

CSE 123 : Computer Networks

Winter 2022 : Discussion-1

P1 TA : Shreyas Anantha Ramaprasad

Acknowledgements - Palash Chauhan and Zesen “Jason” Zhang

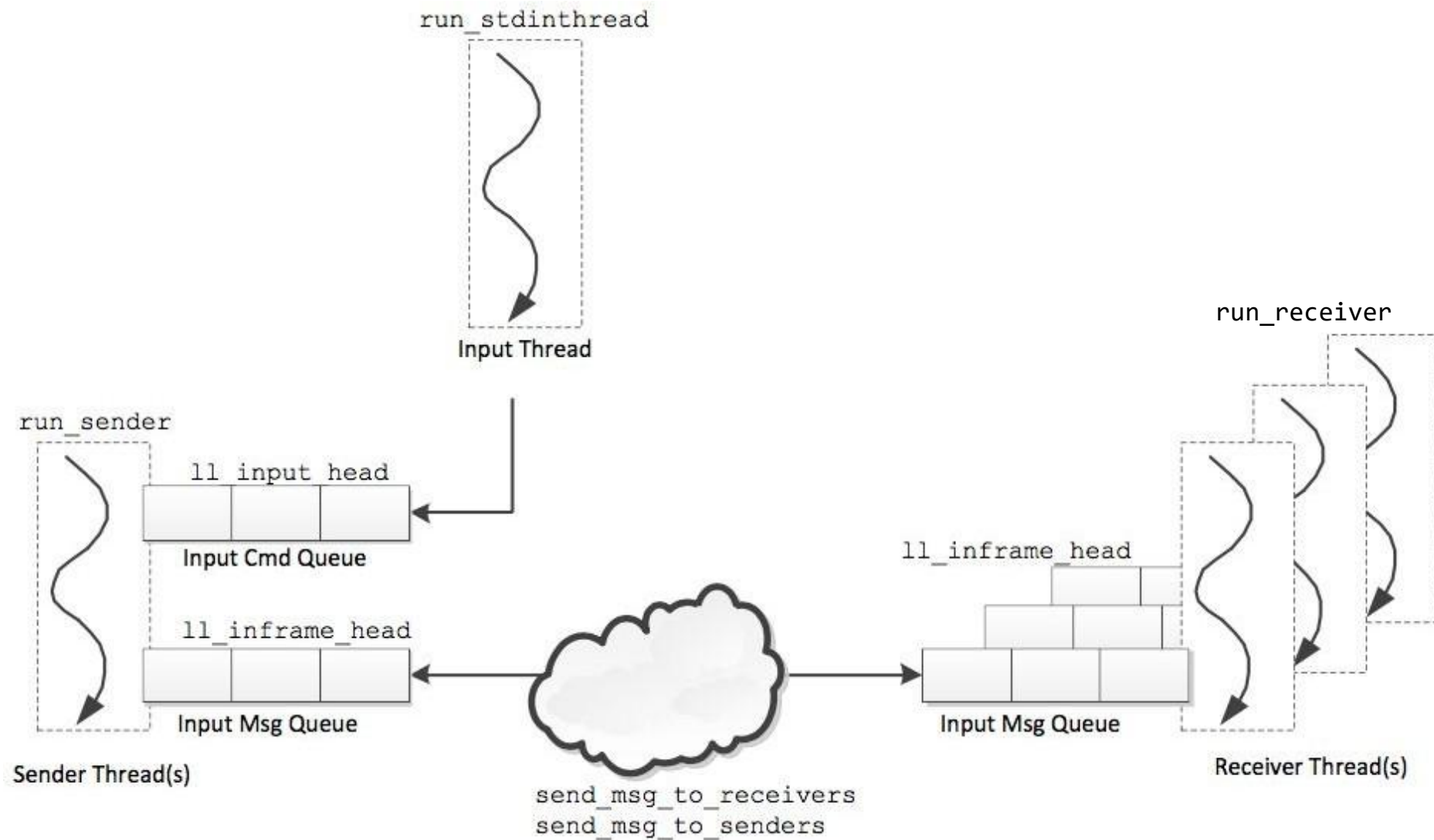
Project 1a - Framing and Retransmission

- Goal : Implementing RELIABLE communication between two or more hosts
- Hosts are implemented as threads, and we will be simulating the network link between these hosts (threads)
- Two types of hosts : senders and receivers
- Messages can and will be dropped in flight
- Implement framing and retransmission for reliable communication

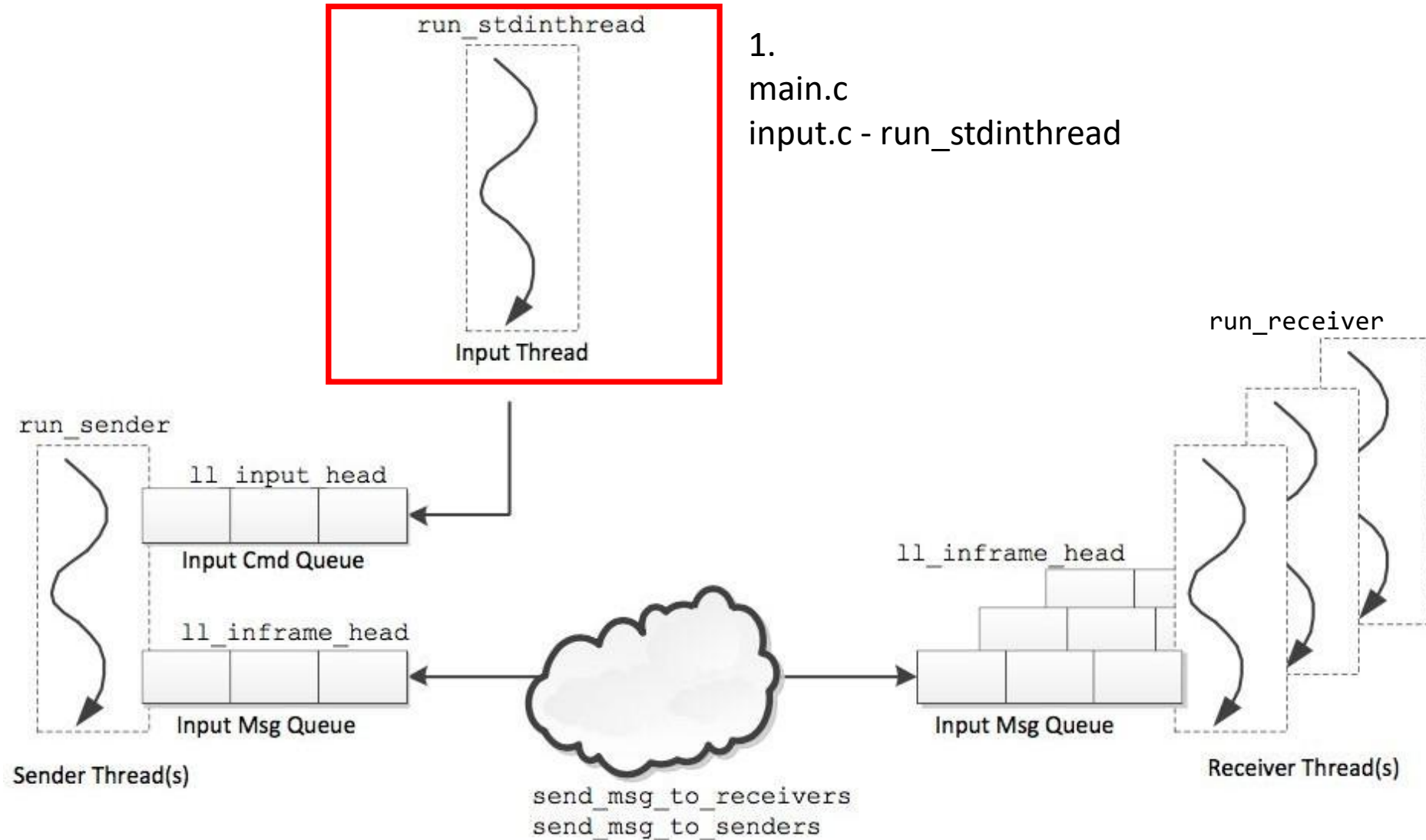
Getting Started

- Accept the invitation (link is on Canvas) to create your copy of the Project 1 base code
- Clone your Project1 repository using : `git clone YOUR-URL`
- Compile the skeleton code by typing: `make`
- Check the options to run the executable named `tritontalk` by using the command `./tritontalk -h`
- Make changes to the code, then commit and push
 - `git add *`
 - `git commit -m "Your commit message"`
 - `git push origin master`

Code Structure

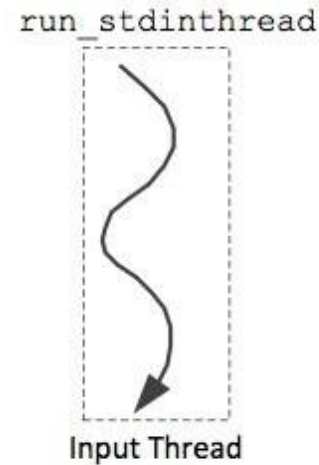
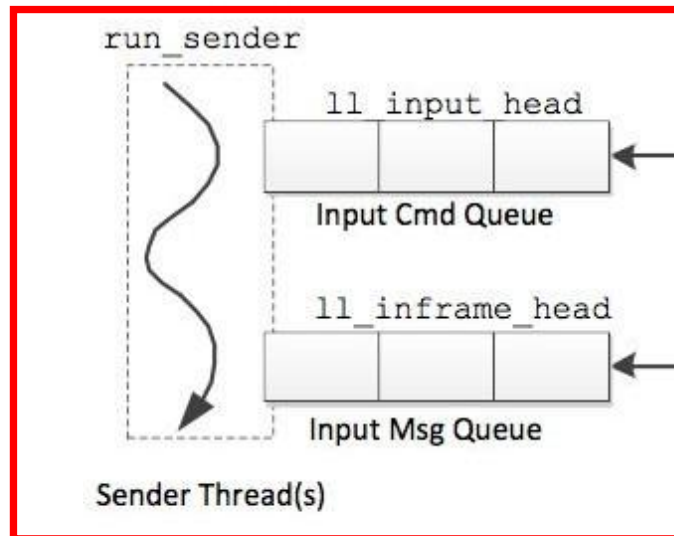


Code Structure - Initialization

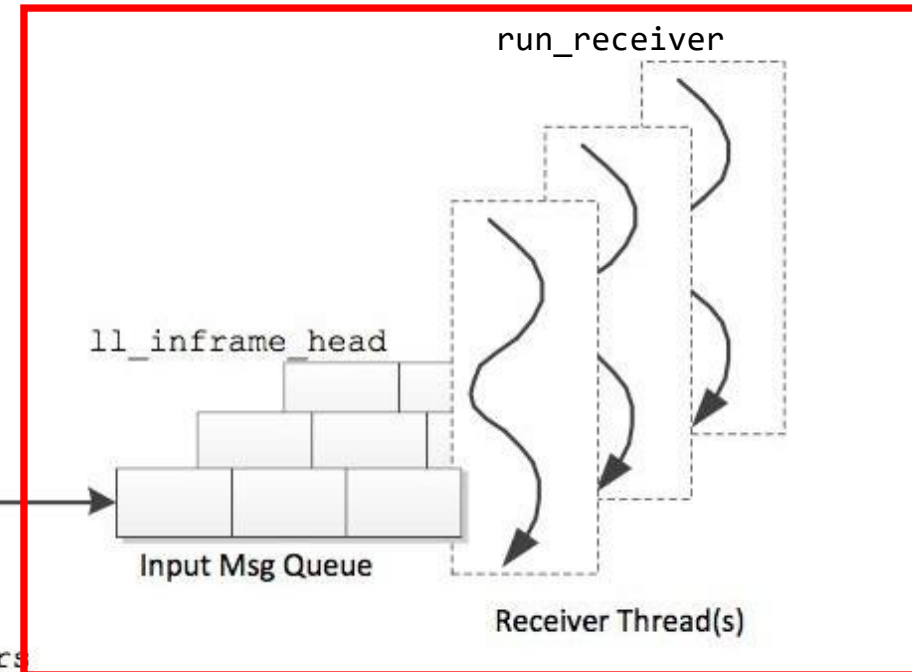


Code Structure - Initialization

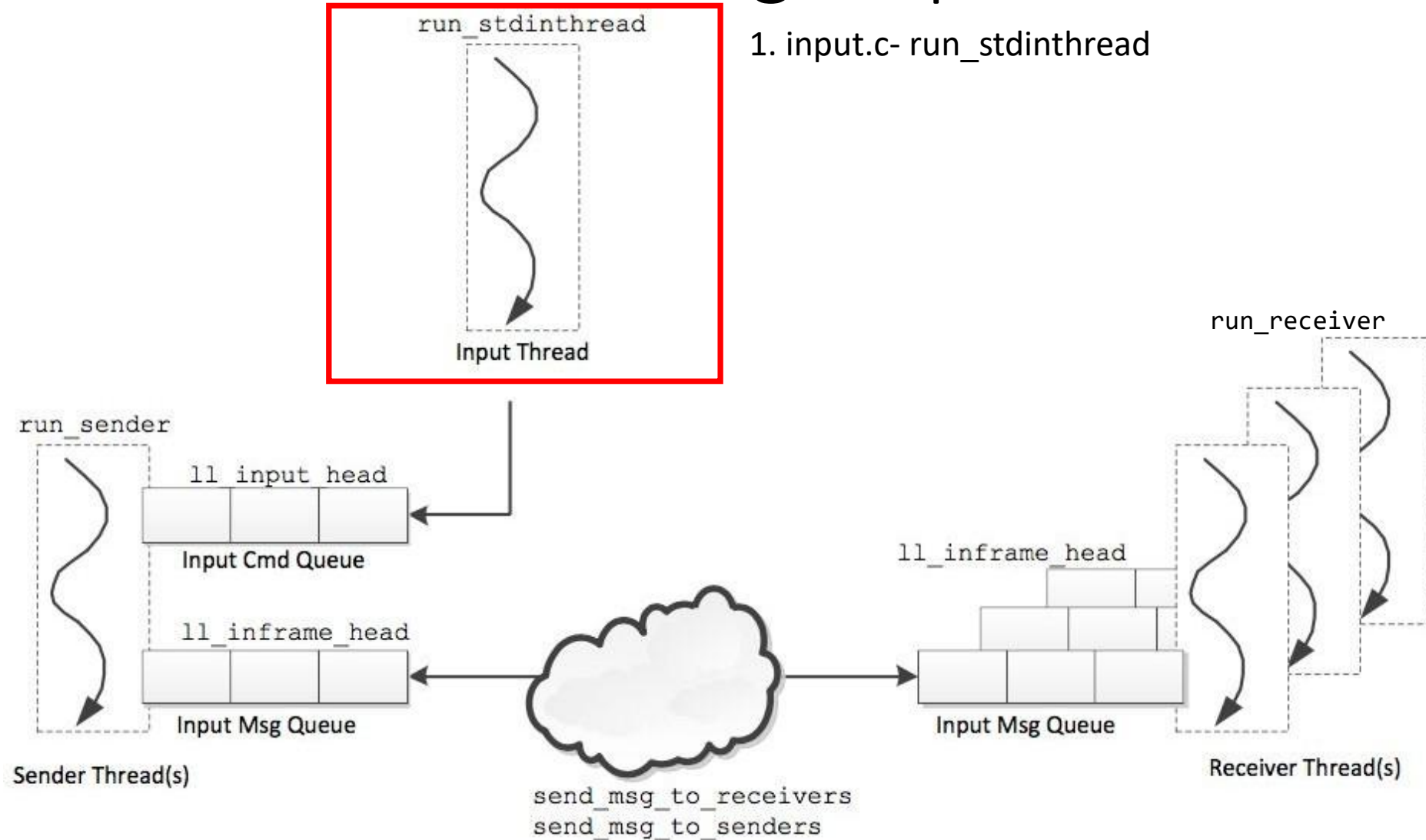
2. sender.c - init_sender



3. receiver.c - init_receiver



Code Structure - Message Input



Code Structure - Communication

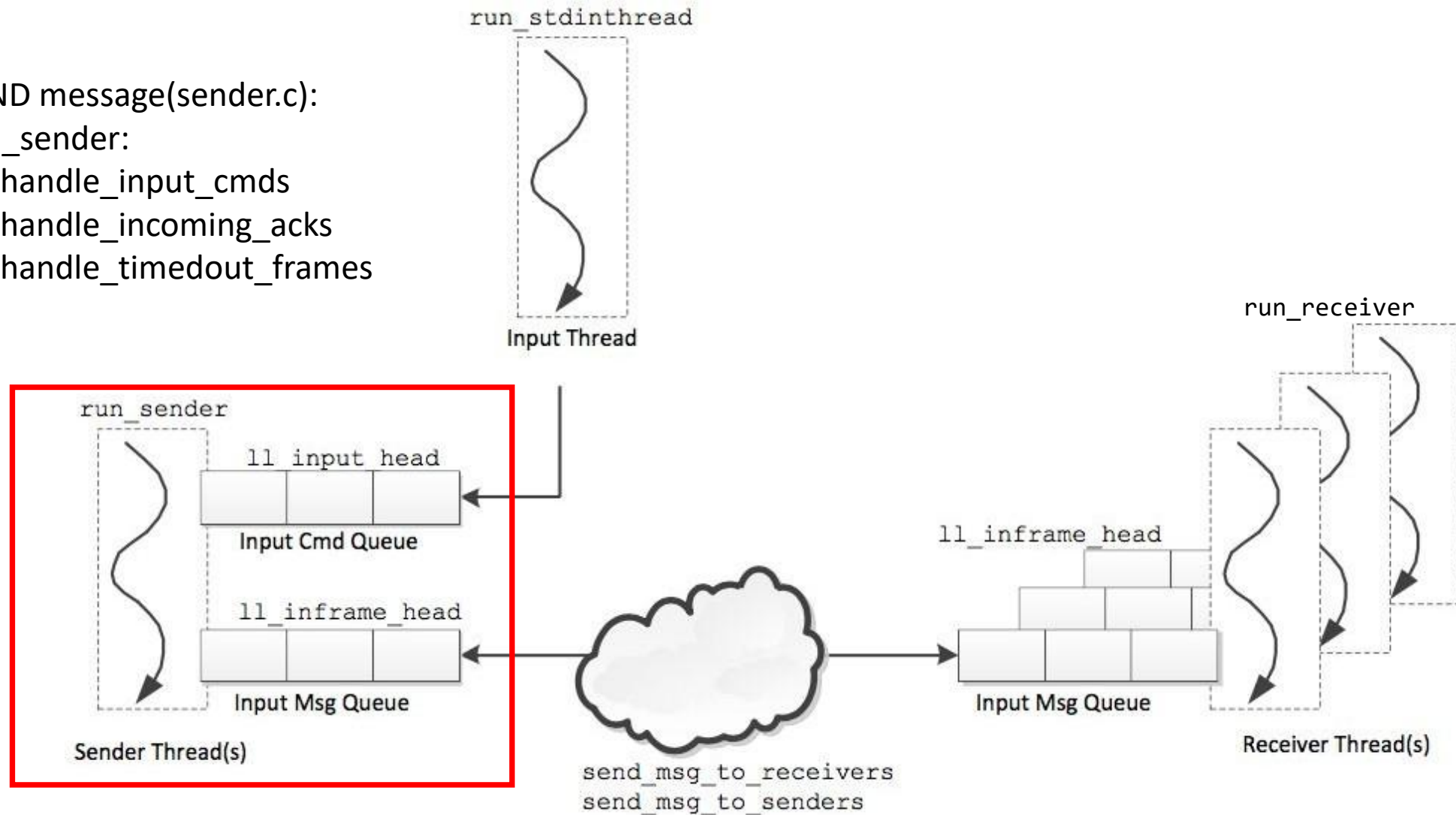
SEND message(sender.c):

run_sender:

handle_input_cmds

handle_incoming_acks

handle_timedout_frames



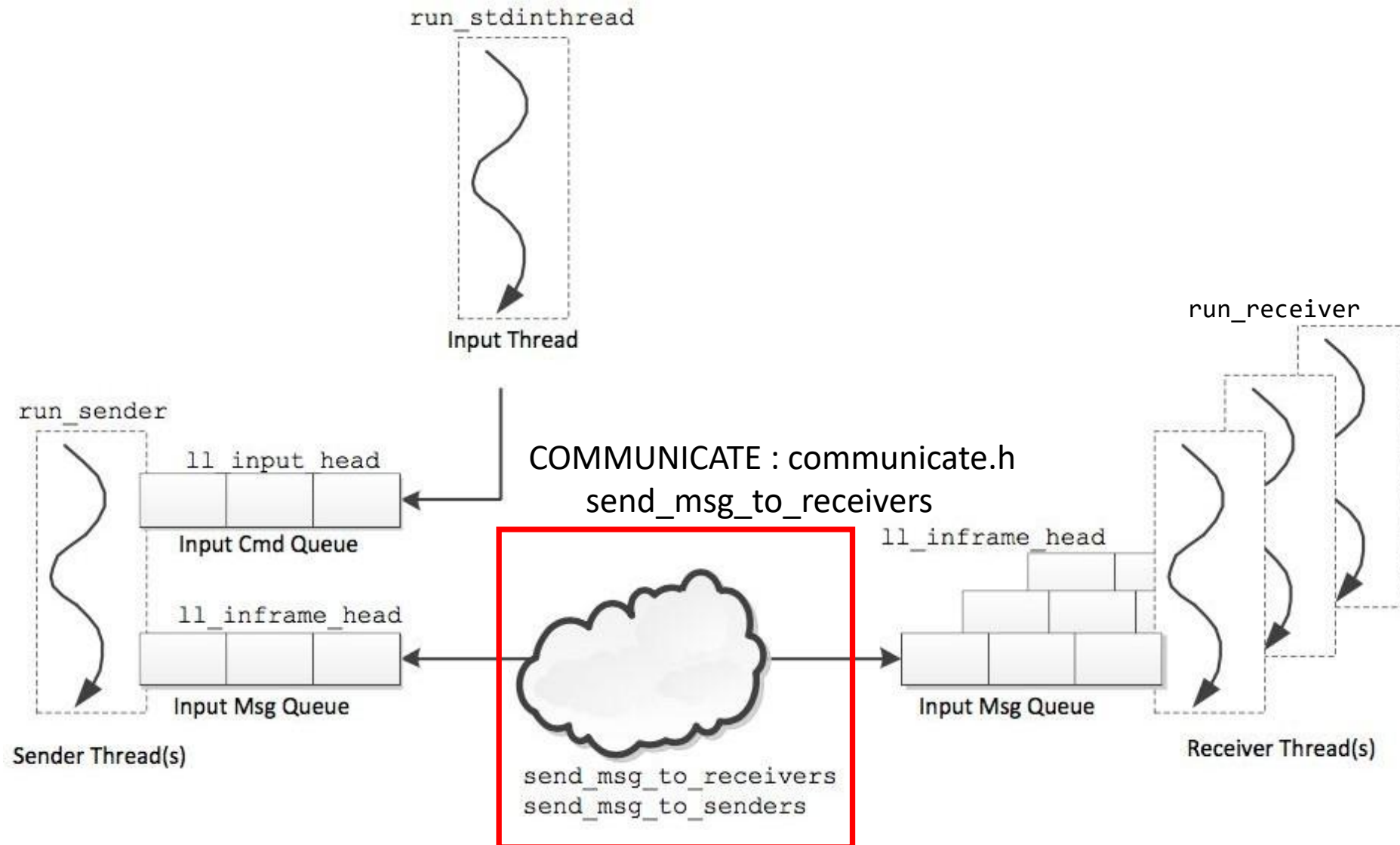
Sender_t (common.h)

```
struct Sender_t {  
    // DO NOT CHANGE:  
    // 1) buffer_mutex  
    // 2) buffer_cv  
    // 3) input_cmdlist_head  
  
    // 4) input_framelist_head  
  
    // 5) send_id  
    pthread_mutex_t buffer_mutex;  
    pthread_cond_t buffer_cv;  
    LLnode* input_cmdlist_head;  
    LLnode* input_framelist_head;  
    int send_id;  
};
```

run_sender (sender.c)

1. Determine the next time the thread should wake up
2. Grab the mutex protecting the input_cmd/inframe queues
3. Dequeues messages from the input queue and adds them to the outgoing_frames list
4. Releases the lock
5. Sends out the messages

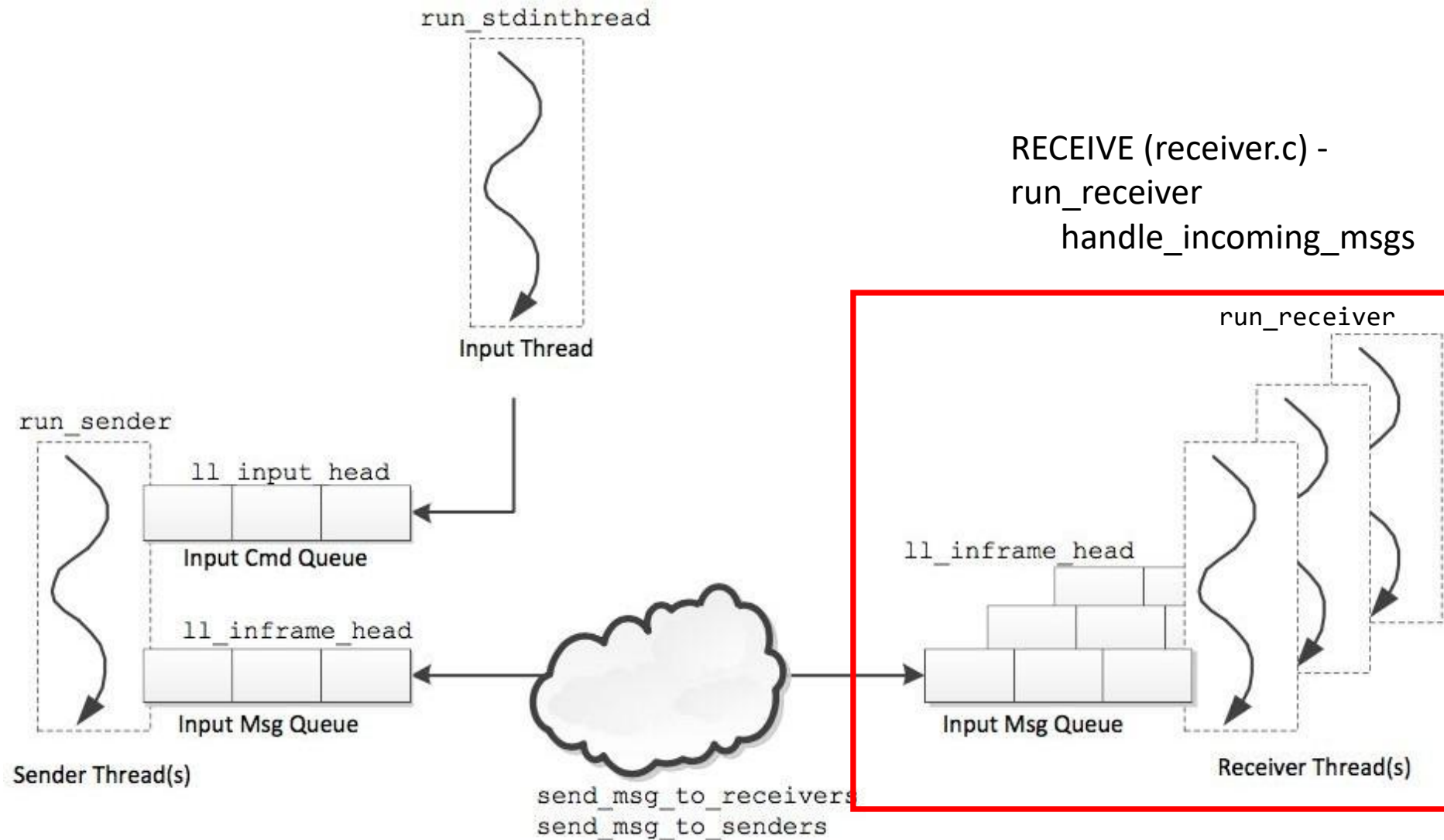
Code Structure - Communication



communicate.c

- Implements transporting of messages between the sender and receiver threads.
- Two main methods:
 - `send_msg_to_receivers`
 - `Send_msg_to_senders`
- Supports buffers of at most 64B
- Communication is **broadcast** based!
- Messages can be dropped!
- Messages can be corrupted! (**not used in Project 1a**)

Code Structure - Communication



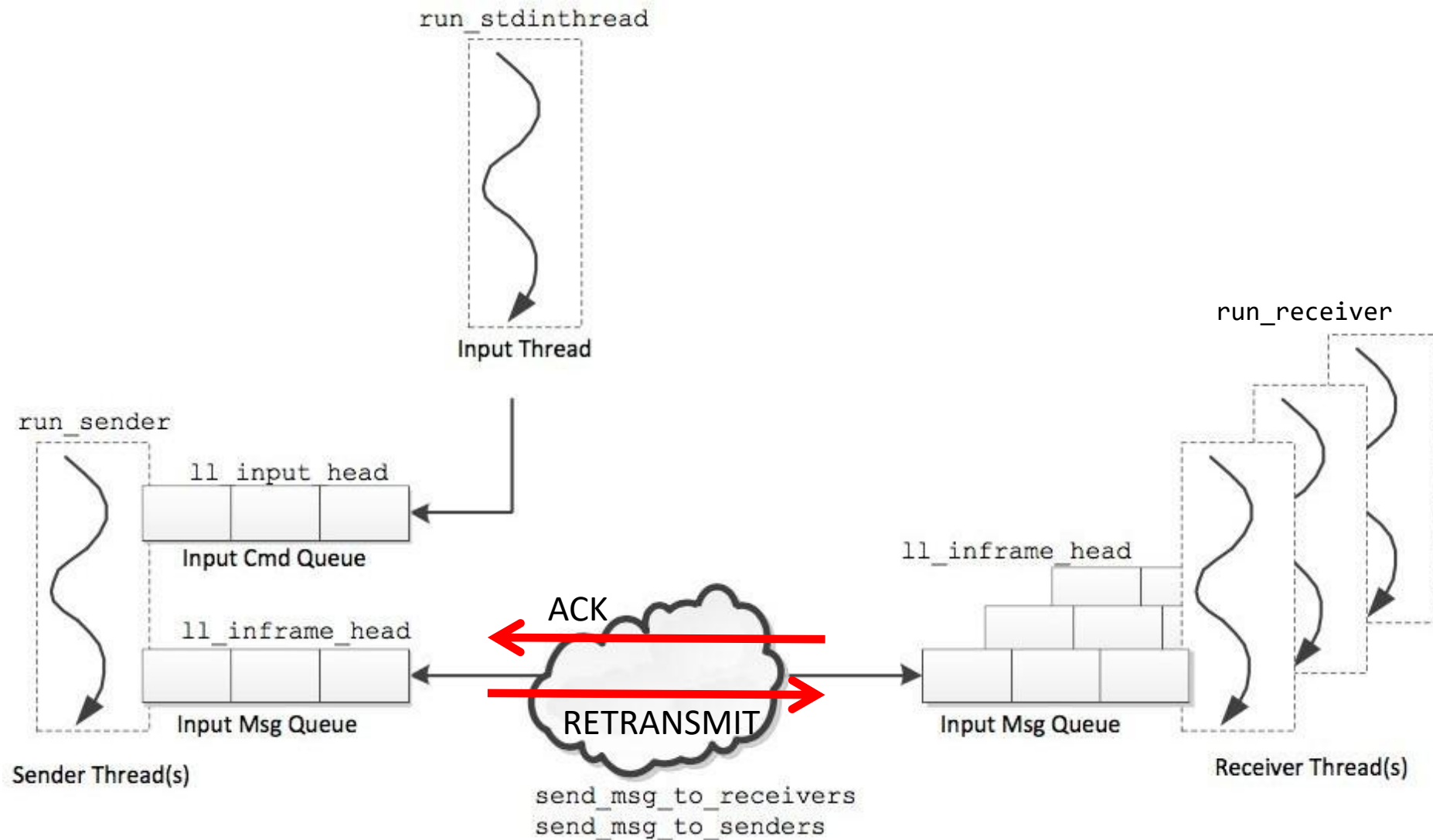
Receiver_t (common.h)

```
struct Receiver_t {  
    // DO NOT CHANGE:  
    // 1) buffer_mutex  
    // 2) buffer_cv  
    // 3) input_framelist_head  
  
    // 4) recv_id  
  
    pthread_mutex_t buffer_mutex;  
    pthread_cond_t buffer_cv;  
    LLnode* input_framelist_head;  
  
    int recv_id;  
};
```

run_receiver (receiver.c)

1. Determine the next time the thread should wake up if there is nothing in the incoming queue(s)
2. Grab the mutex protecting the input_msg queue
3. Dequeues messages from the input_msg queue and prints them
4. Releases the lock
5. Sends out any outgoing messages

Code Structure - Acknowledge & Retransmission



Code Structure

● input.c

- Handles messages inputted by the user
- msg 0 1 hello world

● main.c

- Handles command line options
- Starts the sender threads, receiver threads, stdin thread

● common.h

- Houses commonly used data structures among the various source files

● util.c

- Contains utility functions, namely, all of those for the provided linked list implementation

Tasks

1. Framing

- Divide messages into frames. Helps with error detection and retransmission
- Maximum frame size defined by `MAX_FRAME_SIZE` in `common.h` = 64B
- Create a header format

2. Acknowledgements

3. Retransmission using timeouts

4. Stop-and-Wait scheme

Tasks

1. Framing
2. **Acknowledgements**
 - a. **Receiver should respond to the sender that it has received the corresponding message.**
3. Retransmission using timeouts
4. Stop-and-Wait scheme

Tasks

1. Framing
2. Acknowledgements
- 3. Retransmission using timeouts**
 - a. If a message is lost in transit, your senders should retransmit it after waiting more than 0.085 seconds but less than 0.1 seconds. We recommend 0.09 seconds.
4. Stop-and-Wait scheme

Tasks

1. Framing
2. Acknowledgements
3. Retransmission using timeouts
- 4. Stop-and-Wait scheme**
 - a. Will be discussed in class. Or read the textbook!**

Tasks

- You are responsible for modifying the sender.c, receiver.c and common.h files
- You may modify any of the other files in the repo, and add any additional files as necessary (also taking care to change the Makefile)
- However, we will be overwriting the input.h, input.c, communicate.h, and communicate.c files after you have submitted your project.
- A sender may finish handling one command from the input thread before starting on the next command, but a receiver must be able to receive messages simultaneously from different senders
- Keep pushing code to Github frequently!

Useful Links

- Threads

- <https://www.cs.cmu.edu/afs/cs/academic/class/15492-f07/www/pthreads.html>
- <https://www.geeksforgeeks.org/condition-wait-signal-multi-threading/>
- https://pubs.opengroup.org/onlinepubs/009696699/functions/pthread_cond_timedwait.html
- Note : The above links are for your reference, but you will not require to write any explicit threading code by yourself in project 1a and 1b.

- Git Fundamentals

- <https://www.earthdatascience.org/workshops/intro-version-control-git/basic-git-commands/>

- Structures in C

- https://www.tutorialspoint.com/cprogramming/c_structures.htm

For Windows Users

- The skeleton code will not work as some libraries are Linux specific
- Your code must compile and run on department machines
- Recommendation : Setup WSL2 and run your code on that
 - Step 1 : Setup WSL2 by following <https://docs.microsoft.com/en-us/windows/wsl/install>
 - Step 2 : Once the linux user is setup, execute (for Ubuntu):
 - `sudo apt install make`
 - `sudo apt install build-essential`
 - Step 3 : Edit your code and compile it like you would on regular Linux
- Tip: Editing code on vim or nano can be tedious. You can use an IDE on Windows like VSCode and use a WSL plugin
 - <https://code.visualstudio.com/docs/remote/wsl-tutorial>