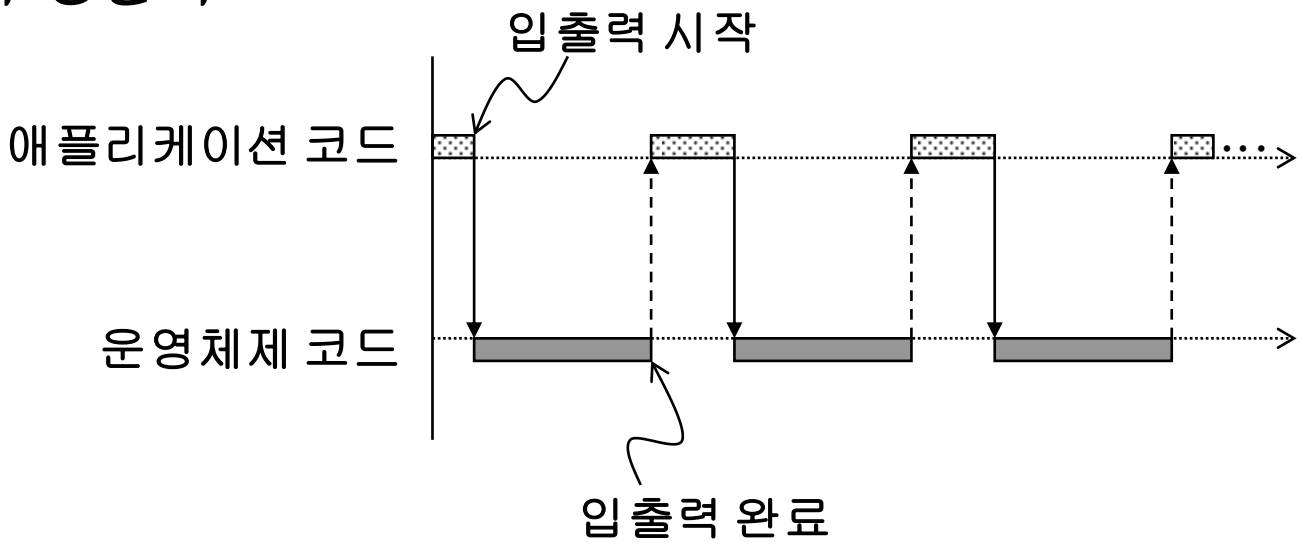
Network Programming - 04

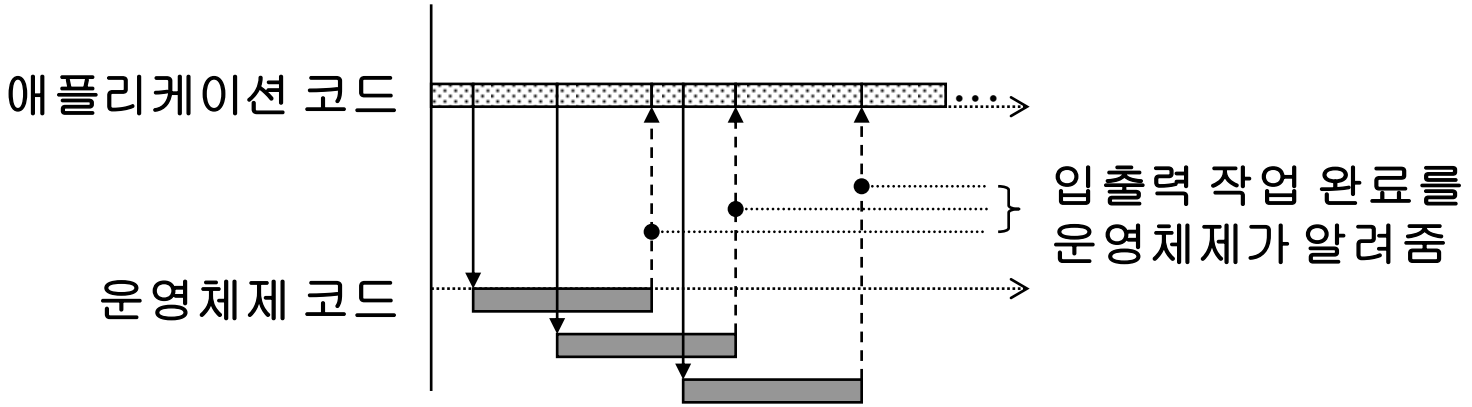
afewhee@gmail.com

- I/O 모델
- IOCP
- 네트워크 Util
  - 메시지 큐, 링 버퍼
  - 패킷 디자인

- 동기식 입출력



- 비동기식 입출력



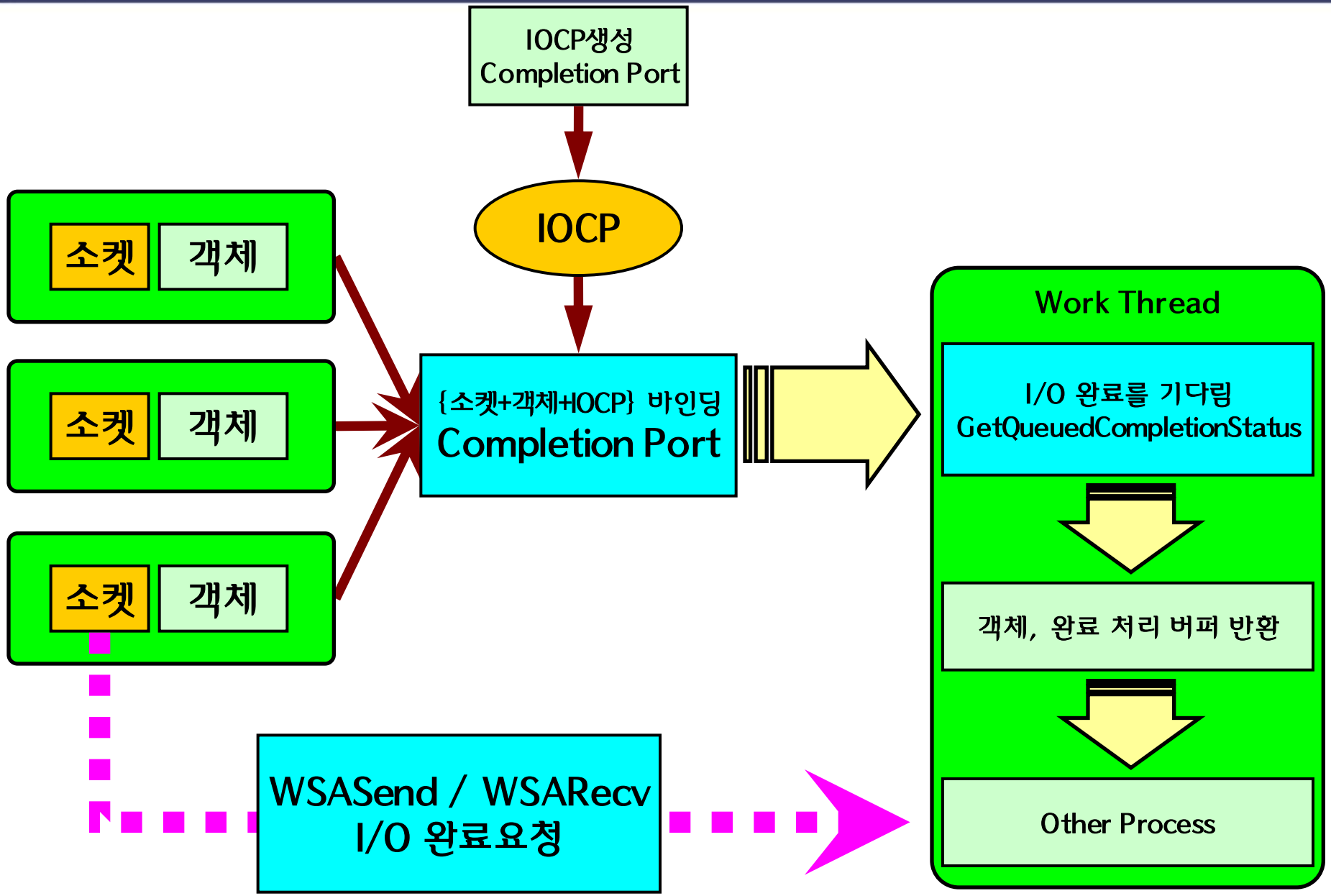
- 동기식 입출력(synchronous I/O)
  - 애플리케이션은 입출력 함수를 호출한 후 입출력 작업이 끝날 때까지 대기
  - 입출력 작업이 끝나면 입출력 함수는 리턴하고 애플리케이션은 입출력 결과를 처리하거나 다른 작업을 진행

- 비동기식 입출력(asynchronous I/O)
  - 윈도우의 Overlapped I/O
  - 애플리케이션은 입출력 함수를 호출한 후 입출력 작업의 완료 여부와 무관하게 다른 작업을 진행
  - 입출력 작업이 끝나면 운영체제는 작업 완료를 애플리케이션에게 알려줌. 이때 애플리케이션은 다른 작업을 중단하고 입출력 결과를 처리
  - 윈도우의 특화 ➔ IOCP

- 입출력 방식에 따른 소켓 입출력 모델 분류
  - 동기 입출력 + 비동기 통지
    - Select 모델, WSAAsyncSelect 모델, WSAEventSelect 모델
  - 비동기 입출력 + 비동기 통지
    - Overlapped 모델(I), Overlapped 모델(II), Completion Port 모델

- Overlapped 모델 사용 절차
  - 비동기 입출력을 지원하는 소켓 생성
  - 비동기 입출력을 지원하는 소켓 함수 호출
  - 운영체제가 소켓 입출력 작업 완료를 애플리케이션에게 비동기로 통지
  - 애플리케이션은 결과를 처리

- Overlapped 모델(I) - 이벤트 방식
  - I/O 완료와 바인딩 된 이벤트 객체를 이용해서 애플리케이션이 이벤트 객체로 작업 완료를 감지.
- Overlapped 모델(II) - 완료루틴 방식
  - 소켓 입출력 작업이 완료되면, 콜백 함수(callback function)로 구현된 완료 루틴 함수를 자동으로 호출
- Overlapped 모델(III) - IOCP
  - I/O 완료를 담당하는 Completion Port를 이용한 모델
  - IOCP: 비동기 입출력 결과와 이 결과를 처리할 스레드에 대한 정보를 담고 있는 구조
  - 특징: Overlapped I/O + 비동기 Notification I/O
  - 장점: 제한된 쓰레드(Work Thread)를 통해서 여러 소켓의 입출력 처리. 컨텍스트 스위칭에 소비되는 시간을 줄임



2. IOCP - 구현


```
RemoteHost* pRmH = new RemoteHost;

CreateIoCompletionPort((HANDLE)scH
, hIocp
, (DWORD)pRmH
, 0);
```

```
pRmH->AsyncRecv();
```

```
RemoteHost* pRmH      = NULL;
OVERLAPPED_EX* pOL    = NULL;
DWORD dTran     = 0;
...
hr = GetQueuedCompletionStatus(
        hIocp
        , &dTran
        , (LPDWORD)&pRmH
        , (LPOVERLAPPED*)&pOL
        , INFINITE);
```

```
struct OVERLAPPED_EX
{
   OVERLAPPED OL;
   DWORD  dTran; // Transfered
   DWORD dFlag; // Flag
   WSABUF WsBuf;
   ...};
```

```
struct RemoteHost
{
   SOCKET  scH;      // 소켓
   OVERLAPPED_EX   olSnd;  // For WSASend
   OVERLAPPED_EX   olRcv;  // For WSARecv
   ...
   INT AsyncSend(char* sBuf/*in*/, int*
iSize){ ...
       hr = WSASend( scH // 송신 소켓
       , &(olSnd.WsBuf)    // 송신 버퍼 포인터
       , 1 // 송신 버퍼의 수
       , &olSnd.dTran, olSnd.dFlag
       , (OVERLAPPED*)&(olSnd) , NULL);
       ...}

   INT AsyncRecv() {...
       hr = WSARecv( scH // 수신 소켓
       , &(olRcv.WsBuf)    // 수신 버퍼 포인터
       , 1 // 수신 버퍼의 수
       , &olRcv.dTran, &olRcv.dFlag
       , (OVERLAPPED*)&(olRcv) , NULL);
       ...}
};
```

2. IOCP - 소켓 입출력 절차

- I/O 완료 포트 생성:
  - CreateIoCompletionPort()
- Work Thread를 CPU에 맞게 적절히 생성:
  - GetQueuedCompletionStatus() 함수를 호출, 대기 상태로 만듬
- 중첩된 비동기 입출력 소켓 생성:
  - WSASocket(..., WSA_FLAG_OVERLAPPED)
- {소켓 + IOCP + 객체} 바인딩, 비동기 입출력 결과는 입출력 완료 포트에 저장됨
  - CreateIoCompletionPort()
- {소켓 + 완료 처리 버퍼} 비동기 요청
  - WSASend(), WSARecv()
  - 비동기 입출력 작업이 곧바로 완료되지 않으면, 소켓 함수는 오류를 리턴, 오류 코드는 WSA_IO_PENDING으로 설정됨
- {객체 + 완료 처리 버퍼} 결과를 비동기 완료 반환:
  - GetQueuedCompletionStatus() 함수에서 객체, 완료 처리 버퍼 등을 반환

- 적절한 Work Thread 수
  - 경험적 ==> CPU * 1 ~ 2

Ex)

```
SYSTEM_INFO SystemInfo;
GetSystemInfo(&SystemInfo);
nWrkPrc =SystemInfo.dwNumberOfProcessors * 2;

for(int i=0; i<nWrkPrc; i++)
    HANDLE hThWrk = (HANDLE)_beginthreadex(NULL, 0
    , (unsigned (__stdcall*)(void*))ThreadWork, (void*)hIocp, 0, NULL);
```

- 중첩된 비동기 입출력 소켓 생성
  - WSASocket(PF_INET, SOCK_STREAM, 0, NULL, 0, WSA_FLAG_OVERLAPPED);

- IOCP 객체 생성
  - hIocp = CreateIoCompletionPort(INVALID_HANDLE_VALUE, NULL, 0, 0);

2. IOCP - 주요함수

- 입출력 완료 포트 생성: {소켓 + IOCP + 객체} 바인딩
  - HANDLE CreateIoCompletionPort (
    HANDLE FileHandle, // IOCP와 바인딩할 소켓
    HANDLE ExistingCompletionPort, // IOCP 핸들
    ULONG_PTR CompletionKey, // I/O를 요청한 객체 또는 완료를 구분해주는 객체
    DWORD NumberOfConcurrentThreads // 동시에 실행될 수 있는 작업 스레드 수 = 보통 0
    );
  - 성공: 입출력 완료 포트 핸들, 실패: NULL

- 전송에 대한 비동기 요청: {소켓 + 송신 처리 버퍼} 비동기 요청
  - int WSASend (SOCKET s, // 송신 버퍼 소켓
    LPWSABUF lpBuffers, // 송신 버퍼 포인터
    DWORD dwBufferCount, // 송신 버퍼 수. 보통 1
    LPDWORD lpNumberOfBytesSent, // 송신 버퍼 크기 ➔ 계산해서 전달
    DWORD dwFlags, // 0으로 설정
    LPWSAOVERLAPPED lpOverlapped, // OVERLAPPED 구조체 포인터
    LPWSAOVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine // IOCP에서는 NULL
) ;
  - 성공: 0, 실패: SOCKET_ERROR

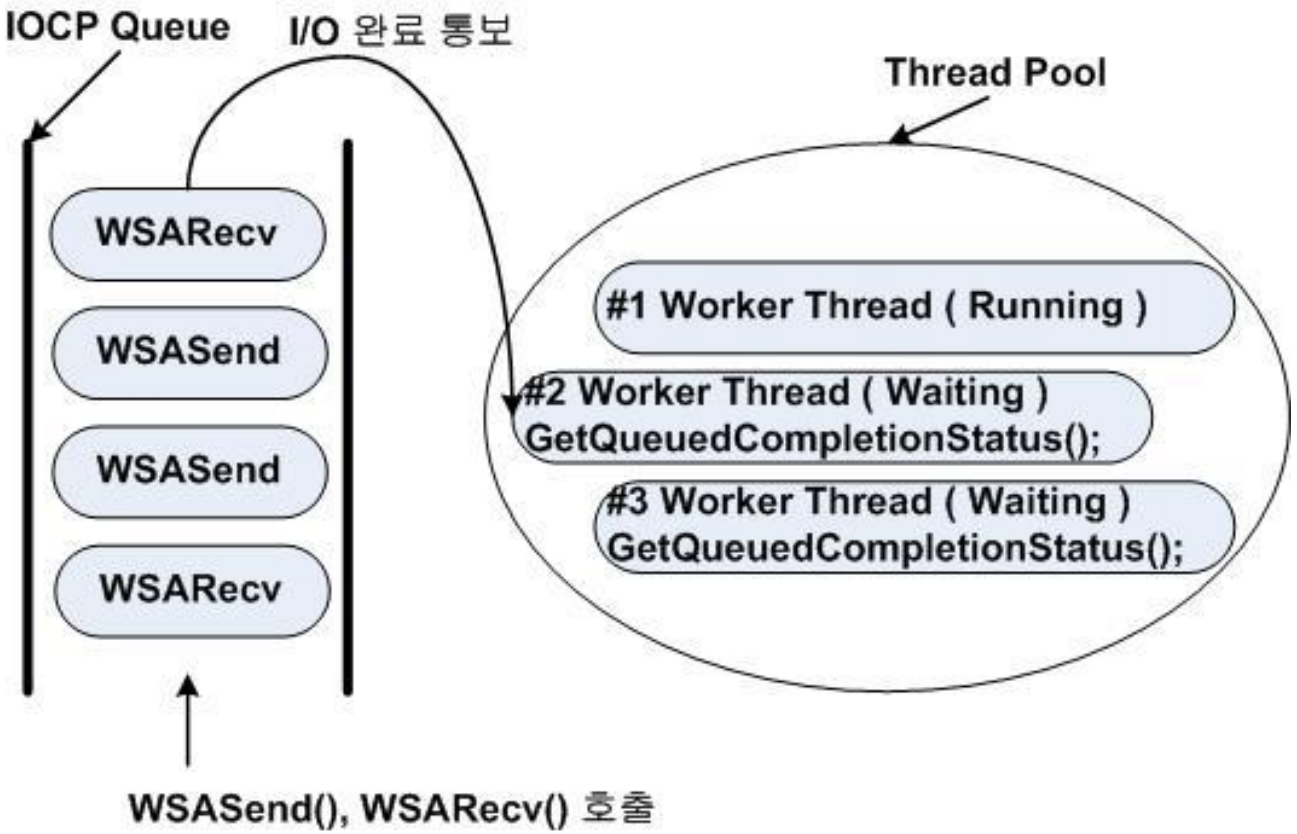
- 수신에 대한 비동기 요청: {소켓 + 수신 처리 버퍼} 비동기 요청
  - int WSARecv (SOCKET s, // 수신 소켓
    LPWSABUF lpBuffers, // 수신 버퍼 포인터
    DWORD dwBufferCount, // 수신 버퍼 수, 보통 1
    LPDWORD lpNumberOfBytesRecvd, // 0으로 설정 DWORD dTran= 0; &dTran
    LPDWORD lpFlags, // 0으로 설정 DWORD dFlag= 0; &dFlag
    LPWSAOVERLAPPED lpOverlapped, // 보통 OVERLAPPED 구조체 포인터로 캐스팅
    LPWSAOVERLAPPED_COMPLETION_ROUTINE lpCompletionRoutine // IOCP에서는 NULL
) ;
  - 성공: 0, 실패: SOCKET_ERROR

2. IOCP - 주요함수

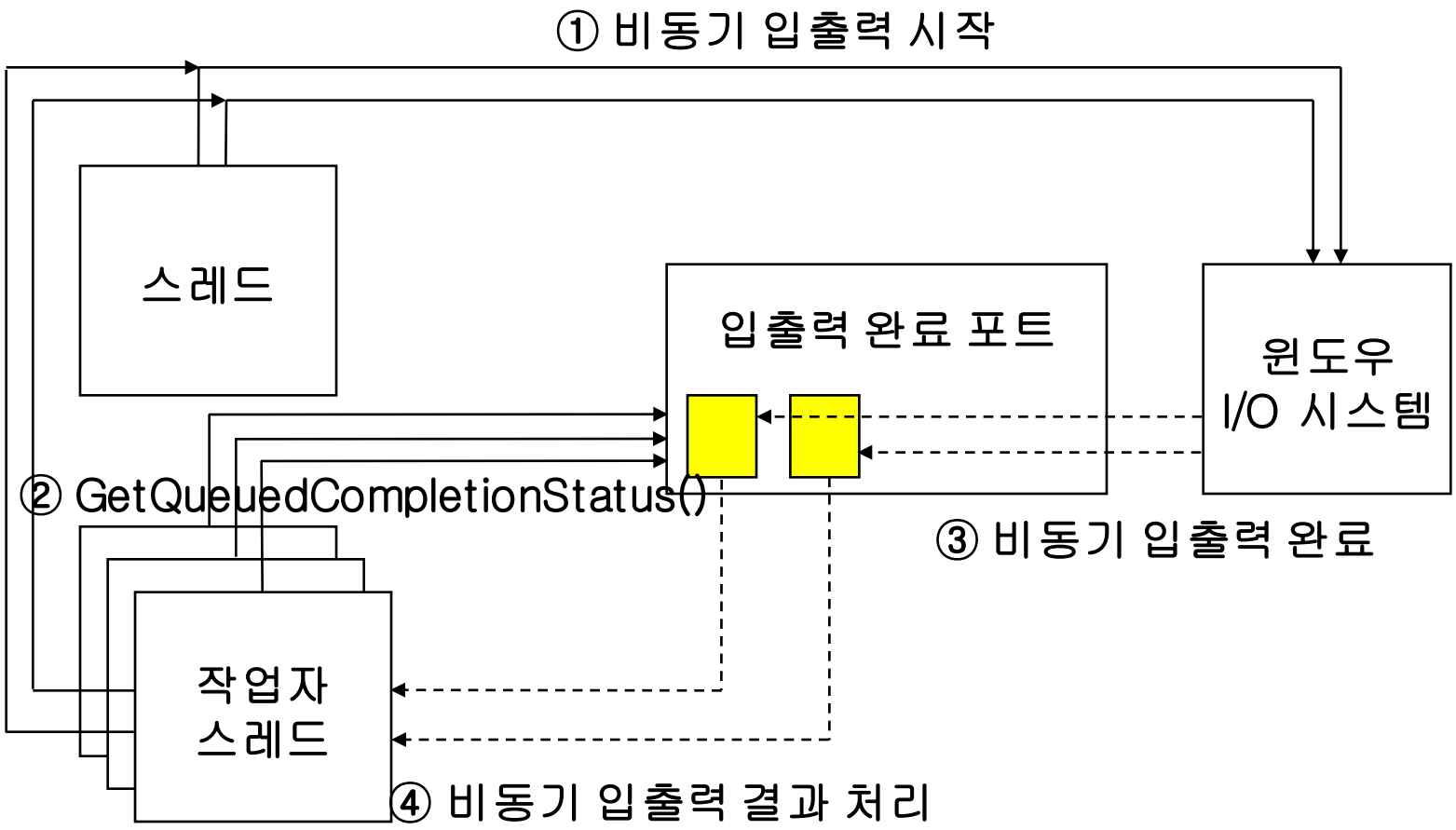
- 비동기 입출력 결과 확인: {객체 + 완료 처리 요청 버퍼} 결과 비동기로 반환
  - BOOL GetQueuedCompletionStatus (
    HANDLE CompletionPort, // IOCP 핸들
    LPDWORD lpNumberOfBytes, // 완료된 I/O의 총 바이트 수
    PULONG_PTR lpCompletionKey, // I/O를 요청한 객체
    LPOVERLAPPED *lpOverlapped, // Overlapped 구조체의 주소
    DWORD dwMilliseconds // 타임아웃 INFINITE ➔ I/O 완료까지 기다림
    );
  - 성공: 0이 아닌 값, 실패: 0
  - 쓰레드에서 호출
  - 완료 결과를 IOCP 내의 I/O 완료 패킷 큐로부터 하나씩 꺼내옴

- IOCP Queue에 직접 데이터를 넣는 경우:
  - BOOL PostQueuedCompletionStatus (
    HANDLE CompletionPort, // IOCP 핸들
    DWORD dwNumberOfBytesTransferred, // bytes transferred
    ULONG_PTR dwCompletionKey, // completion key
    LPOVERLAPPED lpOverlapped // overlapped buffer
    );
  - 성공: 0이 아닌 값, 실패: 0
  - 작업 스레드에게 I/O 완료 처리와 상관없이 메시지 전달할 때 사용

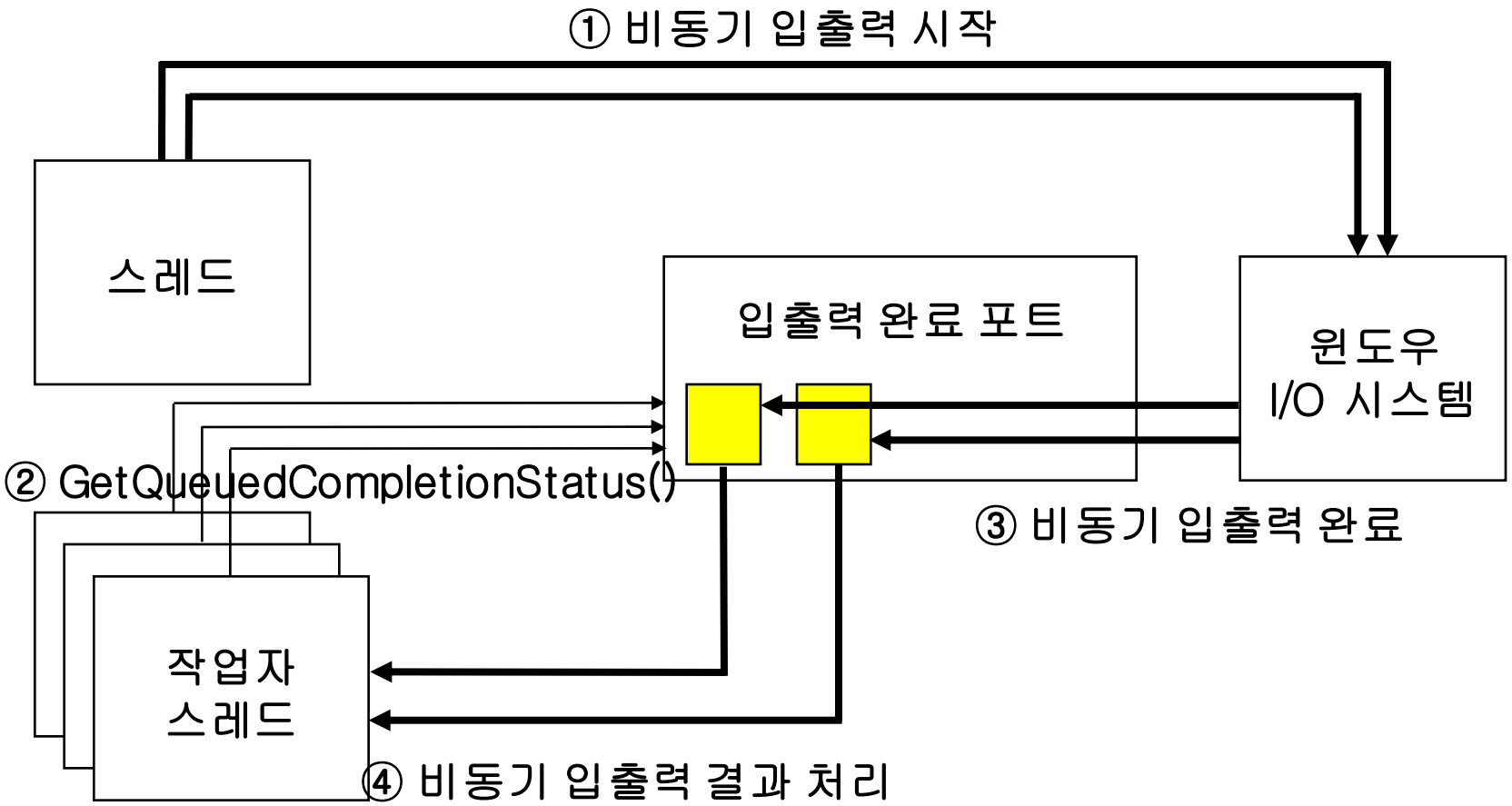
- Completion Port 모델 동작 원리



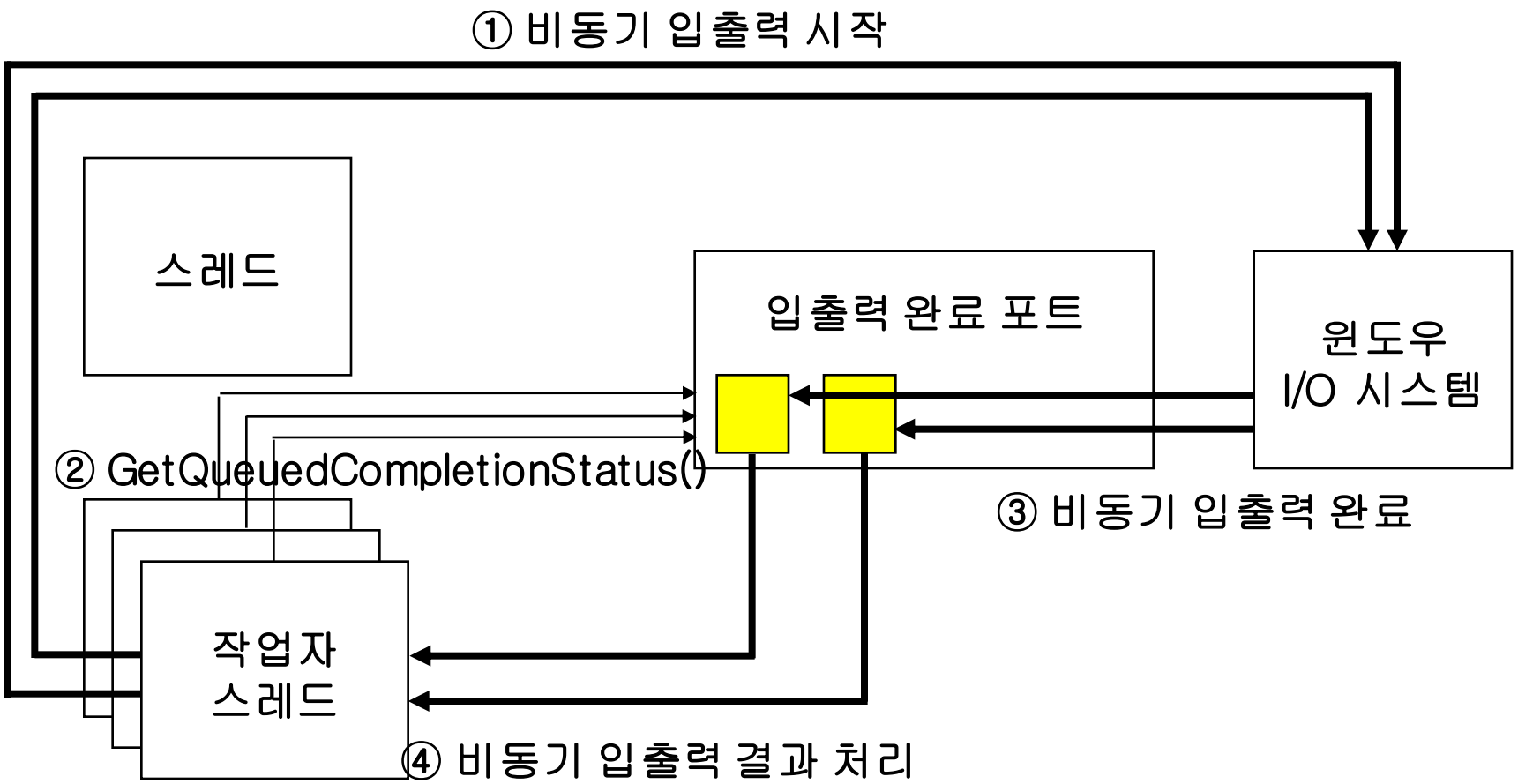
Completion Port 모델 동작 원리



Completion Port 모델 동작 원리



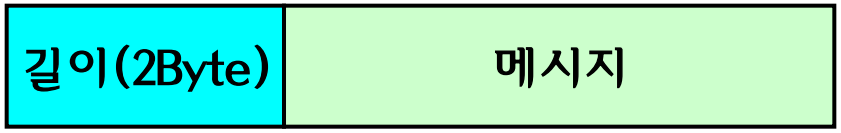
Completion Port 모델 동작 원리



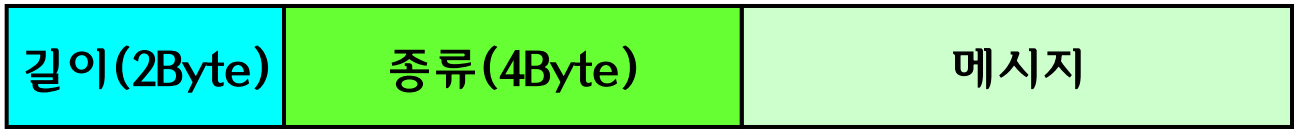
```
template<class T>
class TqueCircular
{
protected:
    T*    m_pBuf;          // 큐의 버퍼
    int   m_iFront;        // 시작 위치
    int   m_iRear;         // 끝 위치
    int   m_iSpace;        // 남아있는 큐의 버퍼 크기
    int   m_iWidth;        // 큐의 버퍼 크기
public:
    TqueCircular();
    TqueCircular(int iWidth);
    virtual ~TqueCircular();
    int   Enqueue(T* p);  // 데이터 추가
    int   Dequeue(T* p);  // 데이터 꺼냄
};


template<class T>
TqueCircular<T>::TqueCircular()
...
```

```
typedef    unsigned char        BYTE;

class CQueStr
{
protected:
    BYTE* m_sBuf;                          // 큐의 버퍼
    int     m_iFront;                      // Front 위치
    int     m_iRear;                       // Rear 위치위치
    int     m_iSpace;                      // 남아 있는 공간
    int     m_iWidth;                      // 큐의 버퍼 크기

public:
    CQueStr();
    CQueStr(int iWidth);
    virtual ~CQueStr();

    int     Enqueue(void* sIn,int iSize);  // 데이터 추가기. 리턴 값은 추가 못한 나머지 사이즈
    int     Dequeue(void*sOut,int* iSize); // 데이터 가져오기. 리턴 값은 꺼내온 사이즈

    void    Clear();                       // 큐의 내용을 지움
    int     GetWidth();                    // 큐의 크기를 가져옴
    int     GetRemain();                   // 큐의 남아 있는 크기를 가져옴
    void    Resize(int iSize);             // 큐의 사이즈를 다시 설정
};
```

- 단순 패킷 구조



- 패킷 종류가 들어간 구조



- End Marker가 들어간 구조