Socket Programming

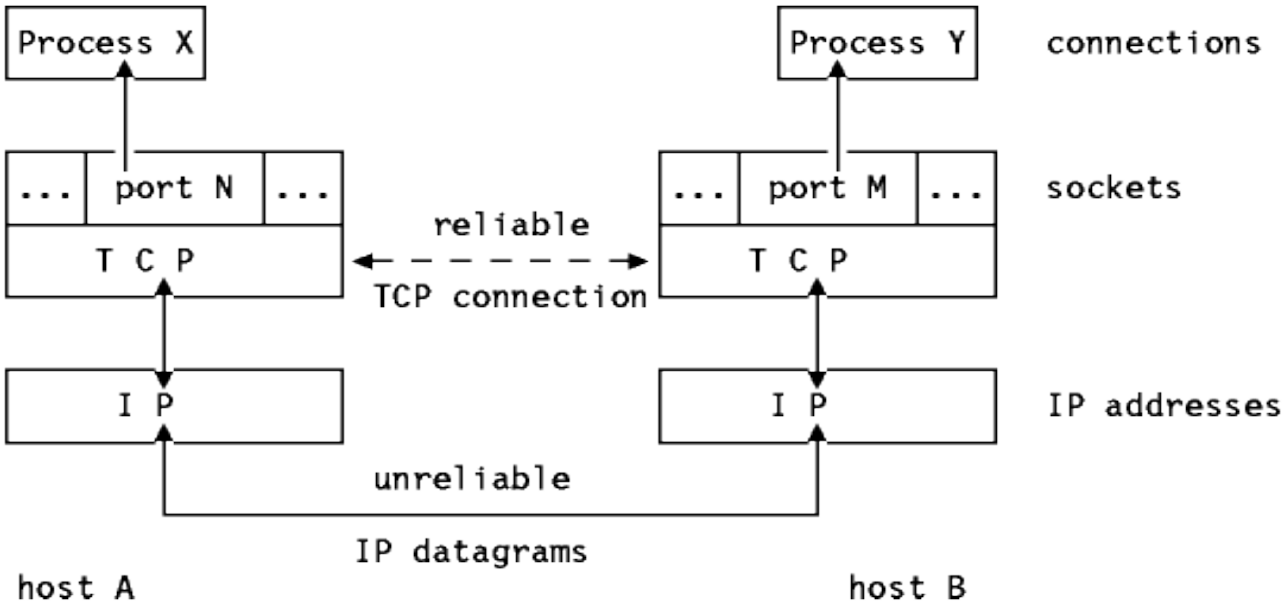
유명성

1. Socket

1.1 Socket

Socket

- ❖ Socket은 통신의 논리적인 접속점이다.
- ❖ Unix의 File API를 확장한 것으로 다른 Host의 Process와 통신할 수 있도록 해준다.
- ❖ 즉 TCP 3계층과 4계층 사이에서 종단간 통신을 위한 Interface 역할을 수행한다.
- ❖ 소켓은 Internet Socket(다른 host와 통신)과 Unix Domain Socket(IPC용)이 있다.
- ❖ 서버/클라이언트 구조이다.



Socket Module

- ❖ Socket Module은 Python에서 BSD Socket API에 대한 인터페이스를 제공한다.
- ❖ Socket Module이 제공하는 API는 C의 Socket API에 직접 매핑된다.
- ❖ socketserver 프레임워크를 사용하면 더욱 쉽게 Socket 응용프로그램을 구현할 수 있다.

```
1.  import socket
2.
3.  serv = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
4.
5.  serv.bind(('0.0.0.0', 8080))
6.  serv.listen(5)
7.
```

5 Tuple

① 통신에 사용할 프로토콜

② 자신의 IP 주소

③ 자신의 Port 번호

④ 상대방의 IP 주소

⑤ 상대방의 Port 번호

*IP 주소는 Host를 식별하고, Port 번호는 Host 내 Process를 식별한다.
MAC 주소는 Host내 NIC를 식별한다.

1. Socket

1.3 Python Socket API

-socket : 통신에 필요한 socket을 생성 후 SD를 리턴 받는다.

-bind : 통신에 사용할 socket의 SD와 자신의 IP, Port 번호를 매핑한다.

-listen : 소켓에서 클라이언트의 요청을 기다린다.(Blocking -> 클라이언트의 요청이 오면 Release)

-accept : 클라이언트가 connect를 수행이 끝날 때까지 대기 후 연결요청 수락.(Blocking -> connect 후 Release)

-connect : 서버의 IP, Port 번호를 통해 클라이언트가 서버에 접속을 요청한다.

-recv : 원격지 host에서 전달된 데이터를 socket을 통해 읽는다.(Blocking -> 데이터를 커널 버퍼에 복사 후 Release)

-send : 원격지 host에 socket을 통해 데이터를 보낸다.(Blocking -> 데이터를 커널 버퍼에 복사한 뒤 Release)

1. Socket

1.3 Python Socket API

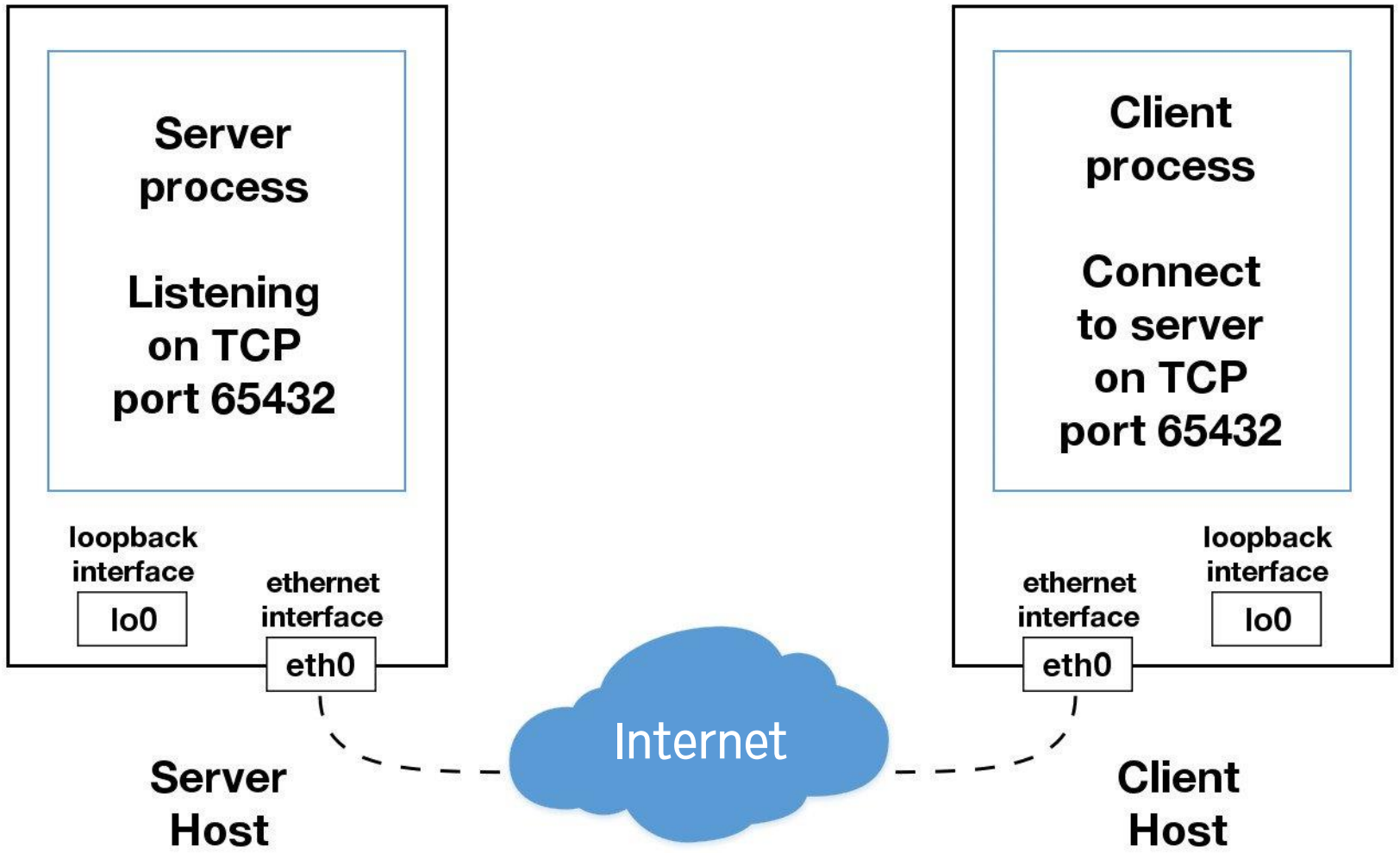
-recvfrom : 비연결 socket에서 원격지 host의 데이터를 읽어온다. (Blocking -> 데이터를 커널 버퍼에 복사 후 Release)

-sendto : 데이터를 비연결 socket을 통해 인자로 주어진 IP와 Port로 보낸다.
(Blocking -> 데이터를 커널 버퍼에 복사 후 Release)

-close : 생성된 socket을 반납한다.

1. Socket

1.3 Python Socket API



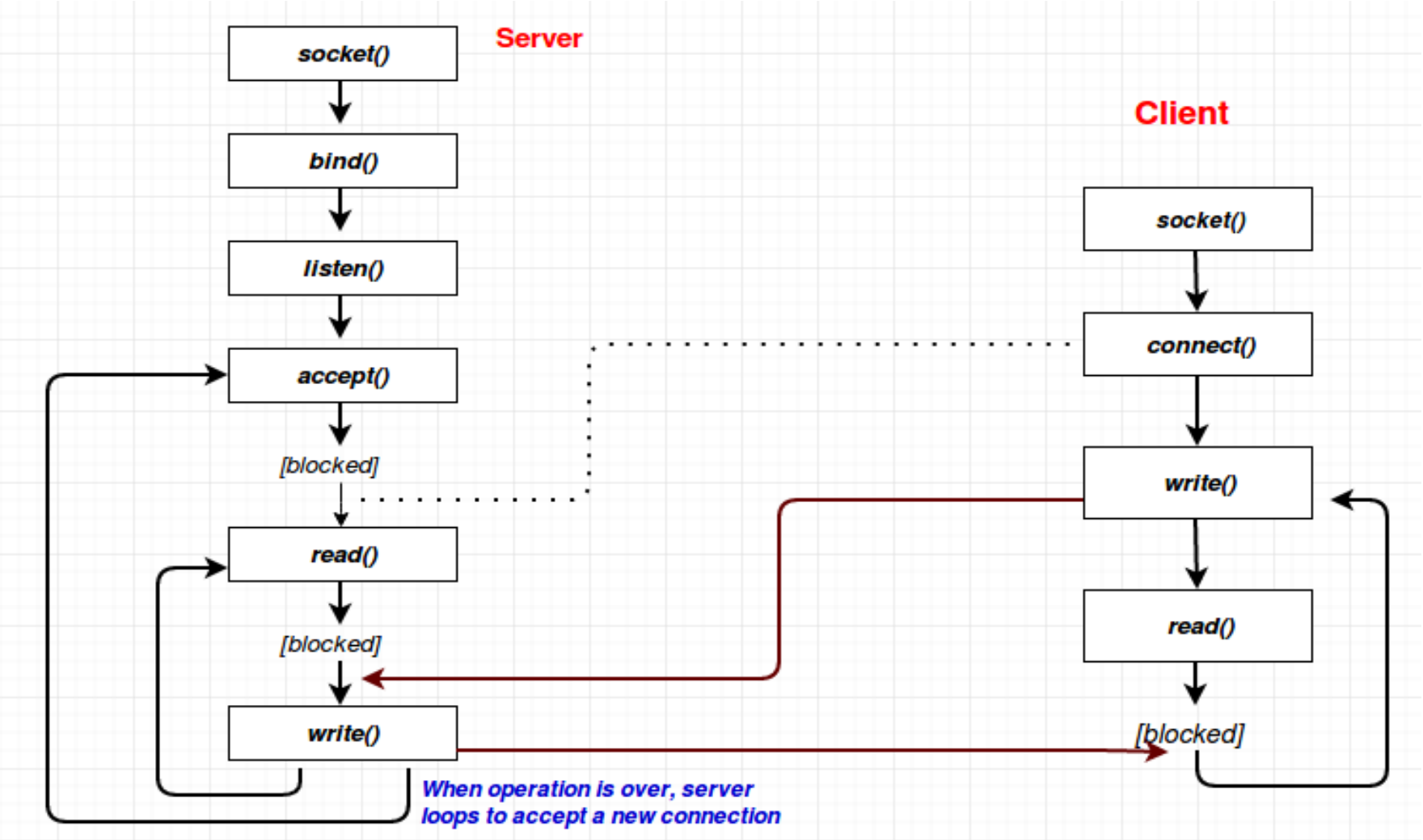
TCP(STREAM) Socket

- ❖ 연결지향형 소켓으로 연결 수립 -> 데이터 송/수신 -> 연결 종료 순서로 동작한다.
- ❖ 종단 Host간 신뢰성 있는 데이터 전송을 지원한다. 모든 데이터는 순서에 맞게 에러 없이 전달된다.
- ❖ 송신측이 보내는 데이터는 수신측에서 하나의 스트림으로 처리된다.
- ❖ socket() 호출 시 socket.SOCK_STREAM을 사용한다.
- ❖ 점대점 연결이기 때문에 Unicast에 사용된다.

```
#!/usr/bin/python

#This is tcp_server.py script

import socket                           #line 1: Import socket module

s = socket.socket()                     #line 2: create a socket object
host = socket.gethostname()             #line 3: Get current machine name
port = 9999                             #line 4: Get port number for connection

s.bind((host,port))                     #line 5: bind with the address

print "Waiting for connection..."
s.listen(5)                             #line 6: listen for connections

while True:
    conn,addr = s.accept()              #line 7: connect and accept from client
    print 'Got Connection from', addr
    conn.send('Server Saying Hi')
    conn.close()                        #line 8: Close the connection
```

1. Socket

1.4 TCP Socket



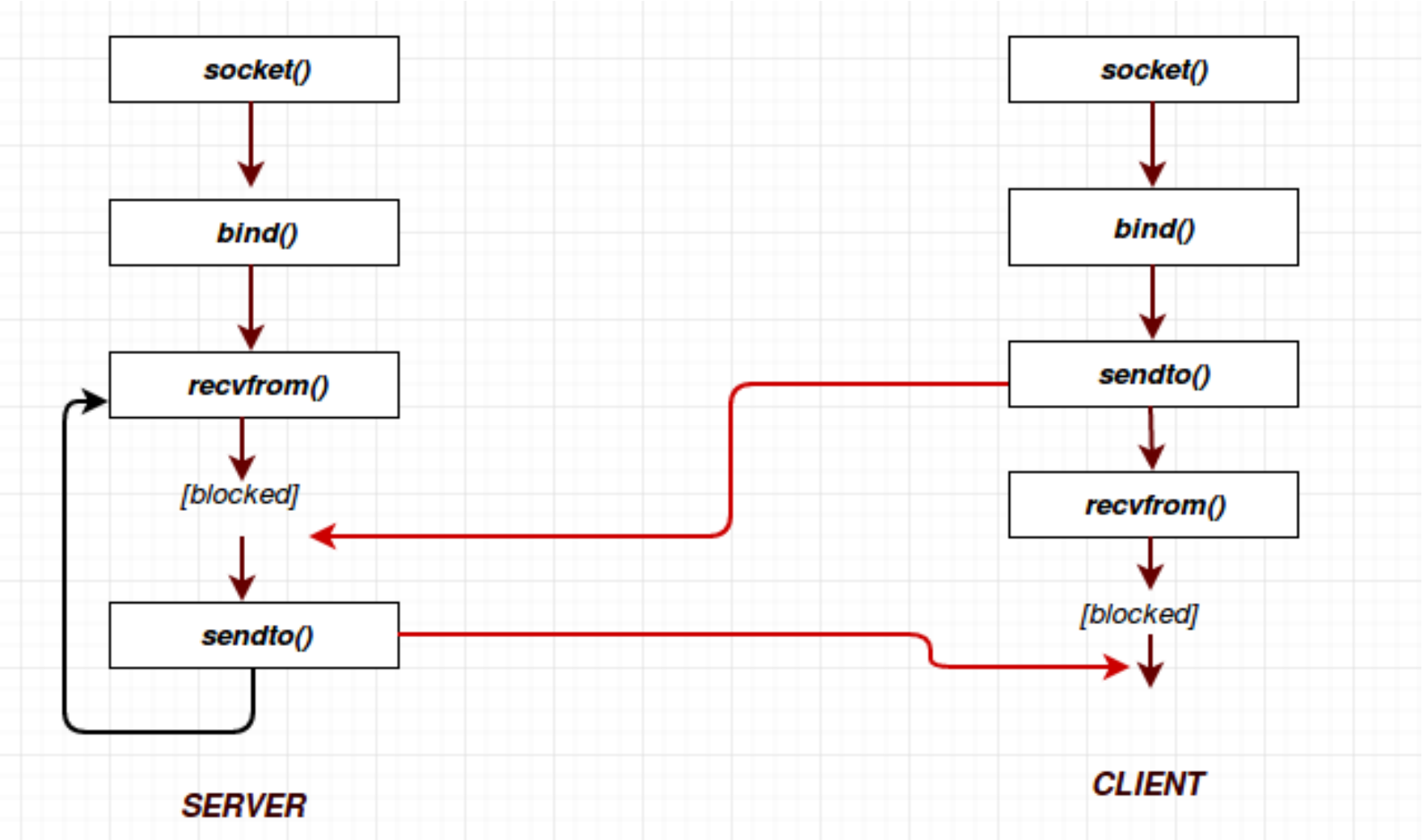
UDP(DATAGRAM) Socket

- ❖ 비연결형 소켓으로 신뢰성 있는 데이터 전송을 보장하지 않는다.
- ❖ 연결 설정 및 해제 절차가 없기 때문에 전송속도가 빠르다.
- ❖ socket.SOCK_DGRAM 사용, 서버에서 listen(), accept()를 클라이언트에서 connect()를 호출하지 않는다.
- ❖ 송신측이 보낸 메시지는 수신측에서 개별 메시지로 처리된다.
- ❖ 하나의 소켓에서 다수의 목적지로 데이터를 보낼 수 있다. -> broadcast, multicast에 이용

```
#!usr/bin/python

import socket

sock = socket.socket(socket.AF_INET,socket.SOCK_DGRAM)      # For UDP

udp_host = socket.gethostname()             # Host IP
udp_port = 12345                            # specified port to connect

#print type(sock) =============> 'type' can be used to see type
                # of any variable ('sock' here)

sock.bind((udp_host,udp_port))

while True:
    print "Waiting for client..."
    data,addr = sock.recvfrom(1024)         #receive data from client
    print "Received Messages:",data," from",addr
```

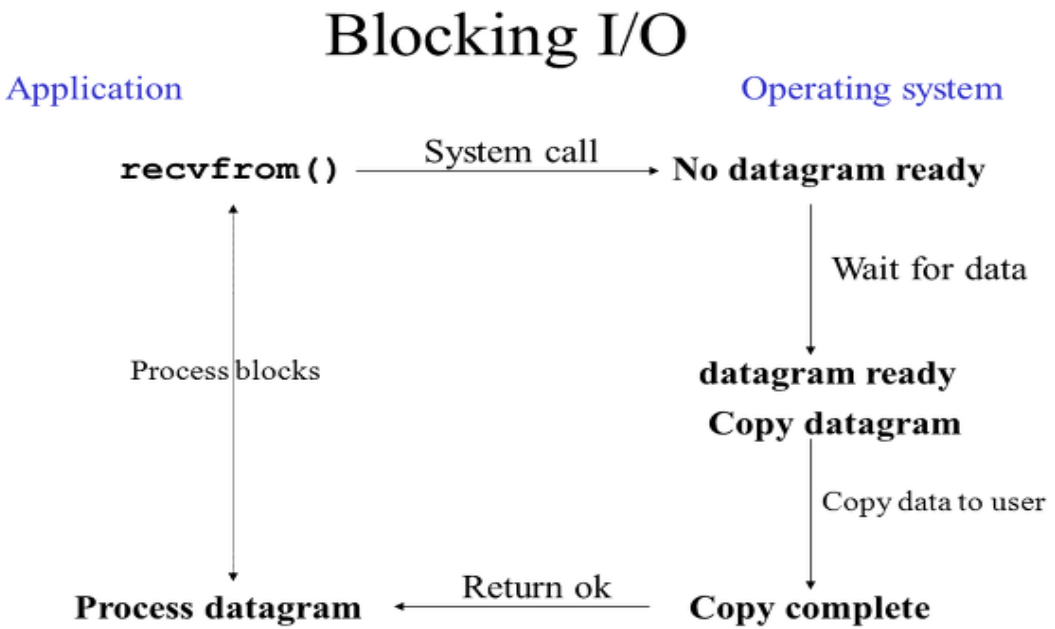
1. Socket

1.5 UDP Socket



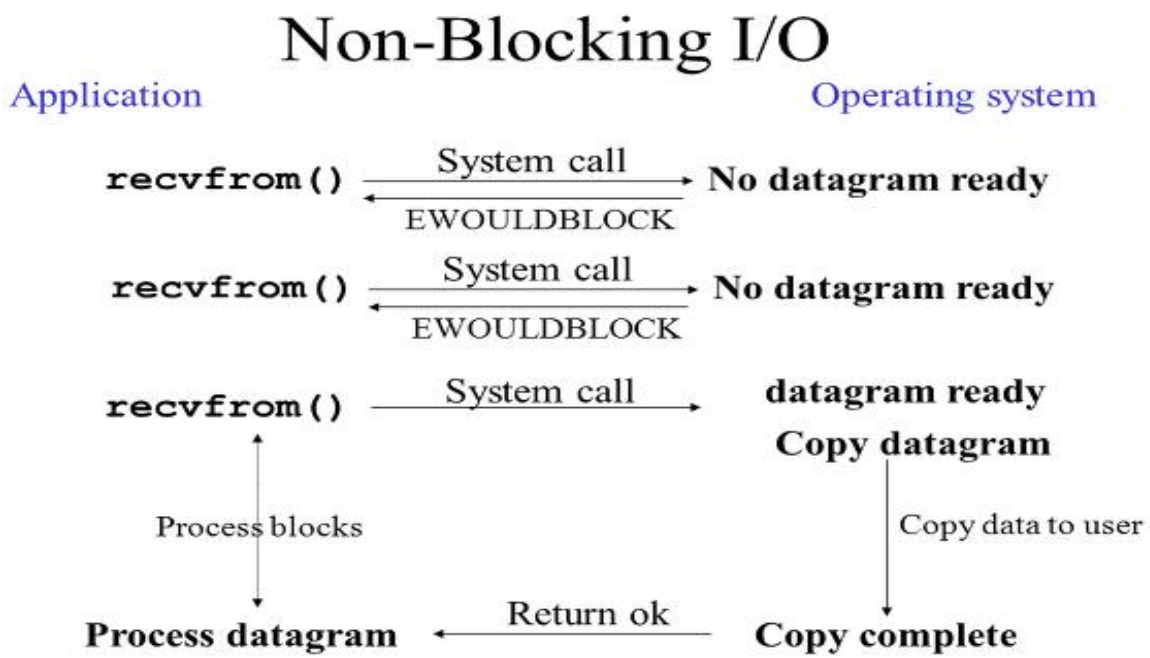
Blocking Socket

- ❖ Socket API 호출 시 조건이 만족되지 않으면 API를 호출한 스레드가 Block 된다.(run -> wait)
- ❖ Socket API는 기본적으로 Blocking mode이다.
- ❖ Blocking된 Socket API가 리턴될 때까지 다른 작업을 할 수 없다. -> Multi thread를 이용
- ❖ 1대1 통신이거나, 한가지 작업만 할 때 사용



Non-Blocking Socket

- ❖ Socket API 호출 후 조건에 관계없이 바로 return하는 모드.
- ❖ 일반적으로 어떤 시스템 콜이 완료되었는지 보기 위해 루프를 돌며 확인하는 Polling을 사용해야 한다.
- ❖ 통신 대상이 여럿이거나, 다른 작업을 병행해야 하는 경우 Non-Blocking 또는 Async 모드를 사용해야 한다.



2. Server Model

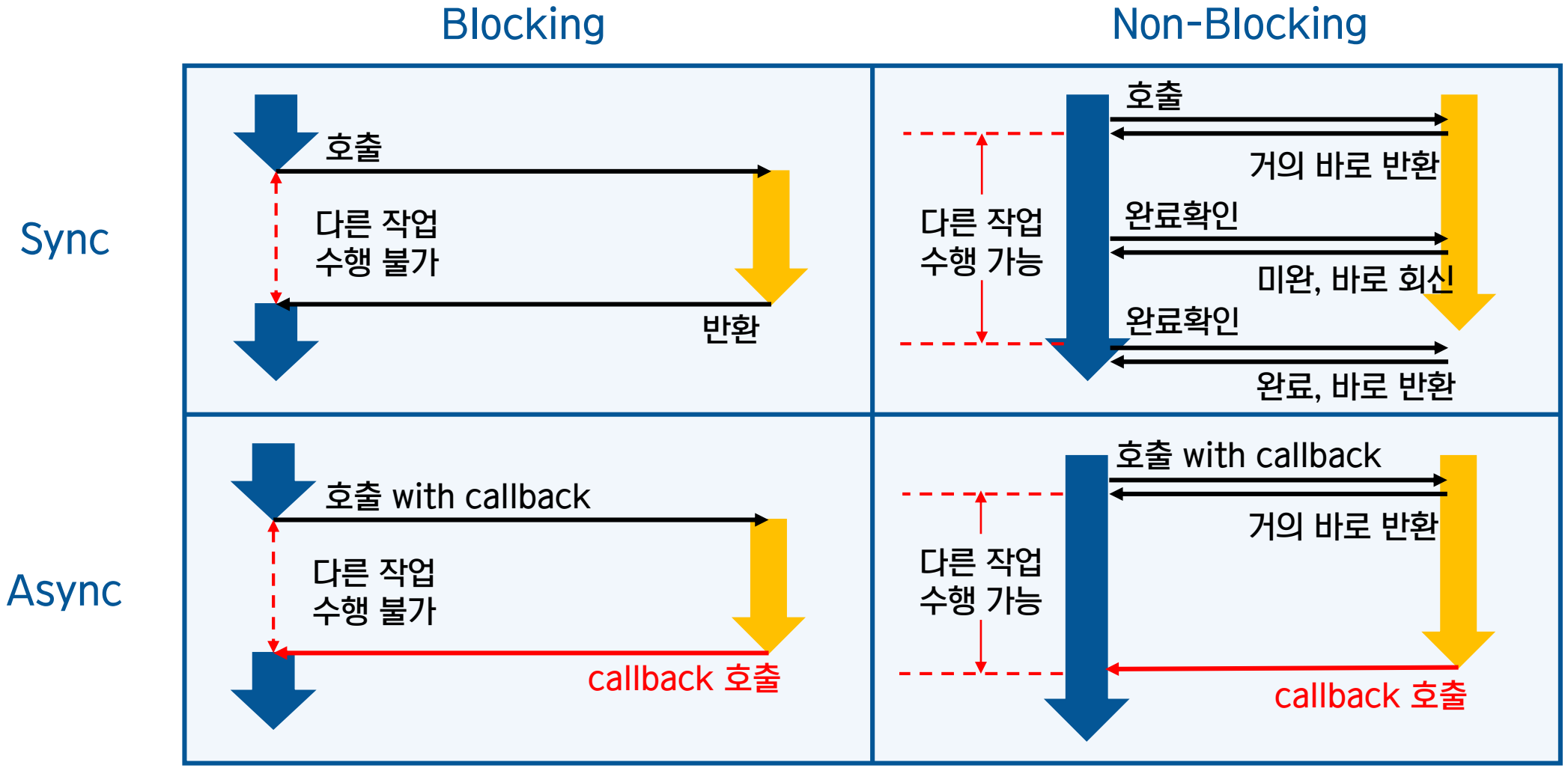
2.1 Socket I/O Mode

Blocking / Non-Blocking : 호출되는 함수가 바로 return하는지에 따라 구분.

Sync / Async : 호출되는 함수의 작업 완료를 누가 처리하는지에 따라 구분.

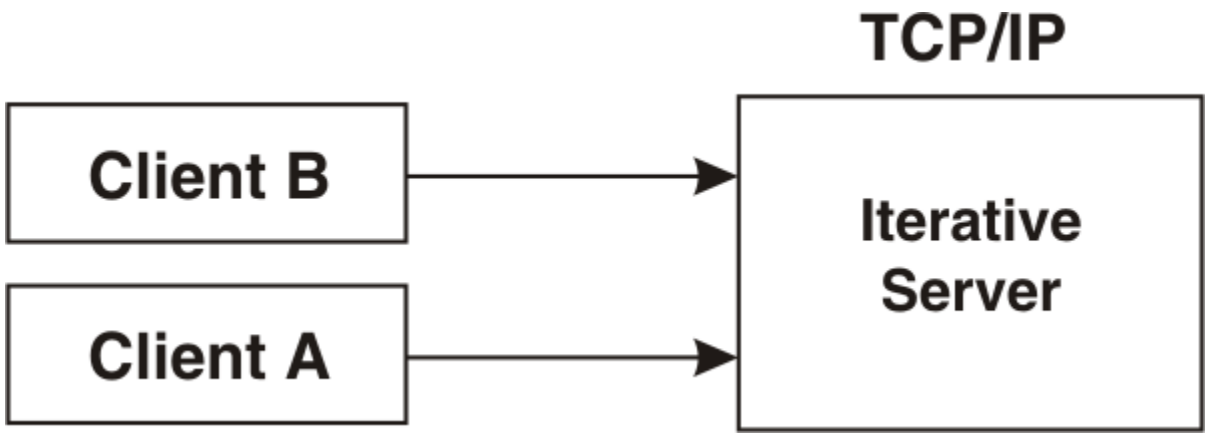
2. Server Model

2.1 Socket I/O Mode



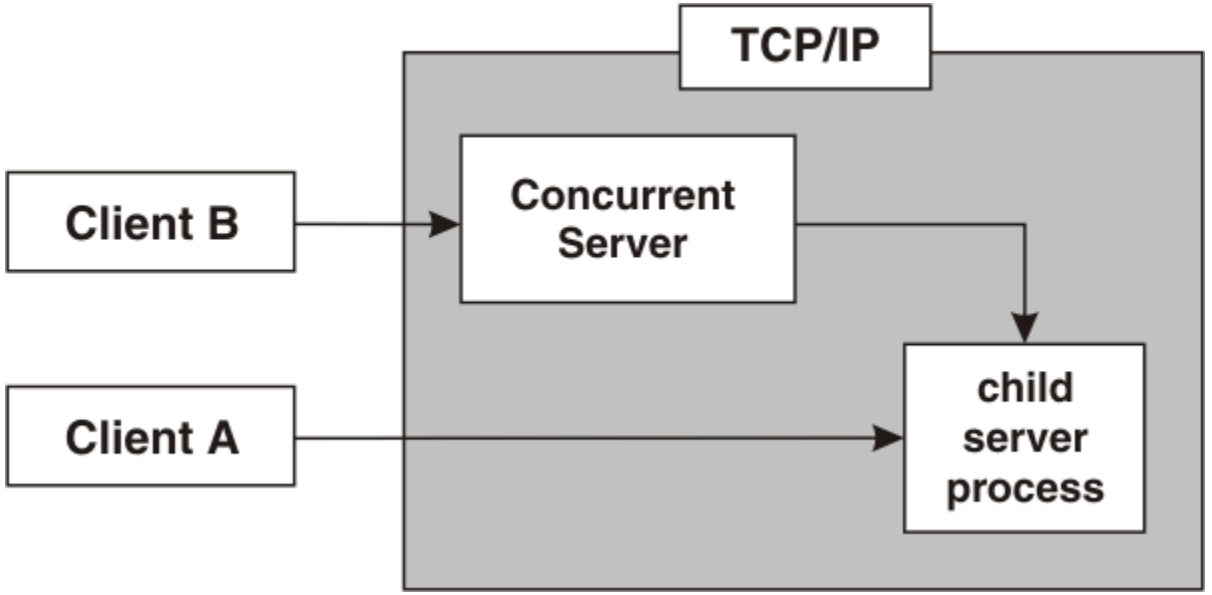
반복서버

- ❖ 접속한 클라이언트를 하나씩 차례대로 처리한다.
- ❖ 하나의 스레드로 모든 요청을 처리하기 때문에 시스템 자원 소모가 적다.
- ❖ 서버와 클라이언트의 통신이 길어지면 그 시간만큼 다른 클라이언트의 요청을 처리할 수 없다.
- ❖ 주로 이벤트 기반 비동기처리를 통해 구현한다. ex) node.js



병행서버

- ❖ 다수의 스레드 혹은 프로세스를 이용해 많은 클라이언트의 요청을 병렬로 처리
- ❖ 서버와 클라이언트간 통신이 길어져도 다른 클라이언트 요청을 즉각 처리할 수 있다.
- ❖ 멀티 스레드 혹은 멀티 프로세스를 이용해 구현하기 때문에 시스템 자원 소모가 크다.
- ❖ 대표적인 예로 Apache 서버가 있으며 pre-fork 방식을 사용한다.



2. Server Model

2.3 Concurrent Server

Thread

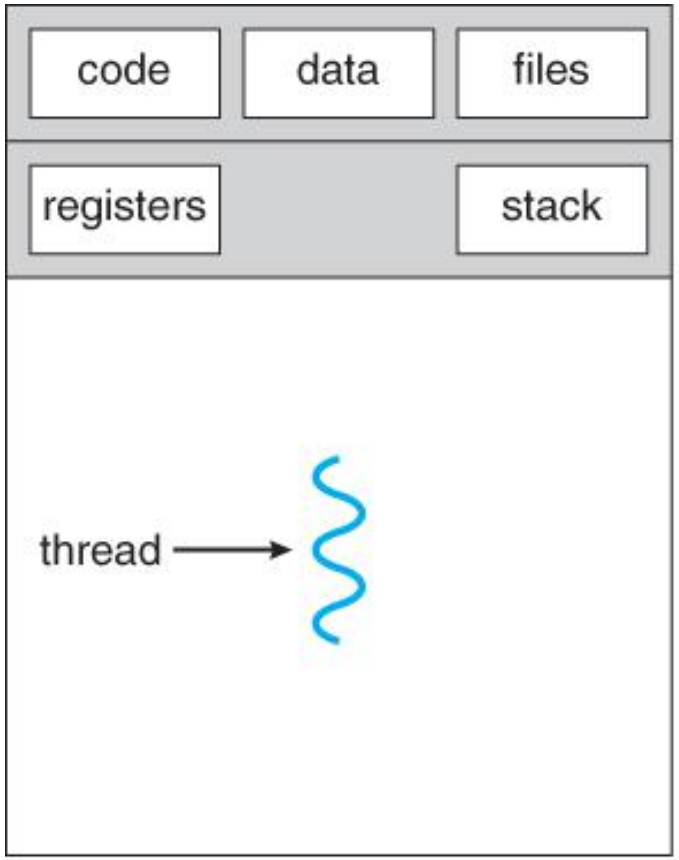
- ❖ 스레드는 경량 프로세스라고도 불리며, 프로세스 내에 존재하는 하나의 독립적인 실행 흐름이다.
- ❖ 한 프로세스 내에서 다른 스레드와 Code, Heap, Data 영역을 공유하며 Stack 영역만 독립적으로 가진다.
- ❖ 스레드는 OS 입장에서 실행의 단위이다.

Process

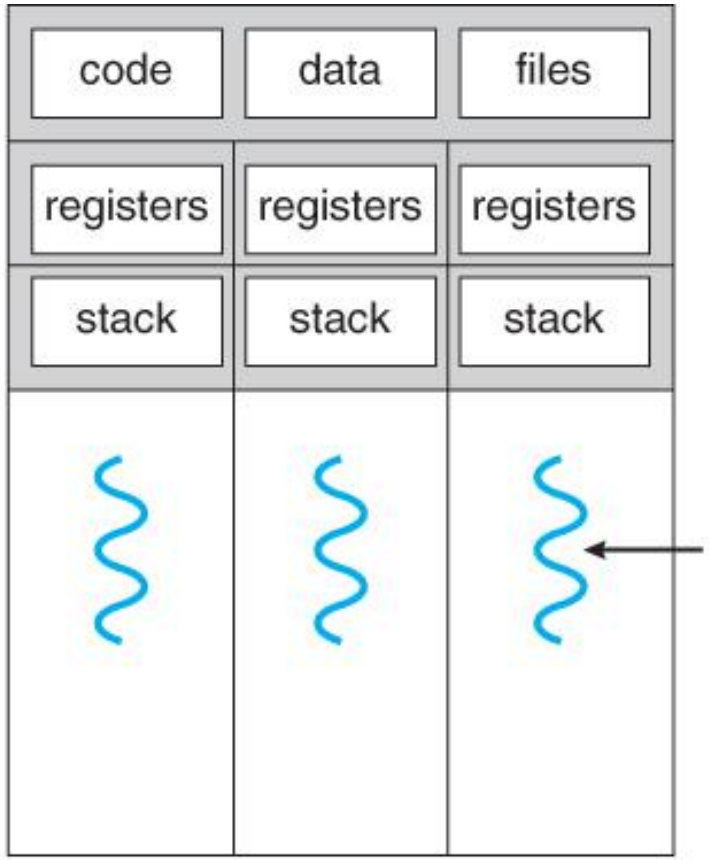
- ❖ 프로세스는 메모리에 올라가 실행중인 프로그램을 뜻한다.
- ❖ 각 프로세스의 메모리 영역은 서로 독립적이며, 가상 메모리에 매핑되기 때문에 서로 알 수 없다.
- ❖ 프로세스는 내부에 다수의 스레드를 가질 수 있다.
- ❖ 프로세스는 OS 입장에서 자원을 할당하는 단위이다.

2. Server Model

2.3 Concurrent Server



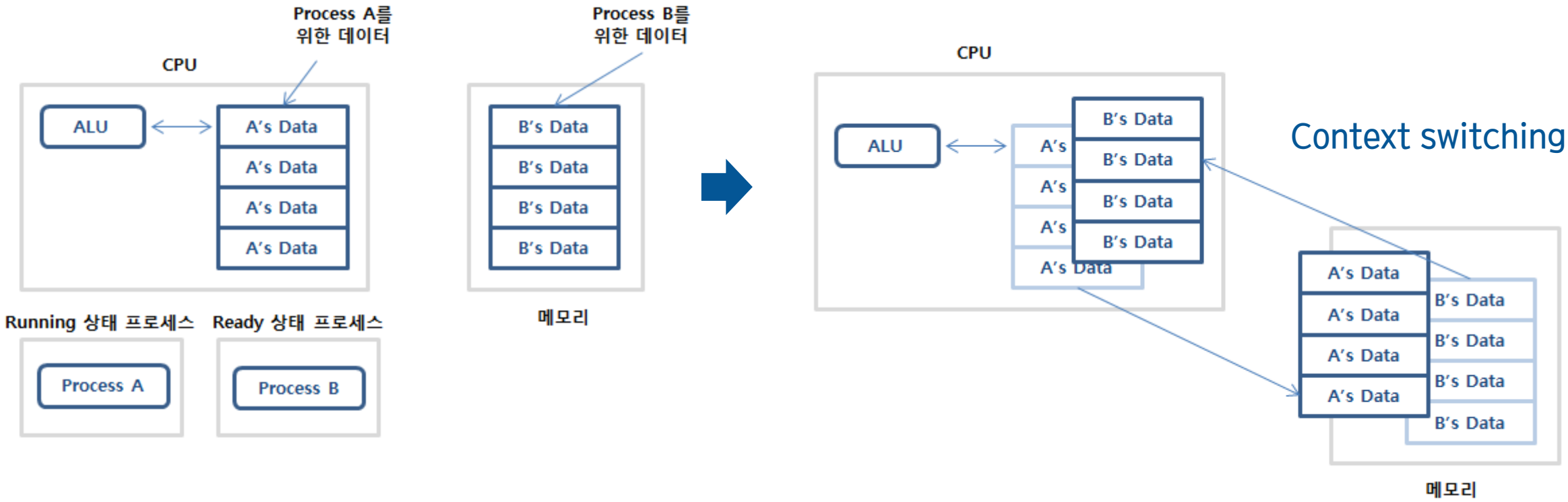
single-threaded process



multithreaded process

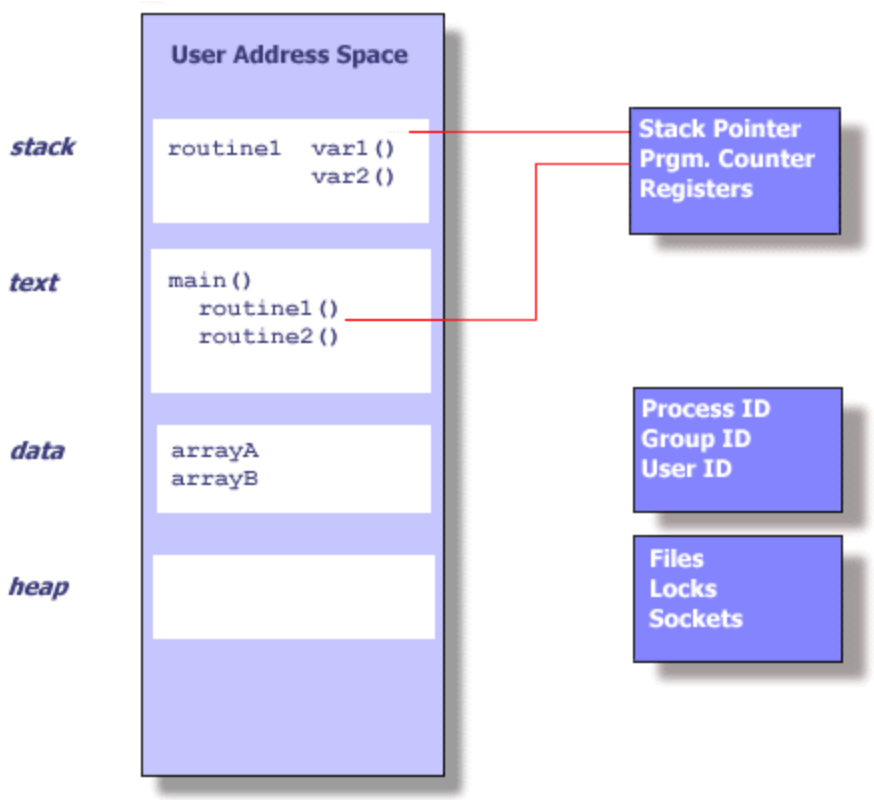
2. Server Model

2.3 Concurrent Server

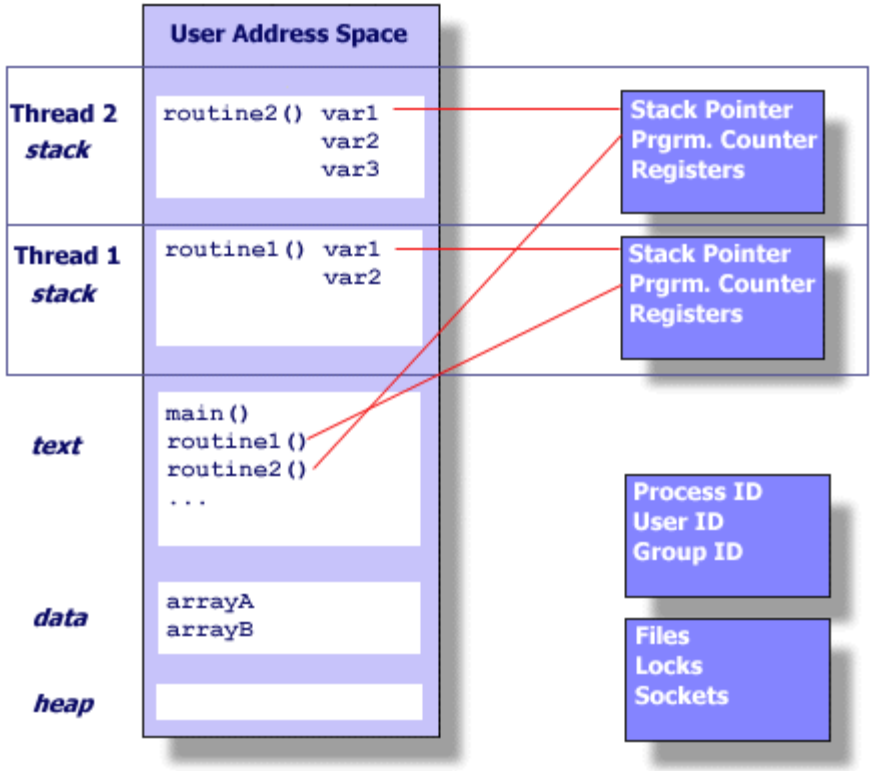


2. Server Model

2.3 Concurrent Server



UNIX Process

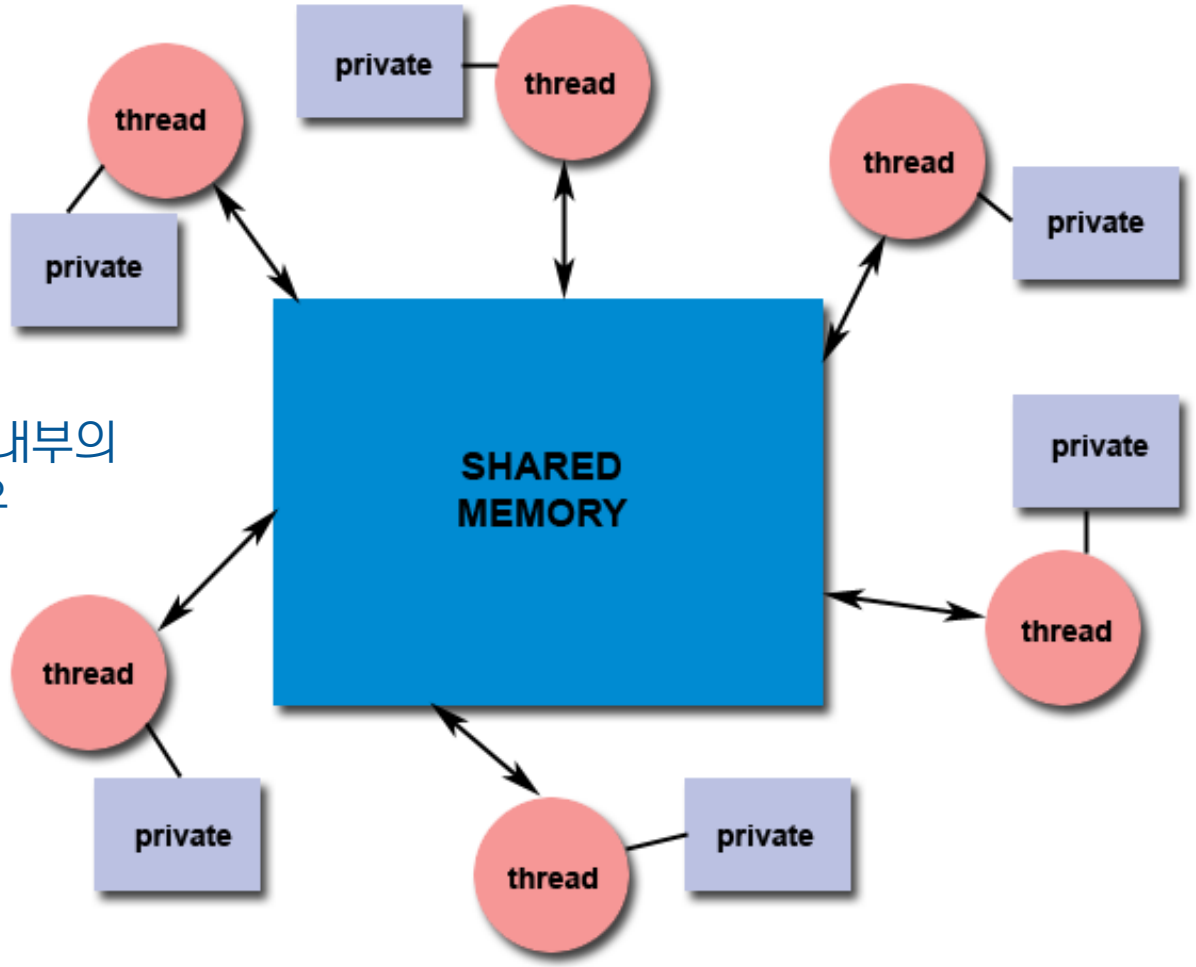


UNIX Process with thread

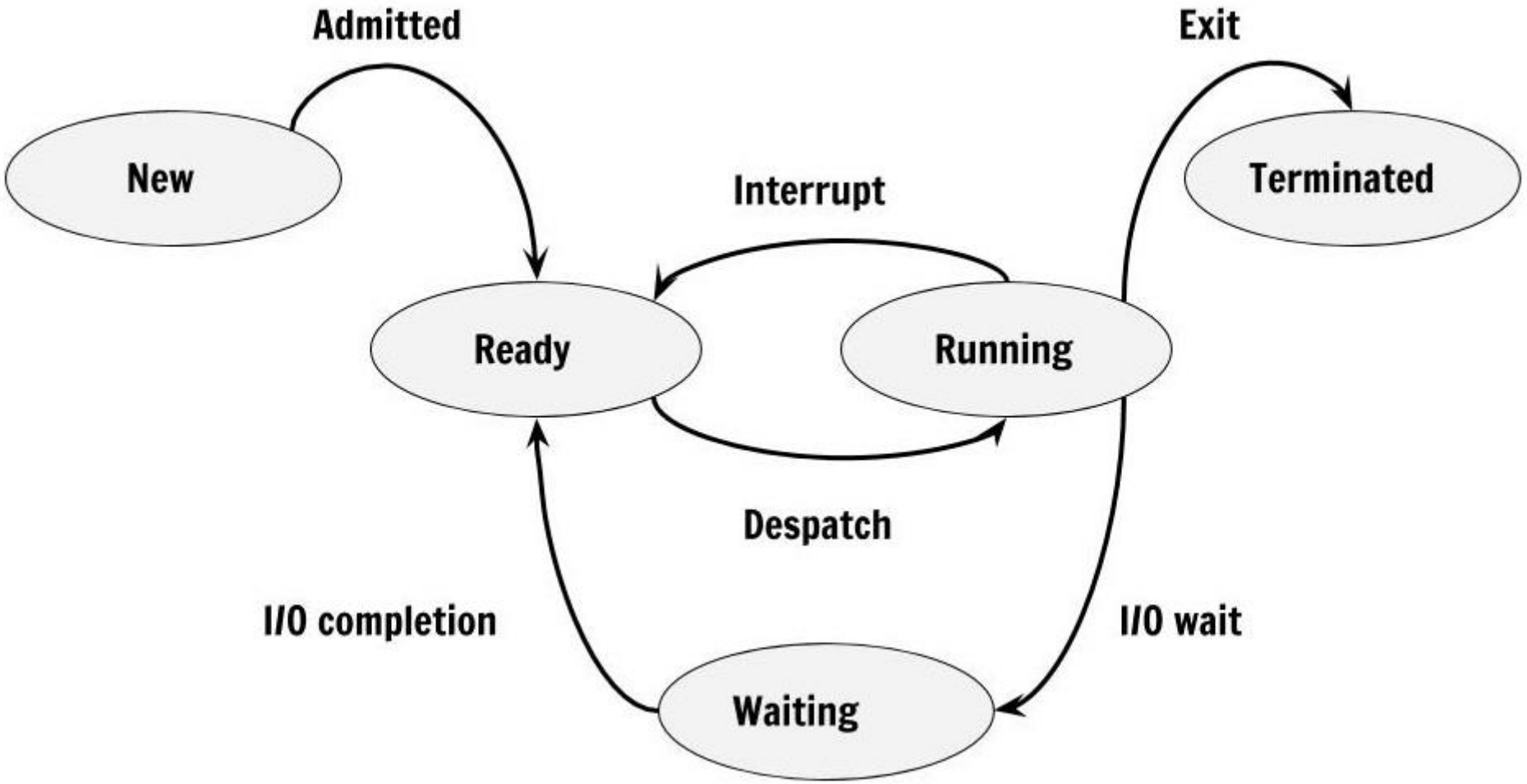
2. Server Model

2.3 Concurrent Server

각 thread가 process 내부의
특정 메모리 영역을 공유



Process state diagram

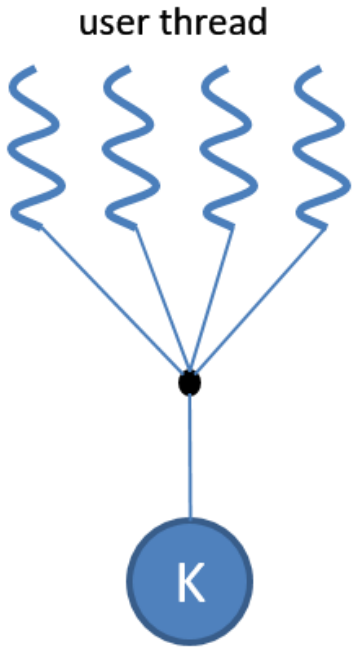


Thread model

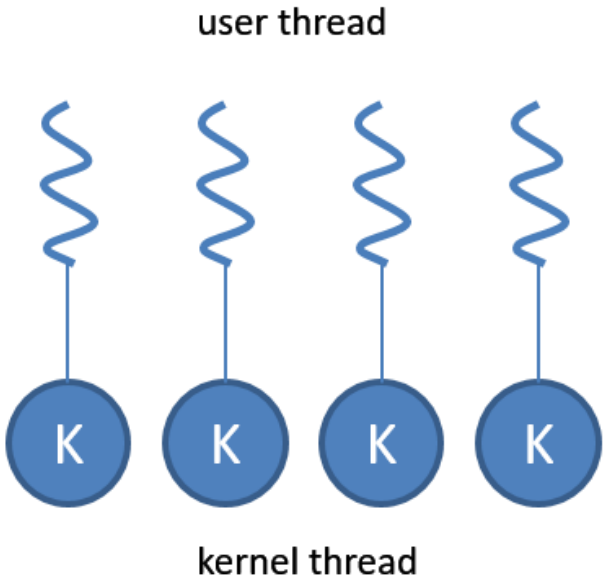
- 1 to 1 모델 : 커널이 모든 스레드를 직접 스케쥴링한다. 스레드 본연의 목적을 가장 잘 나타내는 모델
- 1to N 모델 : 많은 수의 사용자 스레드를 1개의 커널 스레드에 매핑, 사용자 스레드는 라이브러리에서 관리하며 커널은 1개의 스레드로 간주
- N to M 모델 : 여러 개의 사용자 스레드를 그보다 작은 수의 커널 스레드에 매핑, 복잡하지만 가장 효율적인 모델
- Two level 모델 : N to M과 1 to 1을 혼합해서 사용

2. Server Model

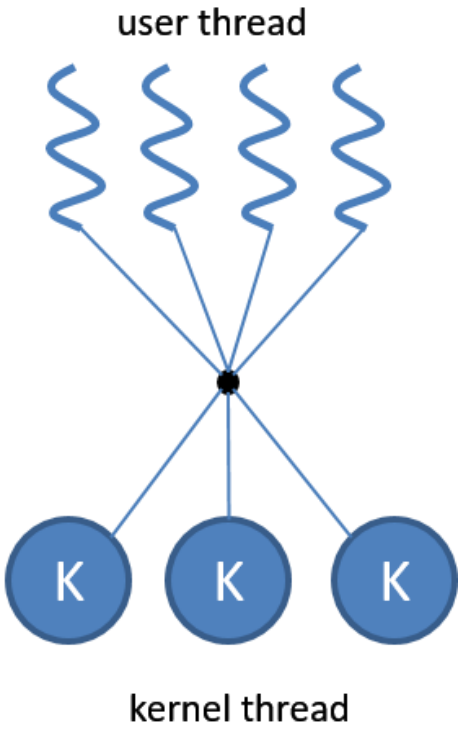
2.3 Concurrent Server



N-to-one model



1-1 model



n-m model

CPython GIL

❖ Global Interpreter Lock은 CPython에서 여러 thread를 이용해 Python 코드를 실행할 경우 단 하나의 스레드만 Python object에 접근하도록 제한하는 mutex이다.

* Mutex : 다수의 thread가 하나의 자원을 사용할 때 충돌이 발생하지 않도록 제어하는 장치

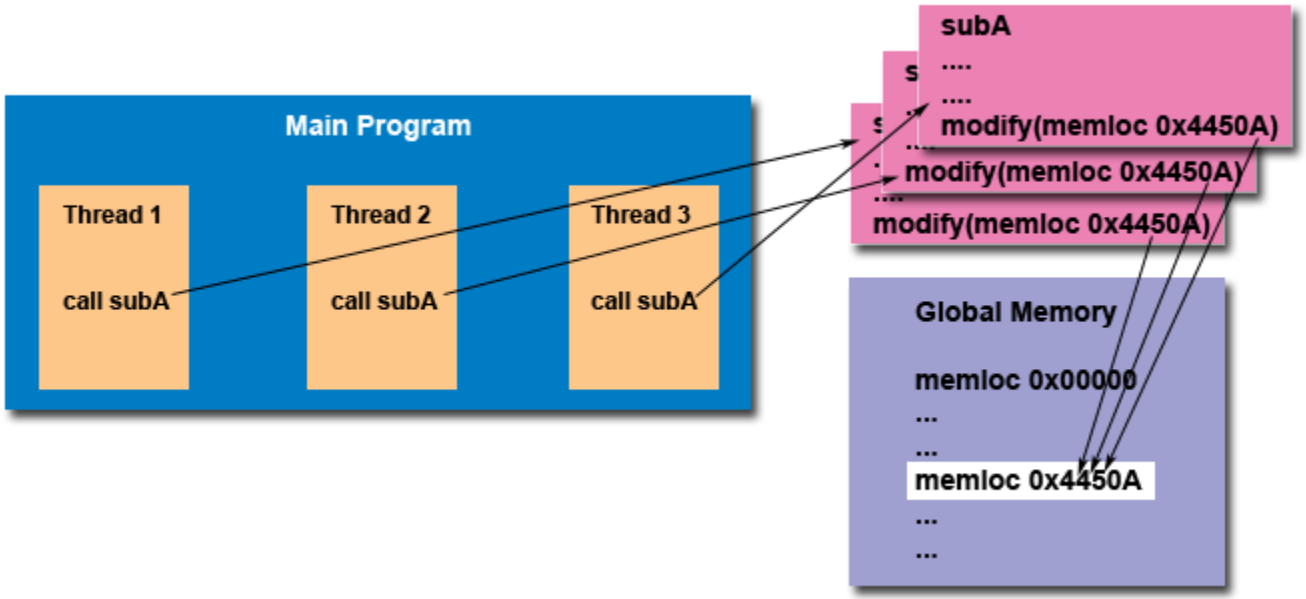
❖ GIL이 필요한 이유는 CPython 구현체가 전역변수 등에 많이 의존하는 등 메모리 관리를 thread-safe하게 하지 않기 때문이다.

❖ GIL은 CPython 인터프리터 내부의 상태를 보호하는 것이지, 사용자가 작성한 코드에 있는 자료구조에 대한 접근을 보호하지 않는다.

- CPython : python 구현 표준으로, C를 이용해 구현한 python 인터프리터
- Jython : Java로 구현한 JVM위에서 실행되는 python 인터프리터
- PyPy : 정적 python으로 구현한 python 인터프리터
- IronPython : .NET 프레임워크에서 구현된 python 인터프리터

Thread-safe

❖ Thread-safe하다는 것은 다수의 thread가 특정 자원에 대해 race-condition을 일으키지 않고 각자의 일을 수행할 수 있음을 뜻한다.



Thread-safe하지 않은 예

Threading 모듈

- Thread를 High-level에서 다룰 수 있게 해주는 모듈
- theading.Thread() 함수에 인자를 전달해 사용하거나, threading.Thread 객체를 상속받은 클래스의 Run() 메소드를 오버라이딩해 thread를 사용할 수 있다.

```python
import threading

def sum(low, high):
    total = 0
    for i in range(low, high):
        total += i
    print("Subthread", total)

t = threading.Thread(target=sum, args=(1, 100000))
t.start()

print("Main Thread")
```

(실행결과)

```
$ python thrd.py
Main Thread
Subthread 4999950000
```

2. Server Model

2.3 Concurrent Server

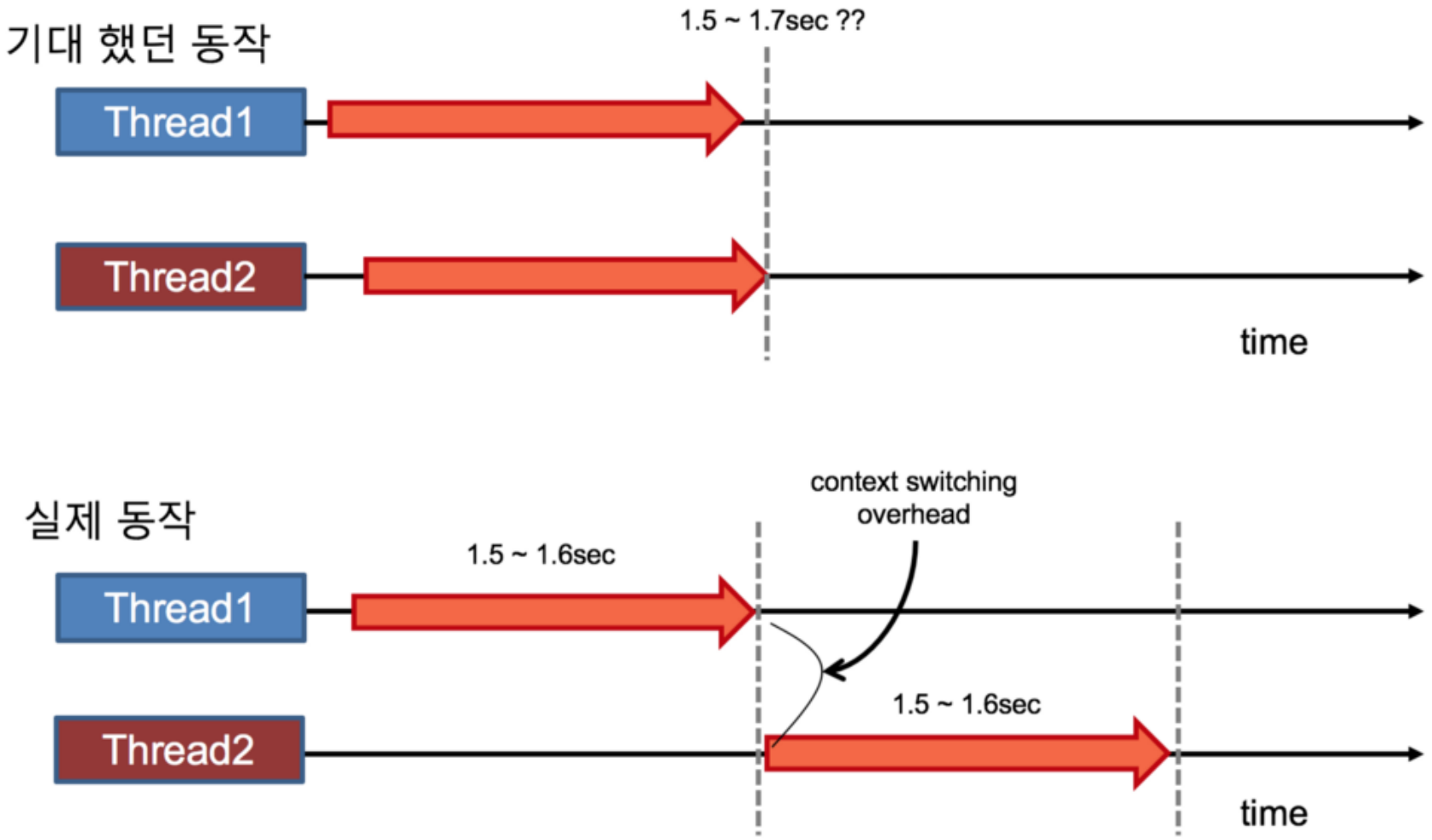
```python
import threading
x = 0 # A shared value

def foo():
    global x
    for i in range(100000000):
        x += 1

def bar():
    global x
    for i in range(100000000):
        x -= 1

t1 = threading.Thread(target=foo)
t2 = threading.Thread(target=bar)
t1.start()
t2.start()
t1.join()
t2.join()
print(x)
```

x가 00이 나오면 thread-safe

```
2617106
>>>
```

2. Server Model

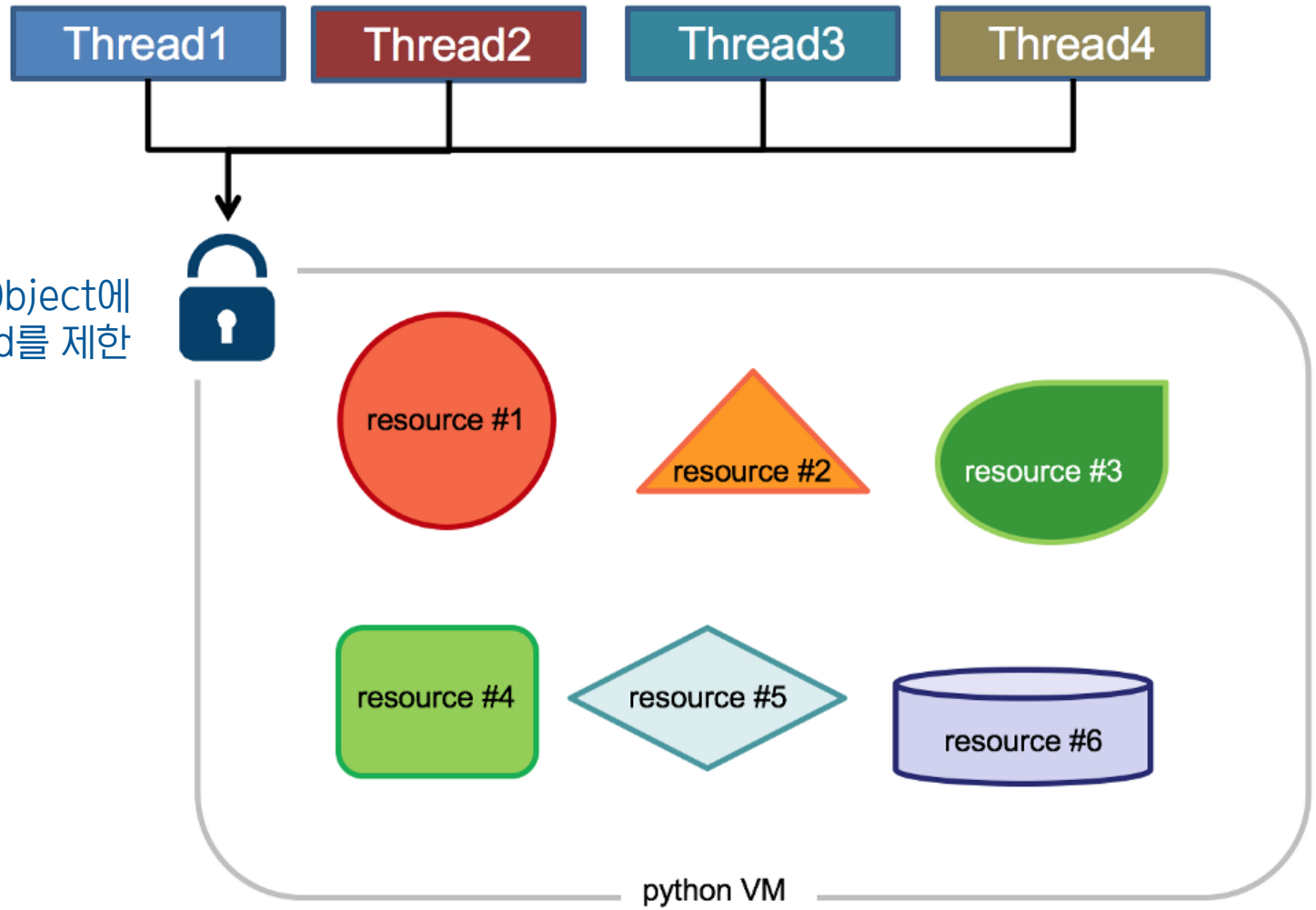
2.3 Concurrent Server



2. Server Model

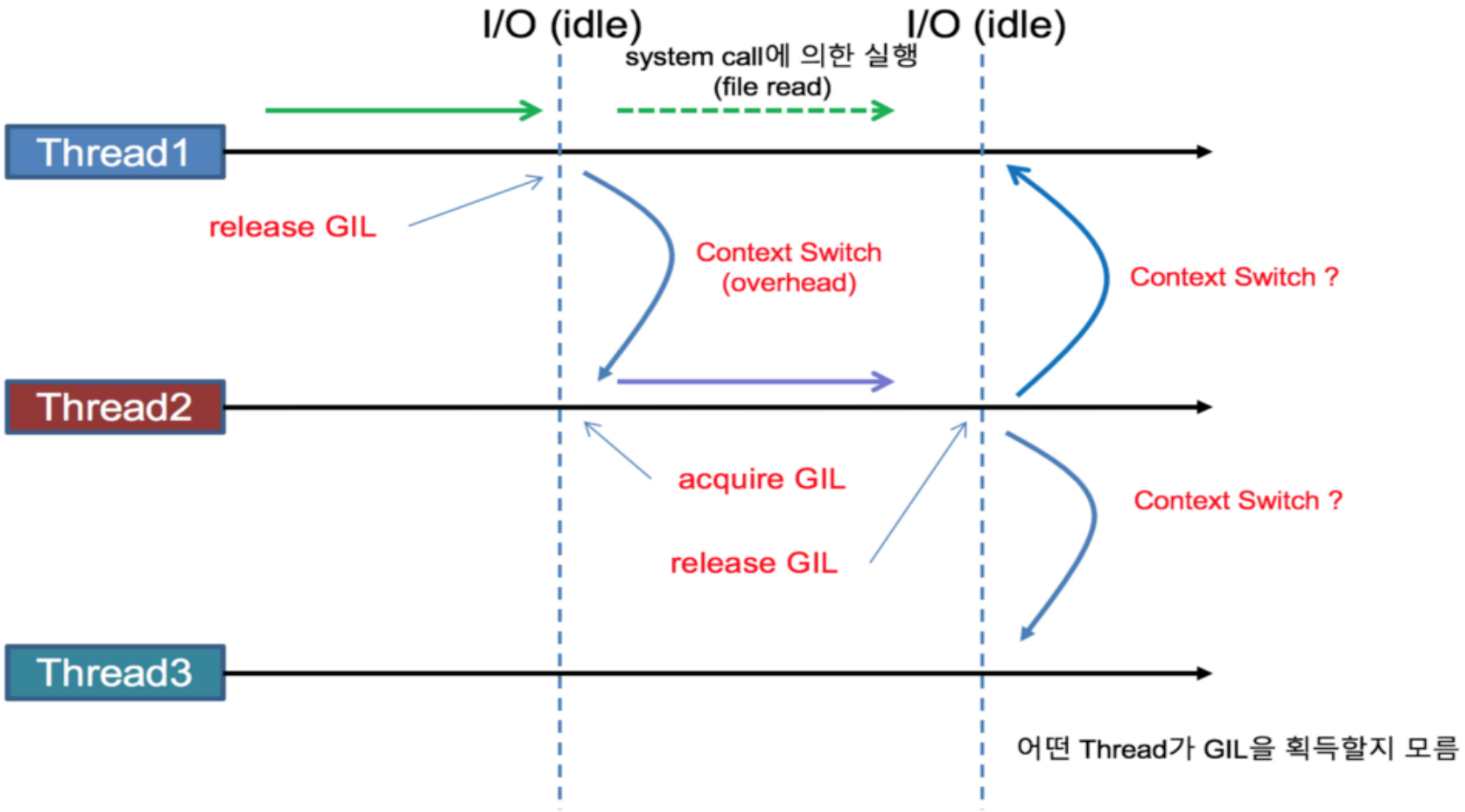
2.3 Concurrent Server

GIL이 Python Object에
접근하는 thread를 제한



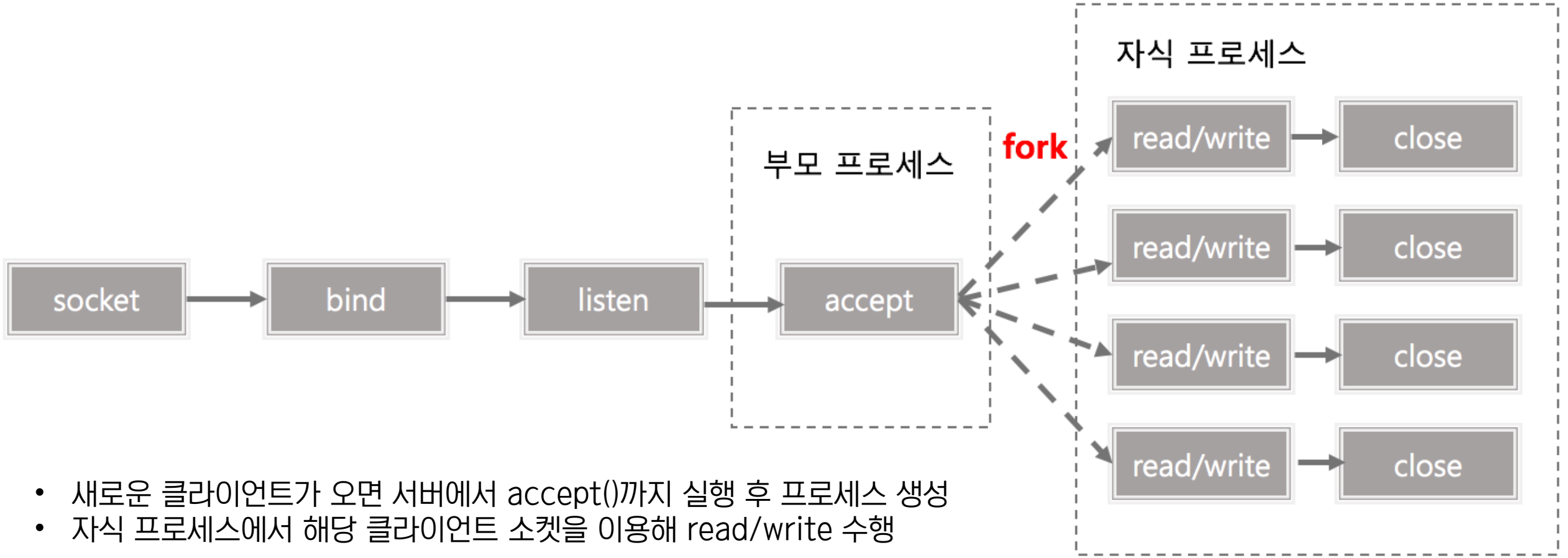
2. Server Model

2.3 Concurrent Server



2. Server Model

2.3 Concurrent Server



- 새로운 클라이언트가 오면 서버에서 accept()까지 실행 후 프로세스 생성
- 자식 프로세스에서 해당 클라이언트 소켓을 이용해 read/write 수행

2. Server Model

2.3 Concurrent Server

- 새로운 클라이언트가 오면 서버에서 accept()까지 실행 후 스레드 생성
- 새로운 스레드에서 해당 클라이언트 소켓을 이용해 read/write 수행

이상적인 서버모델

1. Socket 함수 호출 시 Blocking 최소화
2. I/O를 다른 작업과 병행
3. Thread 및 프로세스의 개수를 최소화
4. 시스템 자원(CPU, memory) 사용량 최소화

Assignment #5

- threading 모듈을 사용해 다수의 client의 요청을 받을 수 있는 서버 작성
  - 서버는 클라이언트가 전송한 문자열을 뒤집어서 클라이언트에게 전송해준다.
  - 클라이언트는 서버 연결 후 input() 함수를 사용해 사용자로부터 문자열을 입력 받는다.
  - python thread_server.py –p 8888
  - python thread_client.py –p 8888 –i 127.0.0.1
- 팀 대표가 barcel@naver.com으로 제출 (4.9일까지)
  - Title : [컴퓨터네트워크][학번][이름][과제_N]
  - Content : github repo url

    팀명 : 길동이네

    팀원 : 홍길동(학번), 고길동(학번)

2. Server Model

2.3 Concurrent Server

Server

```
root@ubuntu:/home/famous/Desktop/network/socket/assignment/assignment_4# python3 thserver.py
Connected to : 127.0.0.1 : 47622
127.0.0.1: Closed
Connected to : 127.0.0.1 : 47624
127.0.0.1: Closed
```

Client1 : 127.0.0.1:47622

```
root@ubuntu:/home/famous/Desktop/network/socket/assignment/assignment_4# python3 thclient.py
>>>Hello World!!!
!!!dlroW olleH
root@ubuntu:/home/famous/Desktop/network/socket/assignment/assignment_4#
```

Client2 : 127.0.0.1:47624

```
root@ubuntu:/home/famous/Desktop/network/socket/assignment/assignment_4# python3 thclient.py
>>>HAHAHAHAH
HAHAHAHAH
root@ubuntu:/home/famous/Desktop/network/socket/assignment/assignment_4#
```