Programming Assignment #3

The main problem

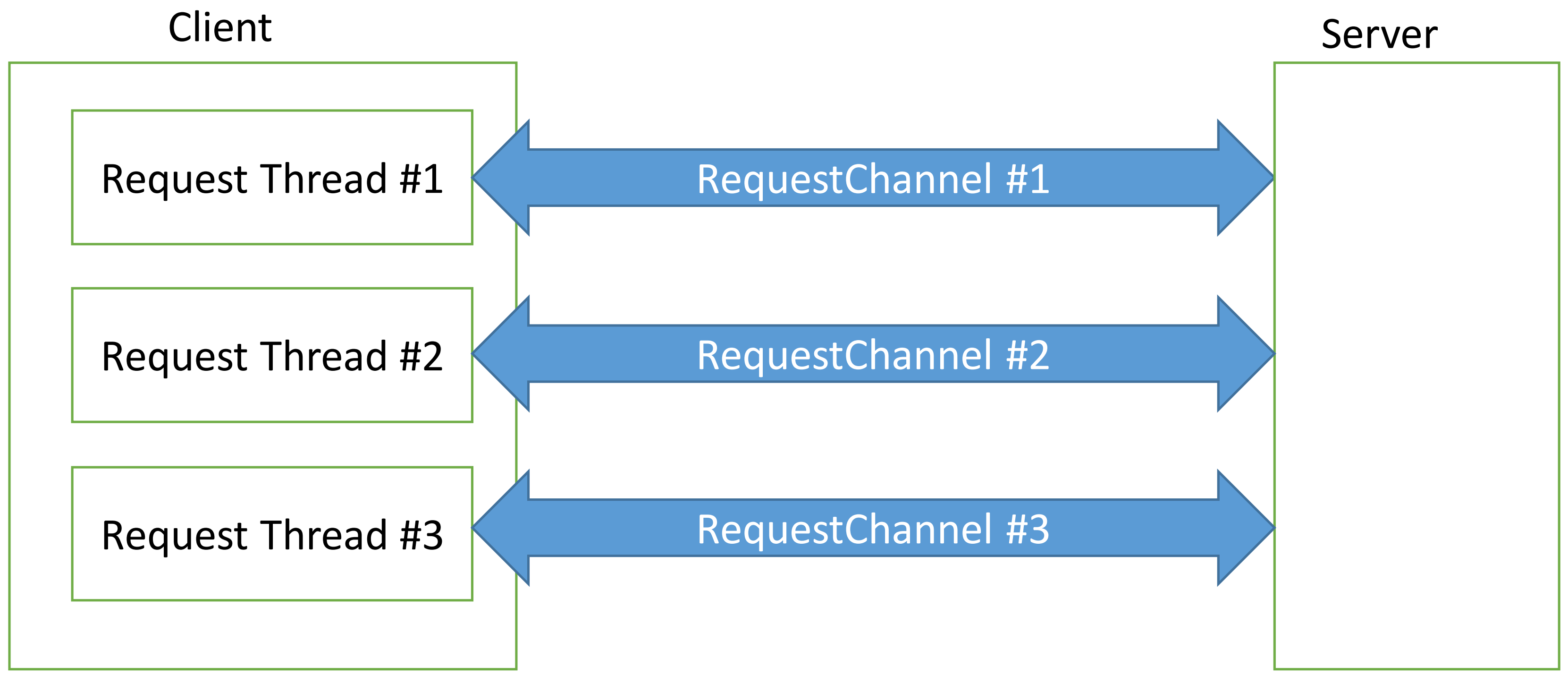
- Turn a sequential piece of code into a multithreaded one and handling/solving **race condition** resulting from that
- You are to collect patient vitals for 3 persons
  - John Smith, Jane Smith and Joe Smith
- There is a client/server architecture here
  - They reside in separate processes
  - The client forks() a child process and runs the server in it (see the given code)
  - No need to modify the server (named dataserver.cpp) for PA3
- Since we have 2 different processes, we need some sort of Inter-Process Communication (IPC) mechanism
  - We are given a **`RequestChannel`** class that encapsulates this communication
  - No need to modify this class for PA3

The Given Code

- The started code takes 2 arguments: `n` and `w`
  - `n` is the # of requests/person
  - `w` is the # of threads to be used
- Then, it sends n reqs/person to the server in the client.cpp main() and updates a histogram
- We have to turn this thing into a multithreaded/parallel version
- First Take:
  - Make 3 threads, each thread would send n requests in parallel
  - Problem #1: Limited parallelism, cannot scale speed beyond 3 times
  - Problem #2: The RequestChannel is not thread safe
    - Can solve #2 by making 3 new isntances of RequestChannel (see client.cpp for example)

First Take contd.

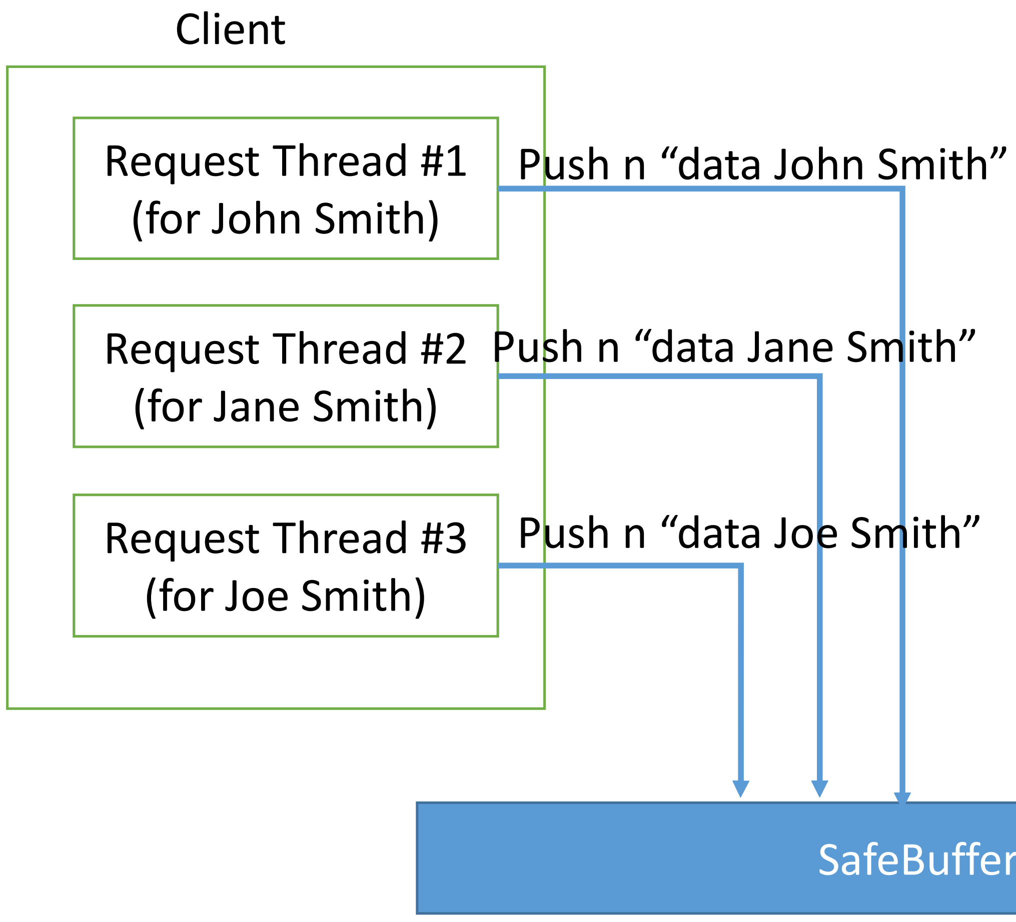
- This works, but limited to maximum 3x speed up



How to Increase Parallelism??

- We want to run more requests/sec
- One way is the following:
  - Instead of sending the requests directly to the server, store them in a buffer
  - Then, start lot more threads (we call worker threads) that would send requests from the buffer to the server
  - We can freely control the #of threads in this case w/o being stuck at 3

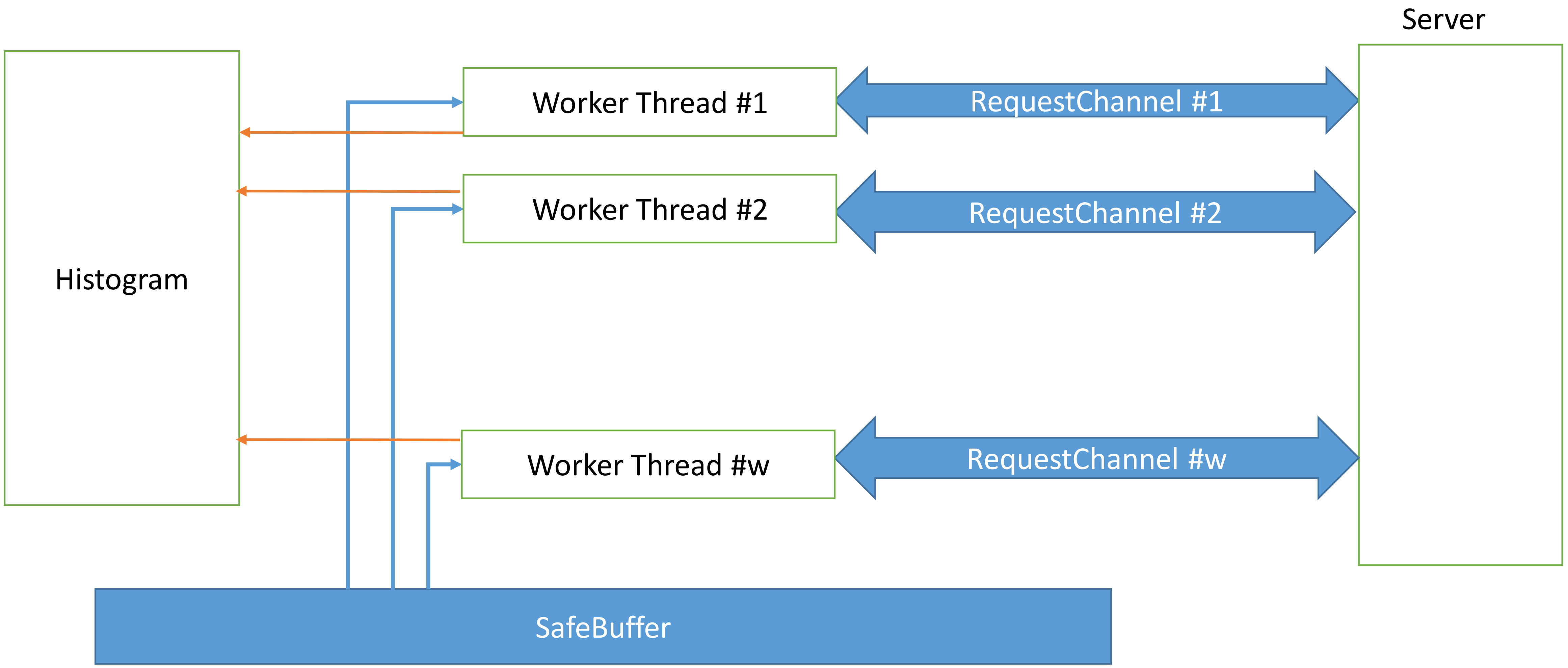
First Episode: Push All Requests in Parallel


Client
Request Thread #1 (for John Smith)
Request Thread #2 (for Jane Smith)
Request Thread #3 (for Joe Smith)
Push n "data John Smith"
Push n "data Jane Smith"
Push n "data Joe Smith"
SafeBuffer (will contain 3n requests)

2 things to do in this episode:

1. Start 3 parallel threads 1 for each person
   - Each thread will push n requests to buffer
2. Make the buffer's push function threads safe to avoid race condition

2nd Episode: Send Requests in Parallel



Things to Do in 2nd Episode

- Write a thread-safe pop() function in SafeBuffer
- Write a thread-safe update() function in Histogram
- Create w RequestChannel instances, let's call them worker channels
- Create w worker threads. Each will do:
    - Send as many requests possible (note that you cannot assume each thread will process same # of requests, the server delays each response by a random amount of time)
    - Collect each response and update histogram
    - Destroy each RequestChannel at the end
- Just make sure to create w channels from the main, not from the worker threads
    - Because the RequestChannels themselves are not thread safe

POSIX API functions needed

- pthread_create () to create a thread
- pthread_join() to wait for a thread
- pthread_mutex_lock()
- Pthread_mutex_unlock()