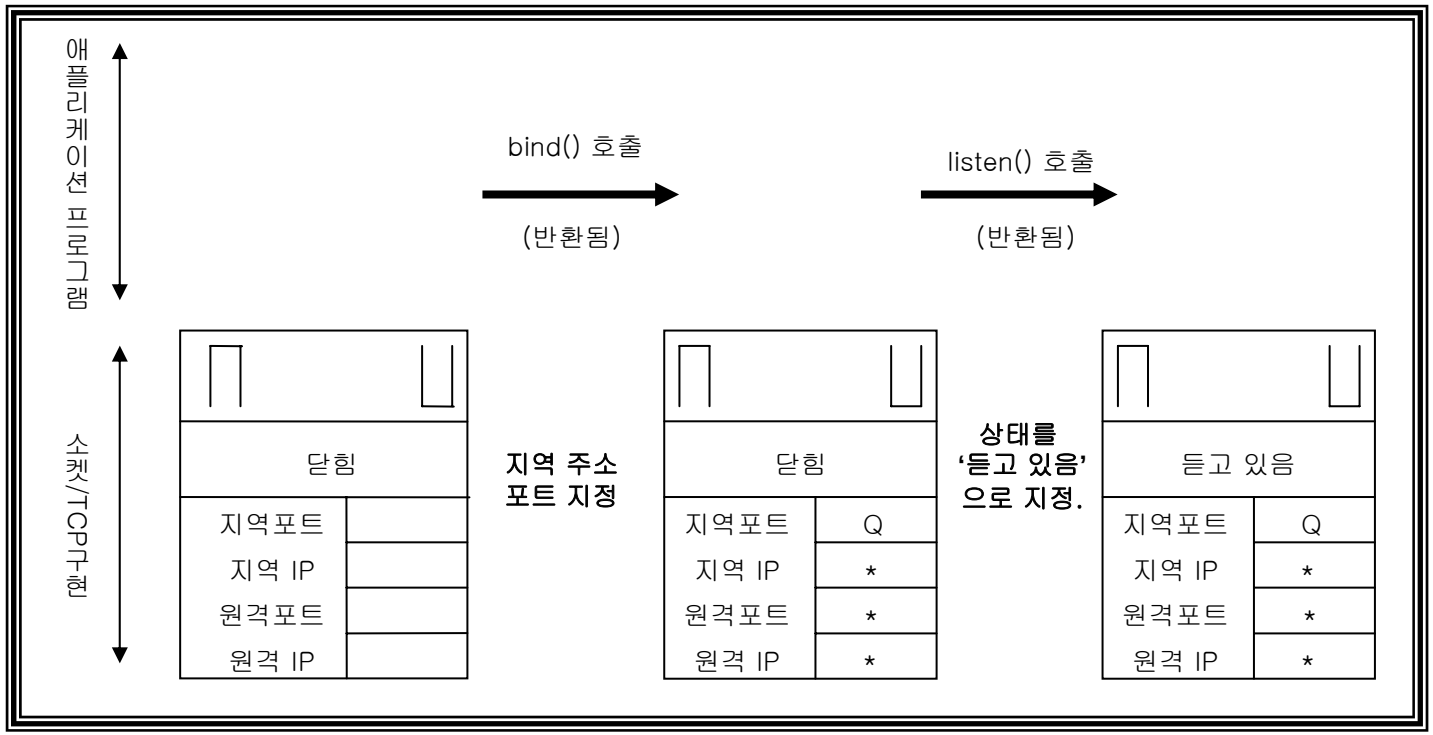
int connect(int socket, (struct sockaddr)*foreignAddress, int addressLength)

socket	소켓
foreignAddress	원격 소켓 주소를 나타낸다. 여기서는 W.X.Y.Z의 주소.
int addressLength	sockaddr 구조체의 바이트수. 보통 sizeof(foreignAddress)

- 클라이언트는 이전에 bind()를 호출하지 않았기 때문에 지역포트번호(P)가 다른 TCP 소켓이 이미 사용하고 있지 않는 번호로 시스템에 의해 선택되고 소켓에 부여.

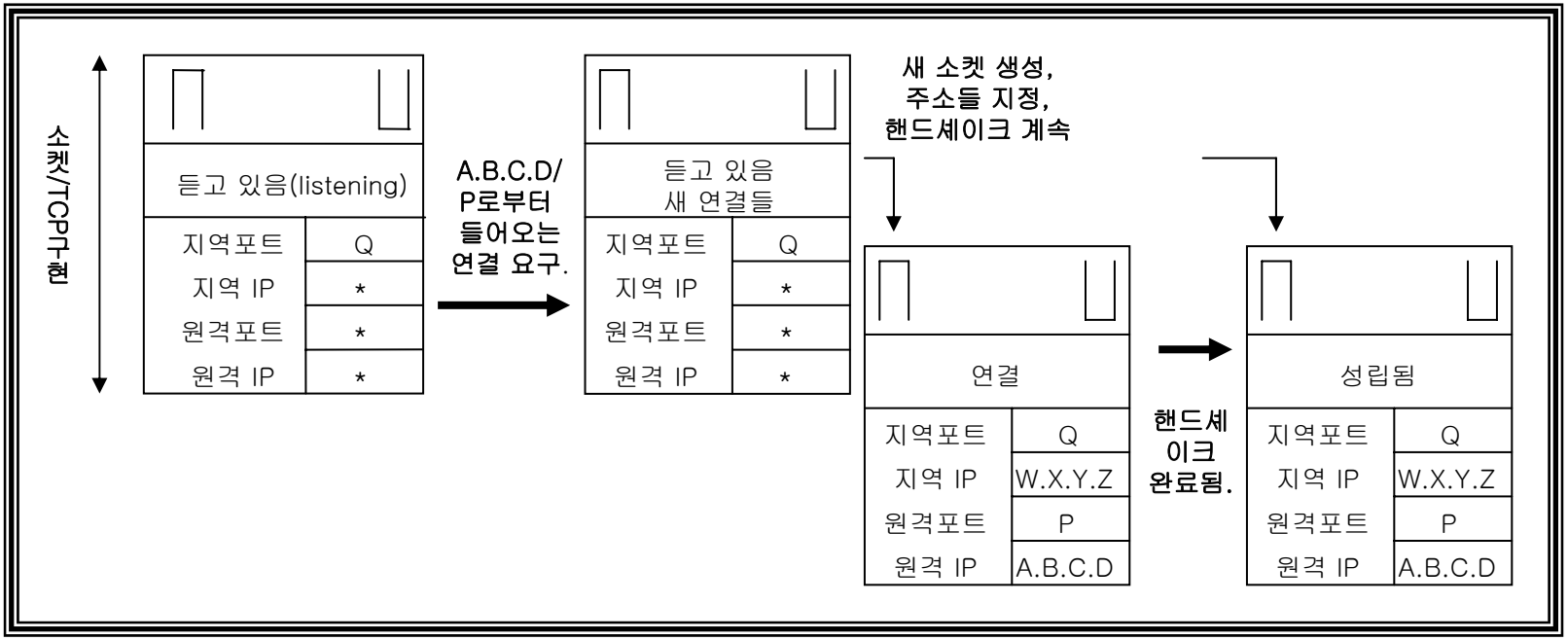
- 지역 인터넷 주소도 지정되는데, 사용되는 주소는 패킷이 서버로 보내질 때 거치는 네트워크 인터페이스 카드(NIC: Network Interface Card)의 주소이다.

▶ 서버측 소켓 설정.



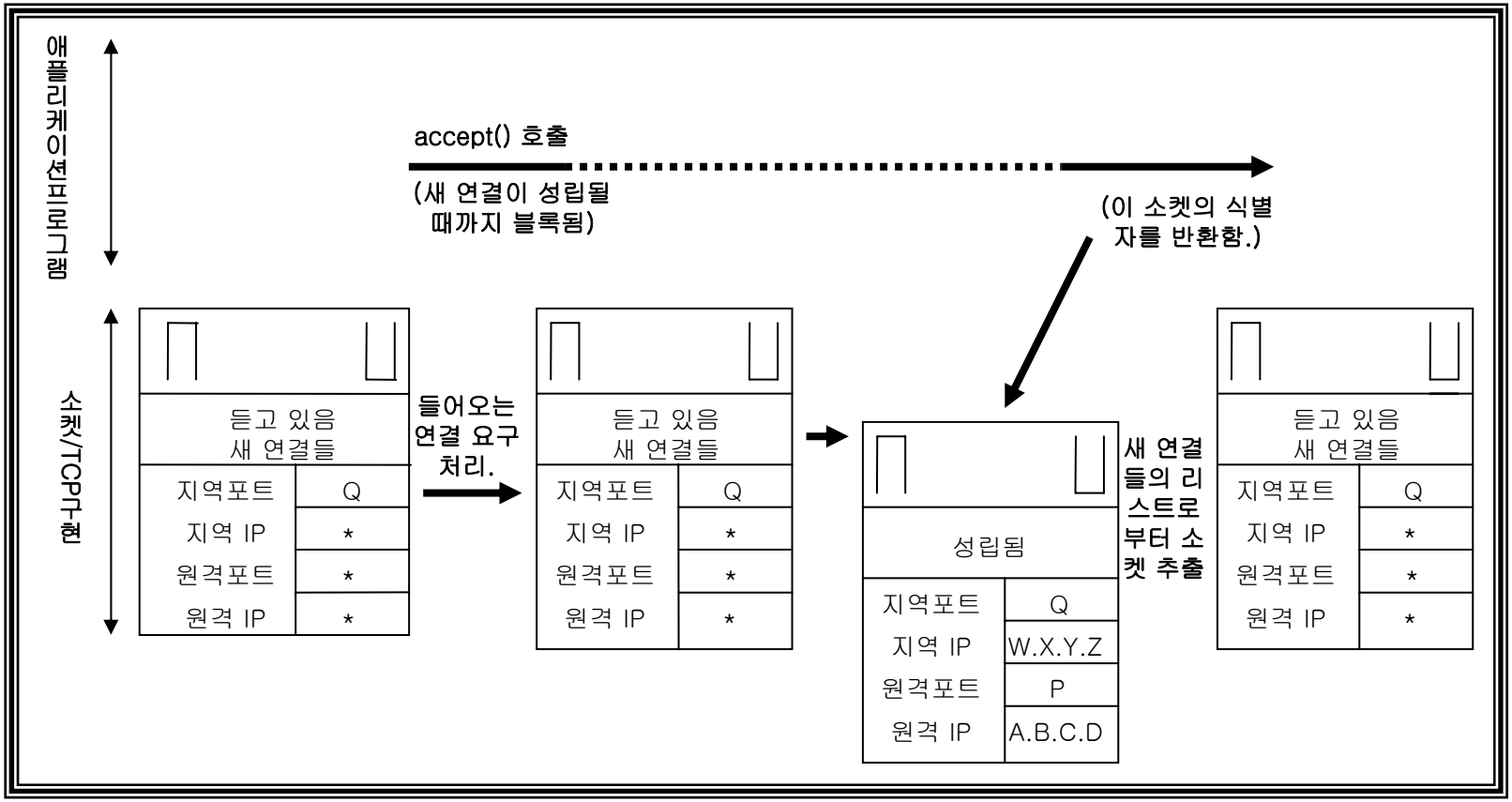
- 서버는 클라이언트에 알려진 특정 TCP포트에 결합된다.
- bind() 호출에서 포트번호만을 명시하고 지역 IP주소를 위해서 특수 와일드카드 주소 INADDR_ANY를 부여하는데 이것은 서버가 둘 이상의 IP주소를 가지는 경우에 소켓이 호스트의 어떠한 IP주소로 오는 연결도 수신할 수 있다.
- listen()을 호출할 때 소켓의 상태는 듣고 있음 상태로 바뀌어 새로운 연결을 받을 준비가 되었다는 것을 나타낸다.

▶ 들어오는 연결 요구 처리.



- 클라이언트로부터 연결 요구가 도착하면 그 연결을 위해 새로운 소켓 구조체가 생성된다.
- 새로운 소켓의 주소들은 도착하는 패킷에 따라 결정.
- 패킷의 목적지 인터넷 주소와 포트는 소켓의 지역 주소와 포트가 되고, 패킷의 발신지 주소와 포트는 소켓의 원격 주소와 포트가 된다.
- 새로운 소켓의 지역 포트 번호는 항상 듣고 있는 소켓의 포트와 동일.
- 새로운 소켓의 상태는 연결로 지정되고 원래의 서버 소켓과 관련 완전히 연결되지 않은 소켓들의 리스트에 추가.
- 원래의 서버 소켓은 그 상태를 변경하지 않는다.

▶ accept() 처리

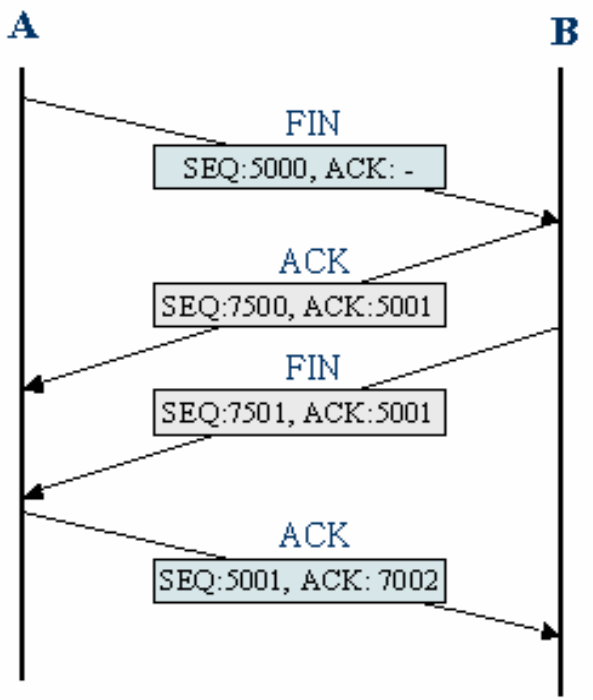


- 새로운 소켓 구조체를 생성하면서 TCP구현은 수신인정 TCP핸드셰이크 메시지를 클라이언트로 돌려보낸다.
- TCP서버는 세번째 메시지를 클라이언트로부터 받기 전까지는 핸드셰이크가 완료되었다고 생각하지 않는다.
- 세번째 메시지가 도착해야만 새로운 소켓의 상태는 성립됨으로 지정되고 accept()될 준비가 된다.

6.4.2 TCP 연결을 닫음

연결 종료 단계.

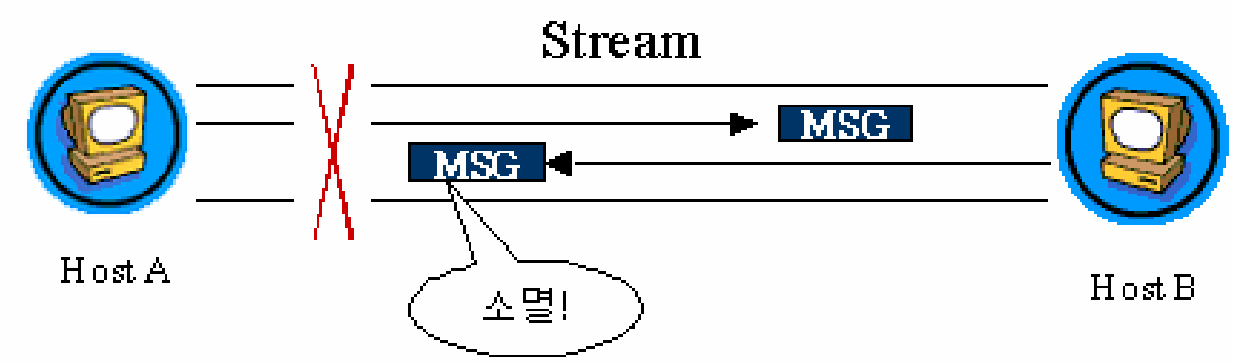
Four-way handshaking



소켓 연결 종료의 문제점

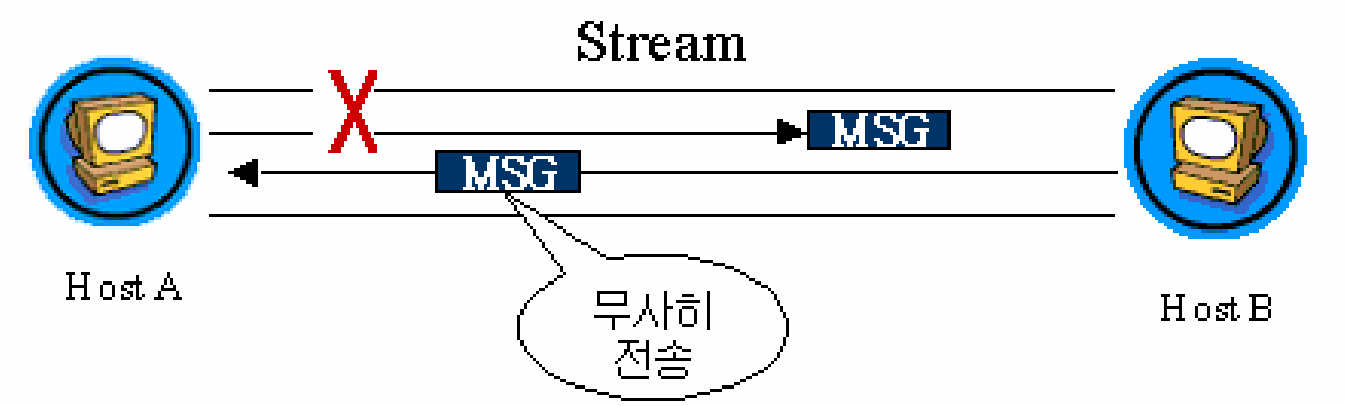
1. close() 함수의 호출 : 입력 출력 스트림 완전 종료.

2. 일방적인 방식의 완전종료는 경우에 따라서 문제가 될 수 있다.



Half-close

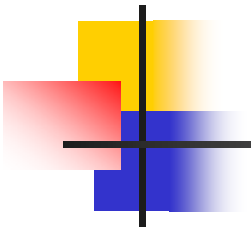
Half-close : 입력 및 출력 스트림 중 하나의 스트림만 종료하는 행위.



Half-close 기능의 함수 [리눅스]

```
#include <sys/socket.h>
int shutdown(int s, int how);
```

상수값	모드	정의
0	SHUT_RD	입력 스트림 종료
1	SHUT_WR	출력 스트림 종료
2	SHUT_RDWR	입 출력 스트림 종료



▶ 우아한 종료(graceful close, 깨끗한 닫기)

1. 애플리케이션은 close()를 호출하거나 0보다 큰 두 번째 인자를 가지고 shutdown()을 호출

→ 소켓으로 데이터를 보내는 일을 중지한다

→ 하부 TCP는 SendQ에 남아 있는 데이터가 있으면 전송하고, 상대편으로 클로징 TCP 핸드셰이크 메시지를 보낸다.

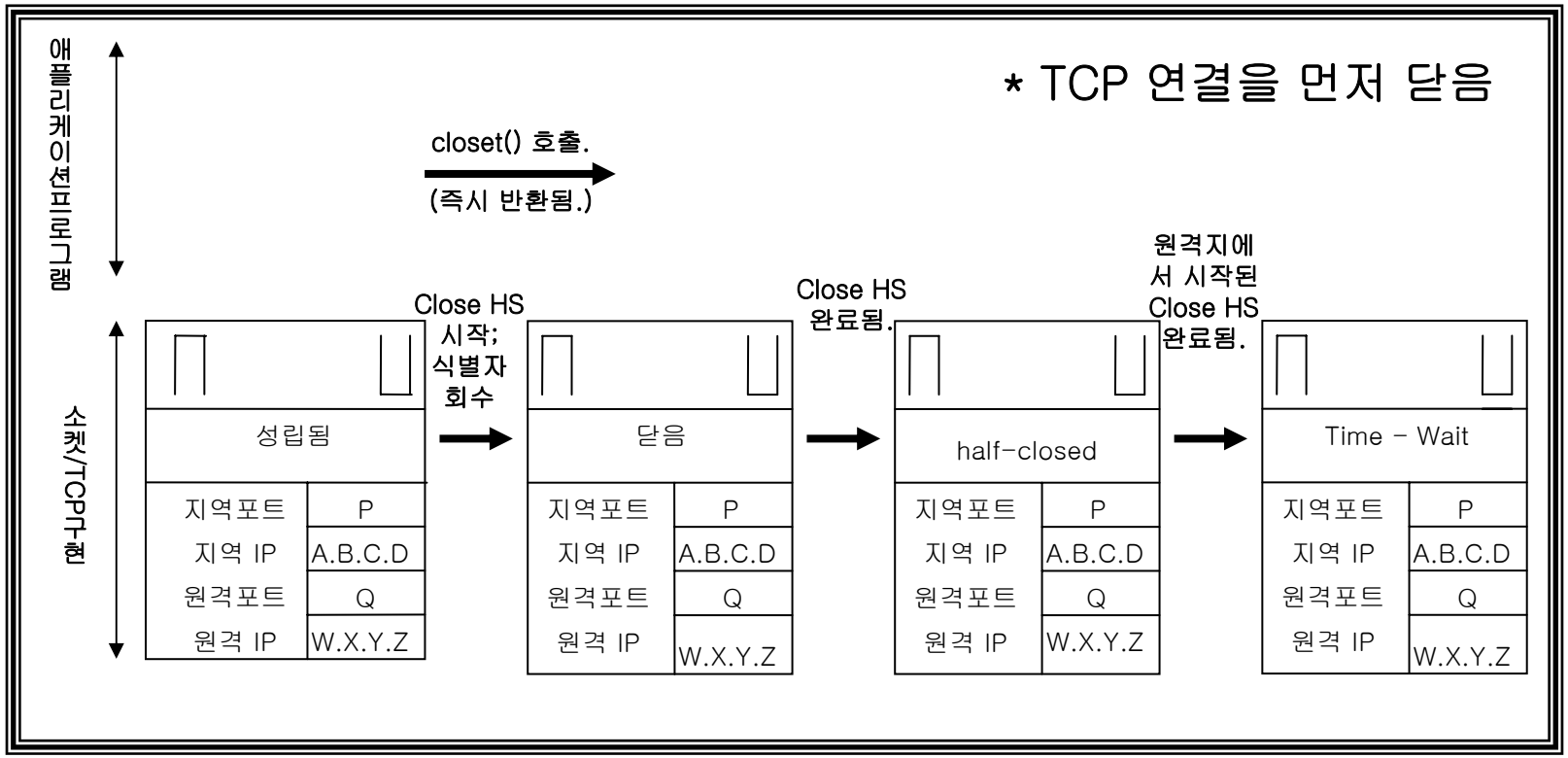
2. 클로징 TCP는 자신의 클로징 핸드셰이크 메시지에 대한 수신인정을 기다린다.

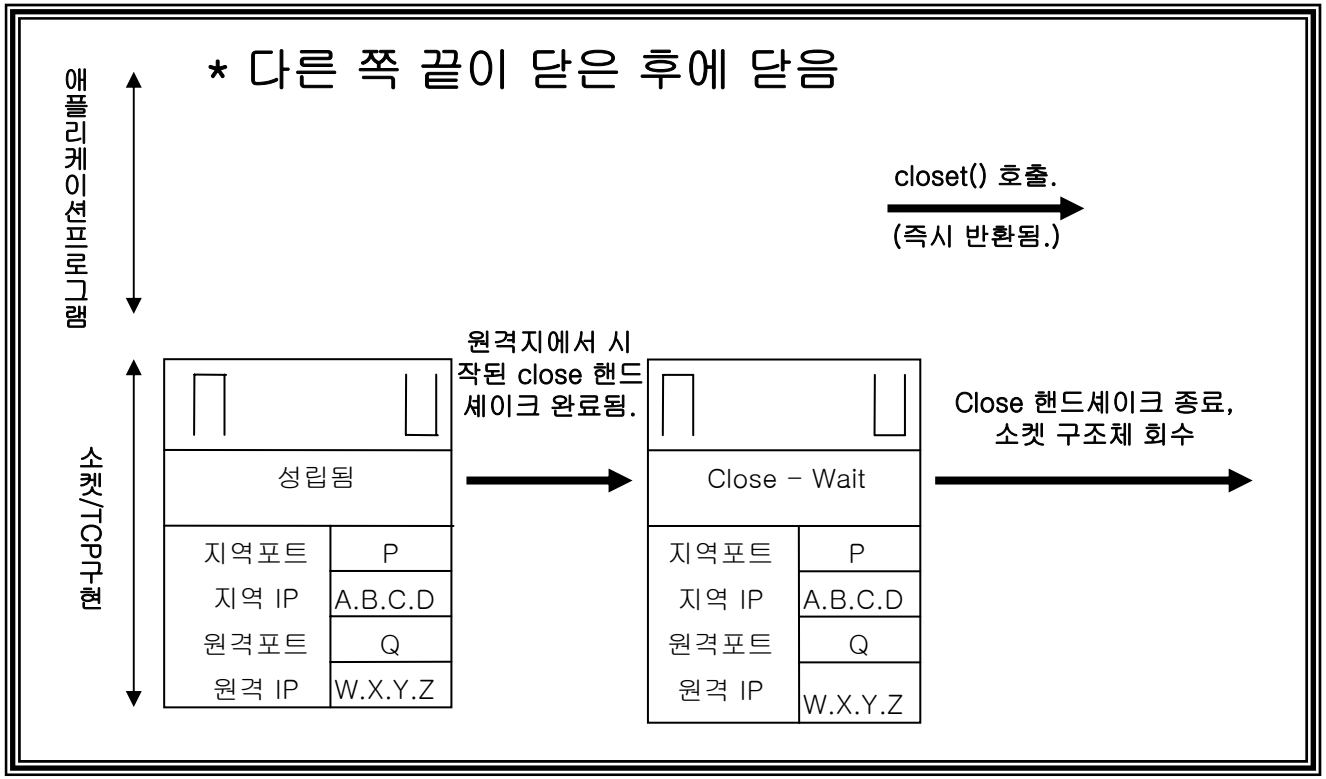
→이 시점에서의 해당 연결은 half-closed 상태.

3. 양 종단이 모두 더 이상 보낼 데이터가 없다고 알릴 때까지 완전히 닫히지 않는다.

▶ TCP의 클로징 사건 순서

- 한 애플리케이션이 close()를 호출하고 상대편이 close()를 호출하기 전에 자신의 클로징 핸드셰이크를 완료하거나, 또는 양쪽이 close()를 거의 동시에 호출하여 각각의 클로징 핸드셰이크 메시지가 네트워크를 따라 운반되는것이다.





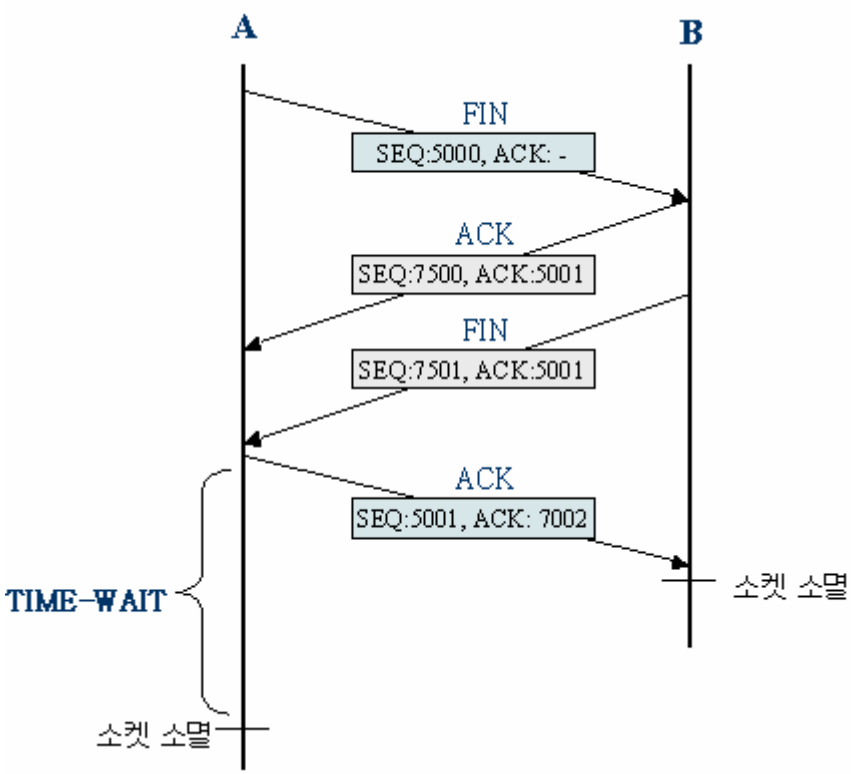
- 먼저 닫지 않는 종단에서는 클로징 핸드셰이크 메시지가 도착했을 때 acknowledgement가 즉시 보내지고 연결 상태는 Close-Wait이 된다.

- 종단에서는 애플리케이션이 close()를 호출하기를 기다린다. close()를 호출되었을 때 그 소켓의 식별자는 회수되고 최종 클로징 핸드셰이크가 시작.

- 이것이 종료되면 전체 소켓 구조체가 회수.

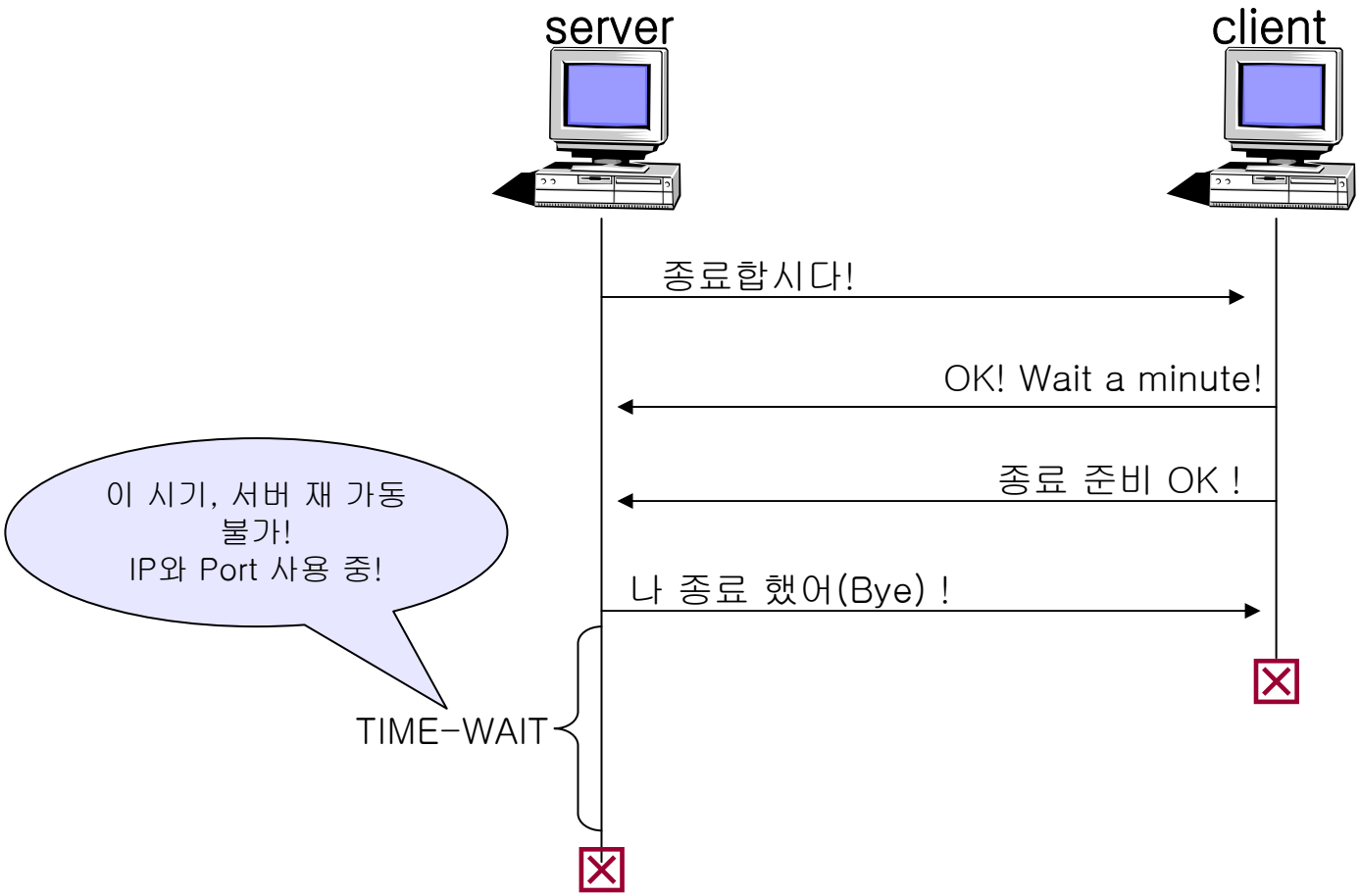
TIME-WAIT 상태란 ?

1. 연결 종료 시 마지막 패킷 전송 실패를 대비하기 위한 상태.



서버의 연결 종료

1. 서버에서 먼저 종료 요청 후, 다시 서버를 가동 시키면?



TIME-WAIT 타이머의 재시작

1. TIME-WAIT 상태는 우리의 생각보다 더 길어 질 수 있다.