CS4514 HELP Session 3

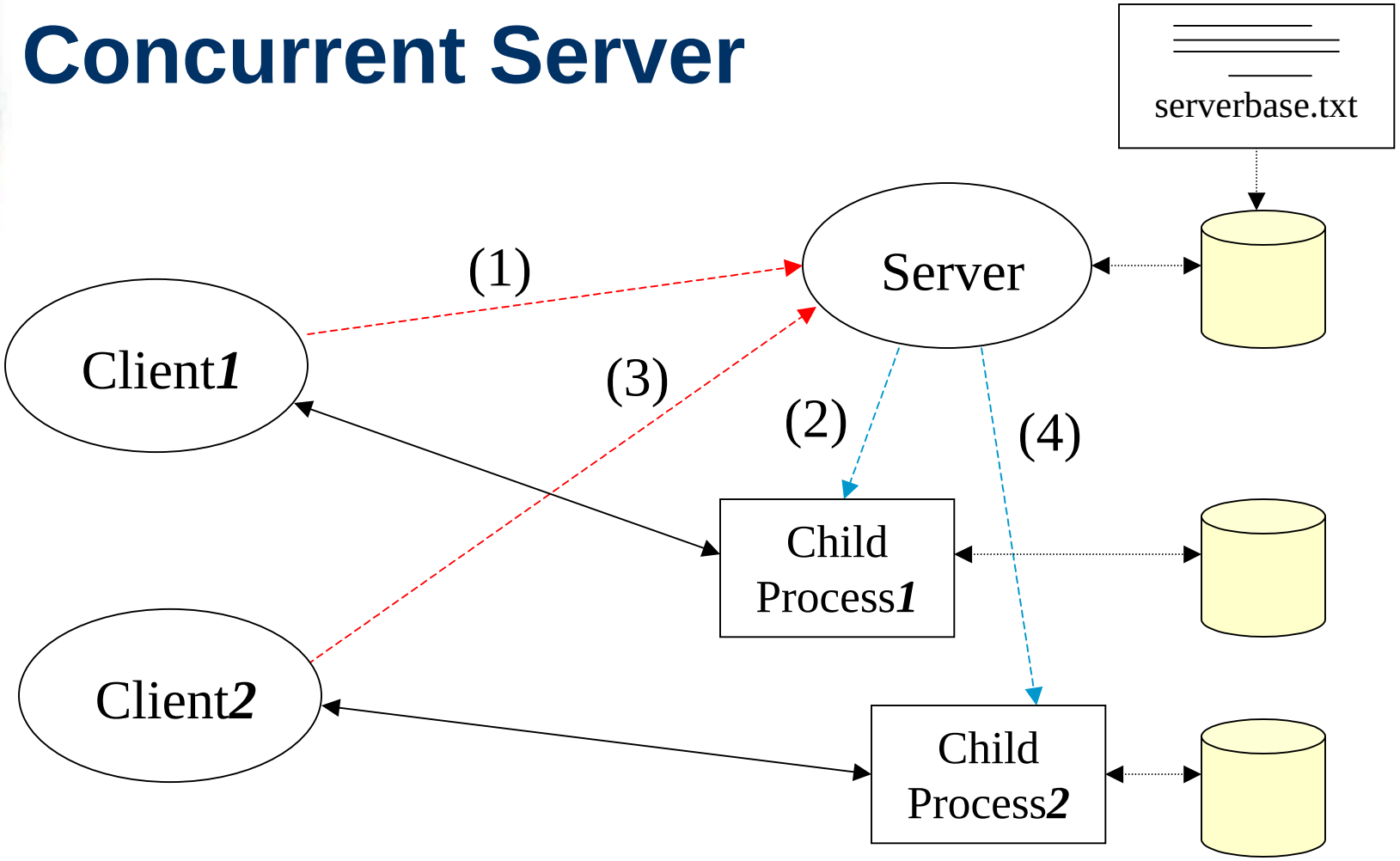
Concurrent Server (in APP)

Using Go-Back-N (in DLL)

Description

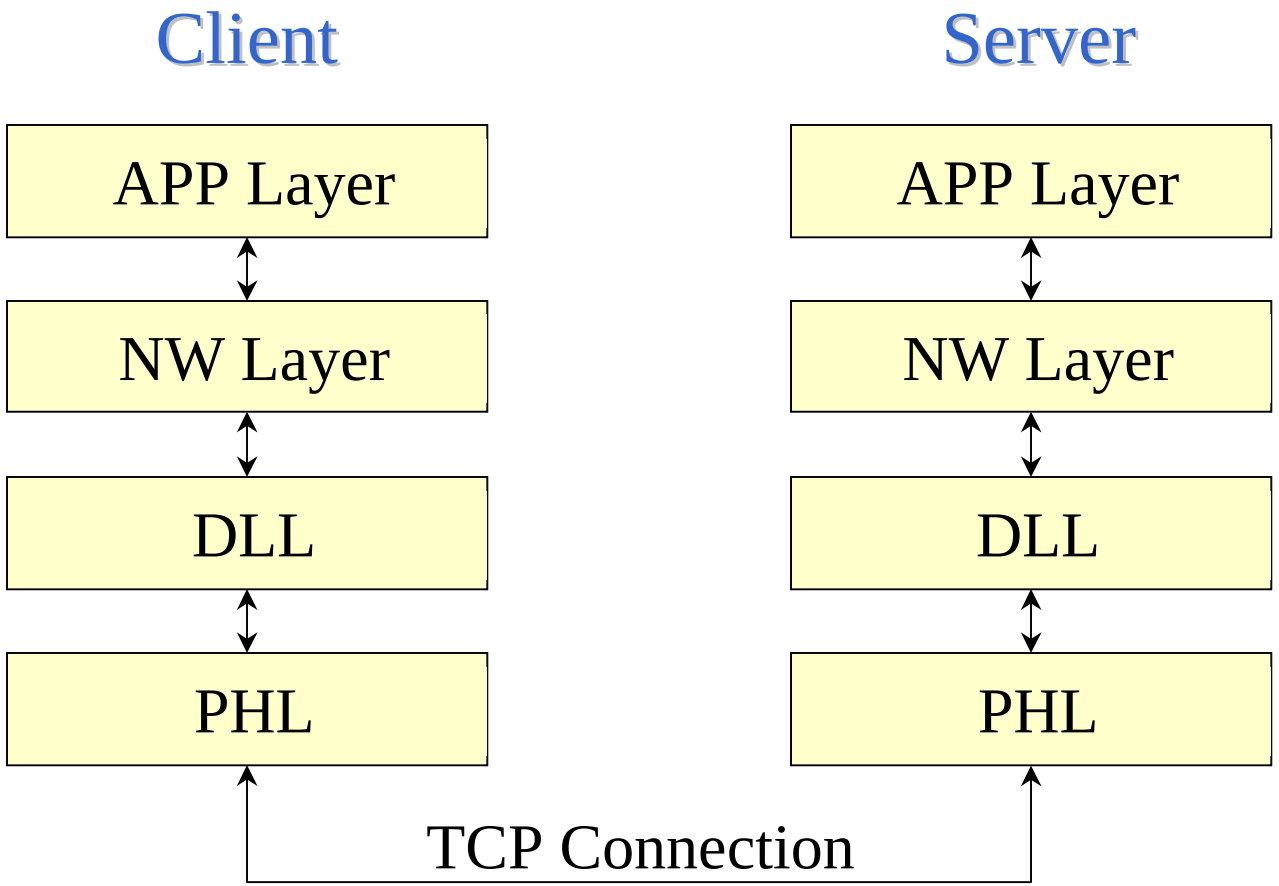
- You are supposed to implement a simple concurrent server on top of an emulated network protocol stack (NWL, DLL, PHL).
  - Network layer: Packet management only
  - Datalink layer: Go-Back-N sliding window protocol
  - Physical layer: An error module + TCP connection.
- Your programs should compile and work on garden.WPI.EDU.

Concurrent Server



Note: each child process keeps a separate copy of the DB.

Framework

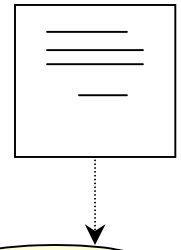

Client
Server
APP Layer
NW Layer
DLL
PHL
APP Layer
NW Layer
DLL
PHL
TCP Connection

Client*i*

Server

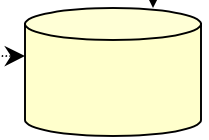
Read “scripted action” from file “script*i*.txt”

Read/Write a message



APP

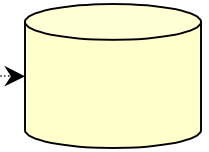
APP



Client Request:

cmd No. [msg]

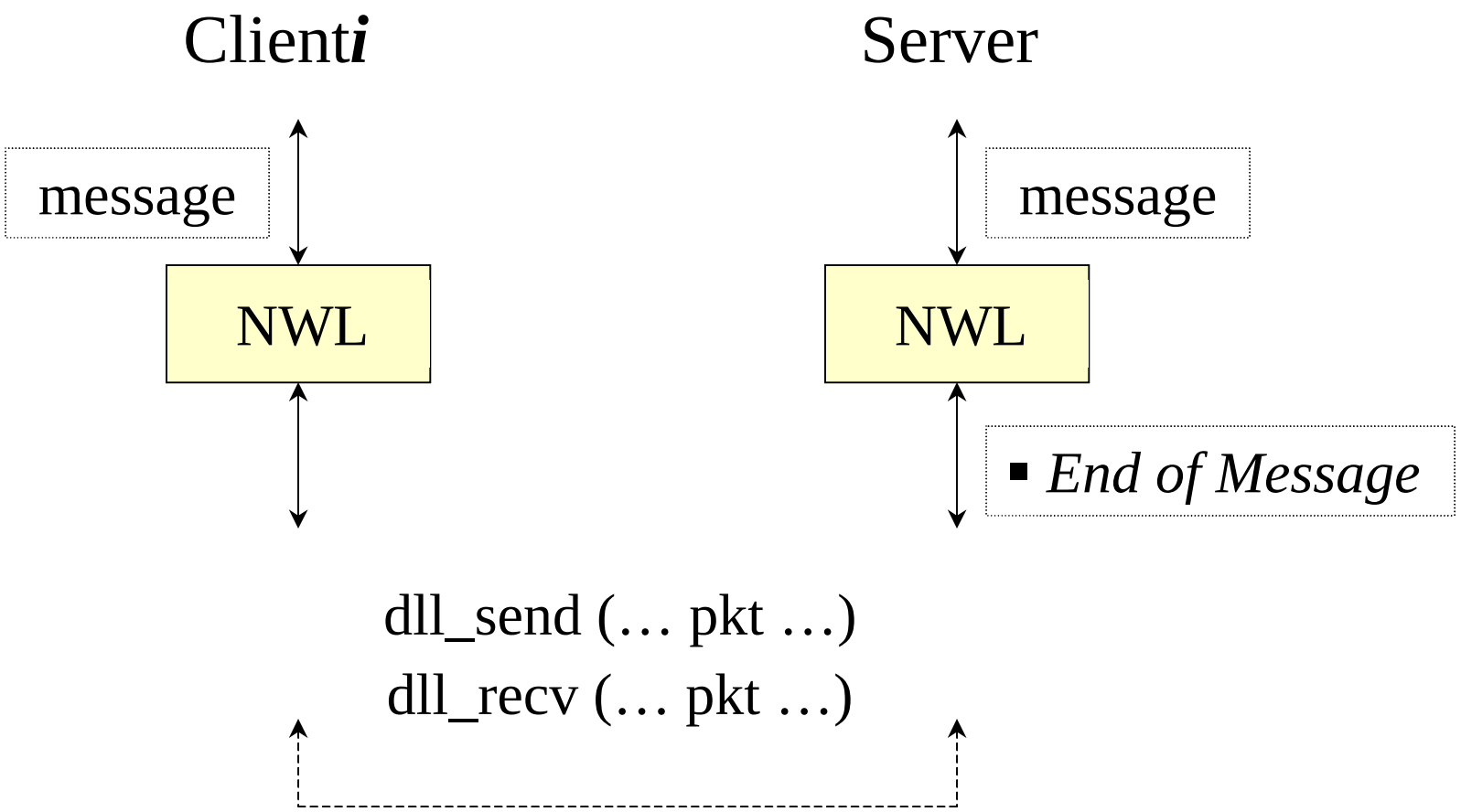
Child Process *i*



nwl_send (… msg …)

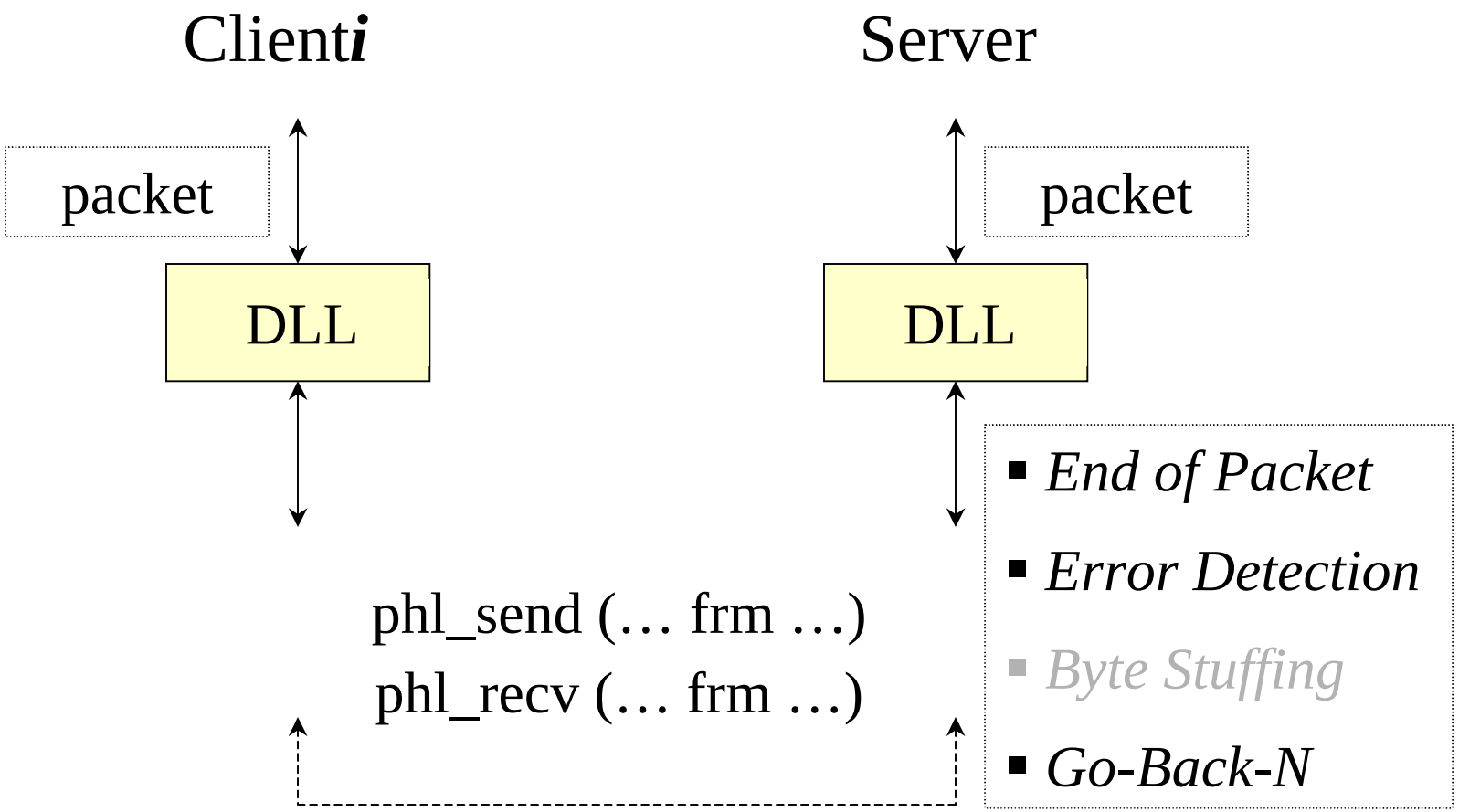
nwl_recv (… msg …)

Note: The max_size of a message is 320 bytes

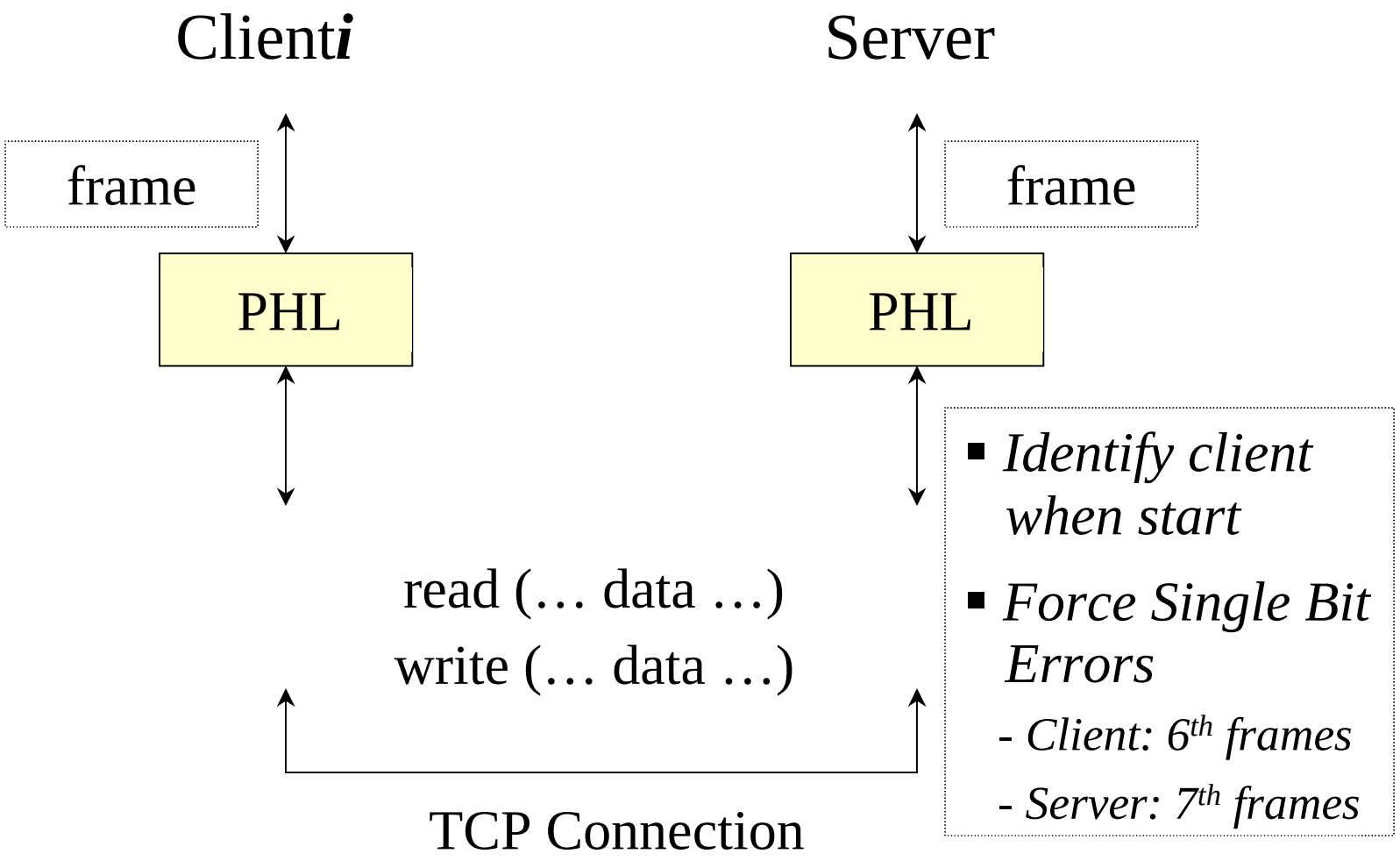


Note: The max_size of a packet is 60 bytes

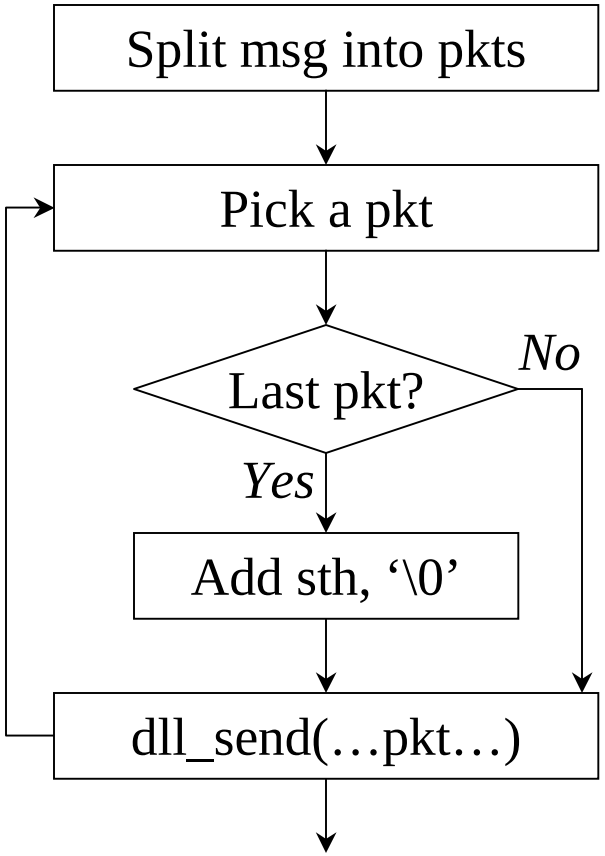
The network layer will send packets until blocked by the Data Link Layer



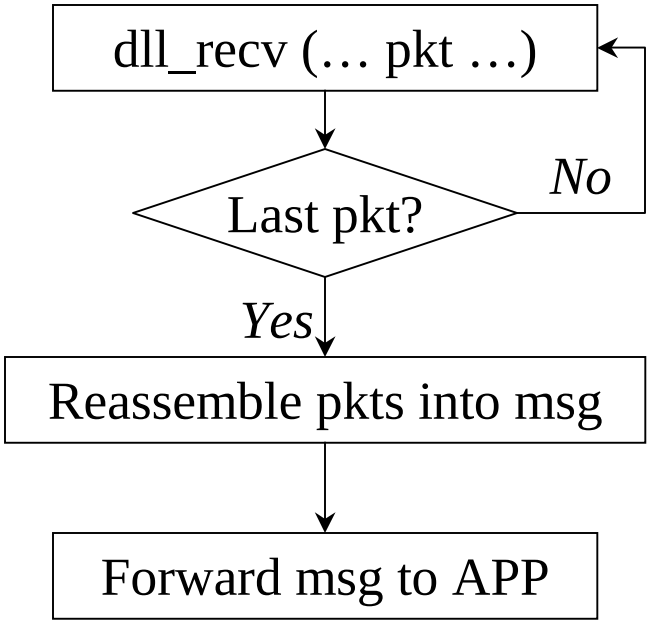
Note: The max_size of a frame payload is 32 bytes


Client*i*
Server
frame
frame
PHL
PHL
read (… data …)
write (… data …)
TCP Connection
Identify client when start
Force Single Bit Errors
- Client: 6th frames
- Server: 7th frames

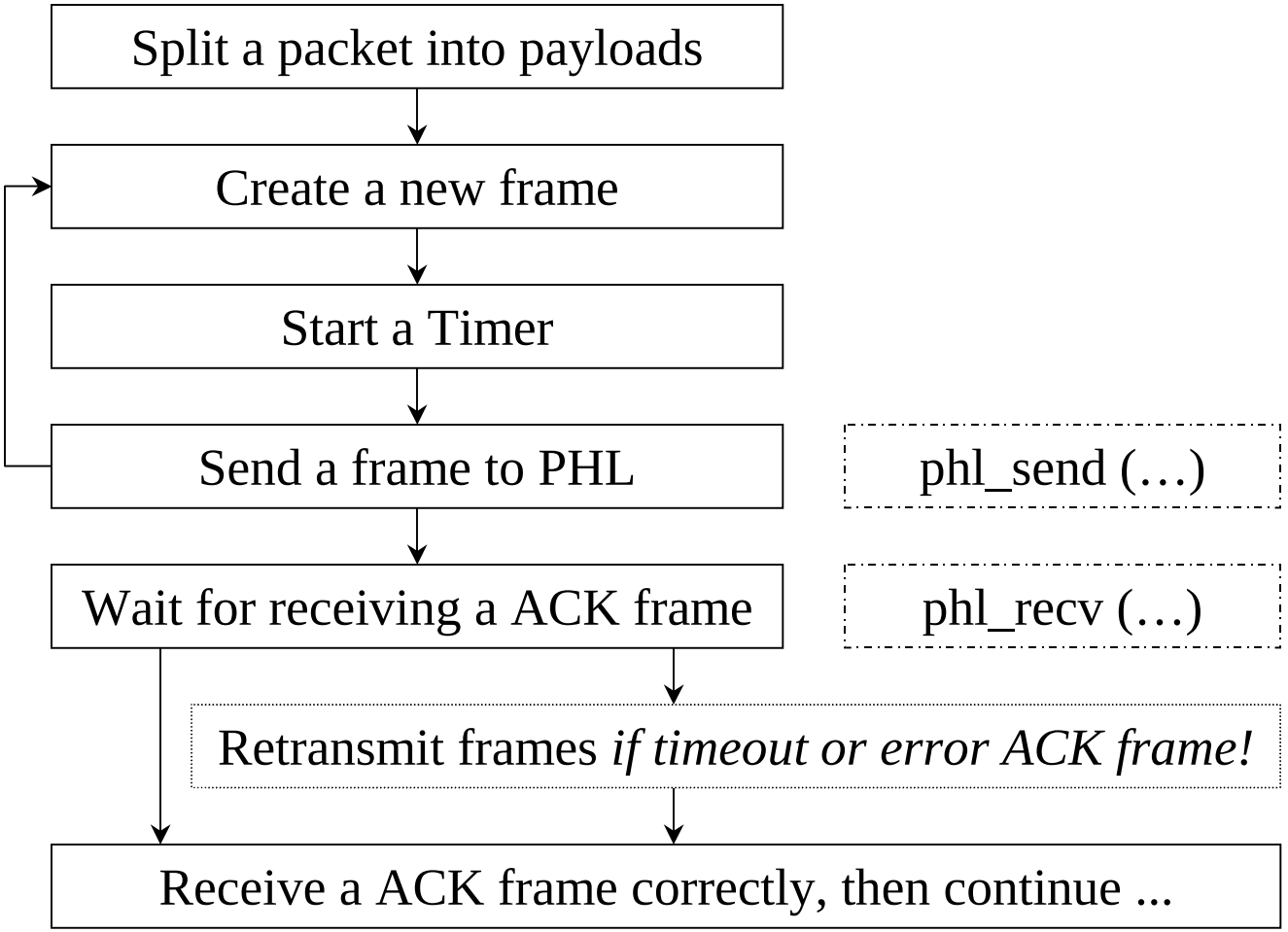
nwl_send (… msg …)



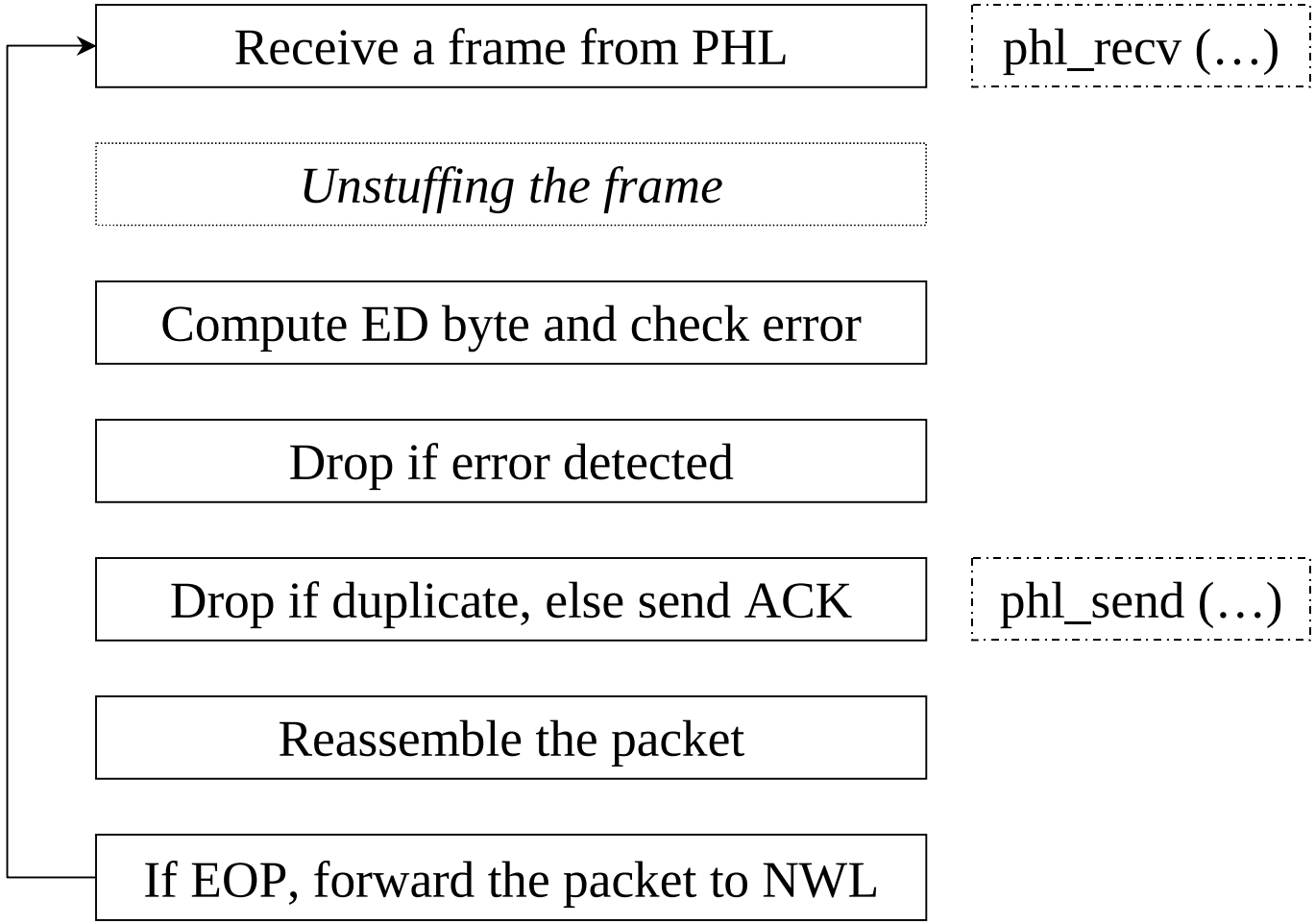
nwl_recv (… msg …)



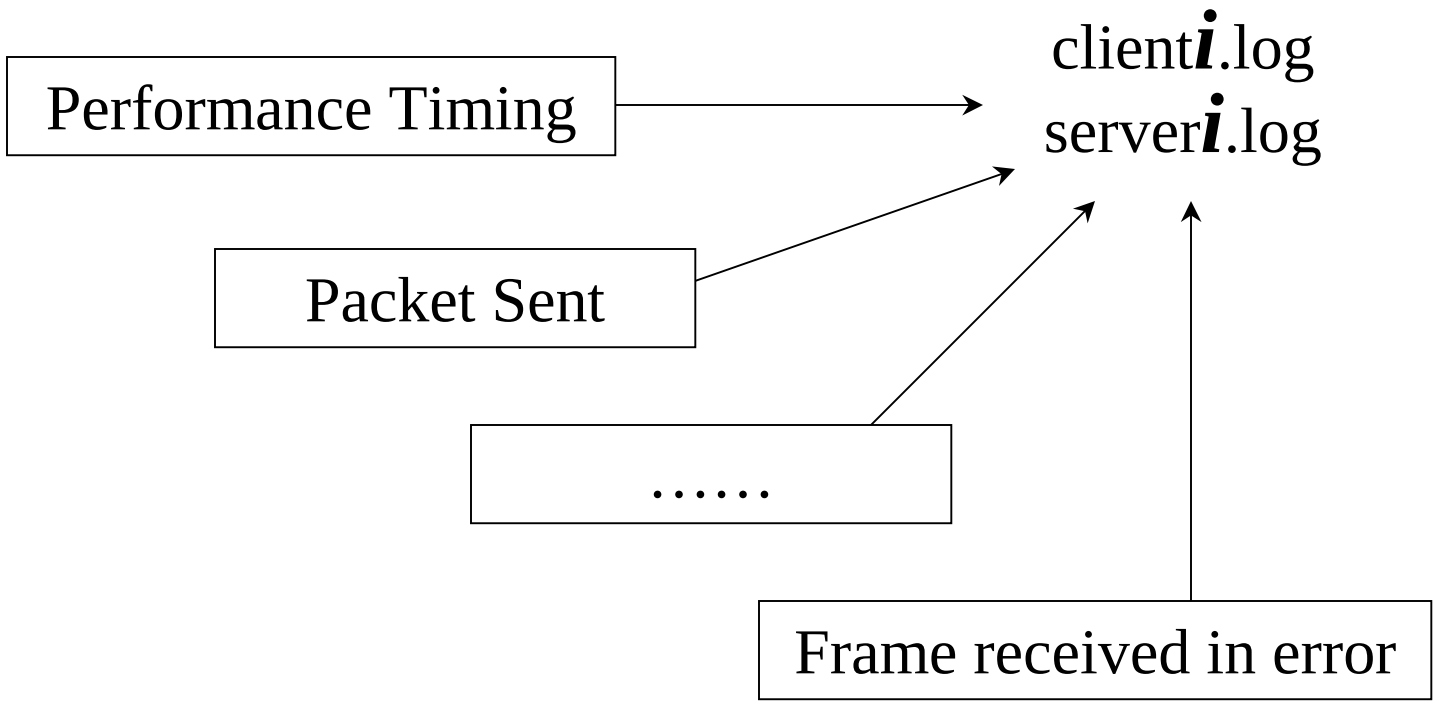
dll_send (… pkt …)


Split a packet into payloads
Create a new frame
Start a Timer
Send a frame to PHL
phl_send (…)
Wait for receiving a ACK frame
phl_recv (…)
Retransmit frames *if timeout or error ACK frame!*
Receive a ACK frame correctly, then continue ...

dll_recv (… pkt …)


Receive a frame from PHL
phl_recv (…)
Unstuffing the frame
Compute ED byte and check error
Drop if error detected
Drop if duplicate, else send ACK
phl_send (…)
Reassemble the packet
If EOP, forward the packet to NWL

Log Significant Events



Project Tips

- Sliding Window Protocol: Go-Back-N (N>3)
  - Try to implement Go-Back-1 first
  - Then implement Go-Back-N (multiple timers)
- Implement non-blocking dll_send()
  - See the example" (2nd next slide)
  - Multi-process programming (maybe)
  - Multi-thread programming (won't suggest)
- Maybe easier to merge PHL and DLL

Concurrent Server Example

```
pid_t pid;
int listenfd, connfd;
listenfd = socket( ... );
/* fill in sockaddr_in{ } with server's well-known port */
bind (listenfd, ...);
listen (listenfd, LISTENQ);
while(1){
    connfd = accept(listenfd, ... );  /* probably blocks */
    if(( pid = fork()) == 0){
        close(listenfd);  /* child closes listening socket */
        doit(connfd);     /* process the request */
        close(connfd);    /* done with this client */
        exit(0);
    }
    close(connfd);    /* parent closes connected socket */
}
```

Relative Timer Example

```
/* example for start_timer, stop_timer, send_packet */
/* you SHOULD modify this to work for project 3, this is just a TIMER EXAMPLE */
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <sys/time.h>
#include <sys/timers.h>
#include <sys/select.h>
#include <sys/types.h>
#include <errno.h>
#define TIMER_RELATIVE 0
#define MAX_SEQ 3
extern int errno;
typedef unsigned int seq_nr;
typedef enum {frame_arrival, cksum_err, timeout, network_layer_ready} event_type;
timer_t timer_id[MAX_SEQ];
```

```
void timeout() {
    printf("time out!\n");
}
void start_timer(seq_nr frame_nr) {
    struct itimerspec time_value;
    signal(SIGALRM, timeout);
    time_value.it_value.tv_sec = 1;  /* timeout value */
    time_value.it_value.tv_nsec = 0;
    time_value.it_interval.tv_sec = 0;  /* timer goes off just once */
    time_value.it_interval.tv_nsec = 0;
    timer_create(CLOCK_REALTIME, NULL, &timer_id[frame_nr]);  /* create timer */
    timer_settime(timer_id[frame_nr], TIMER_RELATIVE, &time_value, NULL);  /* set timer */
}
void stop_timer(seq_nr ack_expected) {
    timer_delete(timer_id[ack_expected]);
}
void send_packet(packet *p) {
    fd_set readfds;
    int sockfd;
```

```
while(packet hasn't been finished sending) {
    /* send frame if we can */
    while(there's place left in sliding window) {
      /* construct a frame from the packet */
      /* send this frame; start timer; update sliding window size */
    }
    /* check data from physical layer */
    FD_ZERO(&readfds);
    FD_SET(sockfd, &readfds);
    if (select(sockfd+1, &readfds, (fd_set *)NULL, (fd_set *)NULL, (struct timeval*)NULL) < 0) {
      if (errno == EINTR) {  /* receive timeout signal */
        /* timeout handler should have resent all the frames that haven't been acknowledged */
        continue;
      }  else {
        perror("select error");  /* select error */
        exit(1);
      }
    }
```

```
            if (FD_ISSET(sockfd, &readfds)) {  /* a frame come from socket */
            /* read a frame from the socket */
            if (cksum() == FALSE) {  /* error check */
               continue;  /* do nothing, wait for timer time out */
            }
            else {
               /* check to see if this frame is a data or ACK frame, and do corresponding processing */
               continue;
            }
         }
      }
   }
```

More Issues

- The dll_recv() behavior
  - Require received frame buffer
- How to terminate client process:
  - When the client gets the response to the quite message
  - A “clean” way to terminate the server child process?