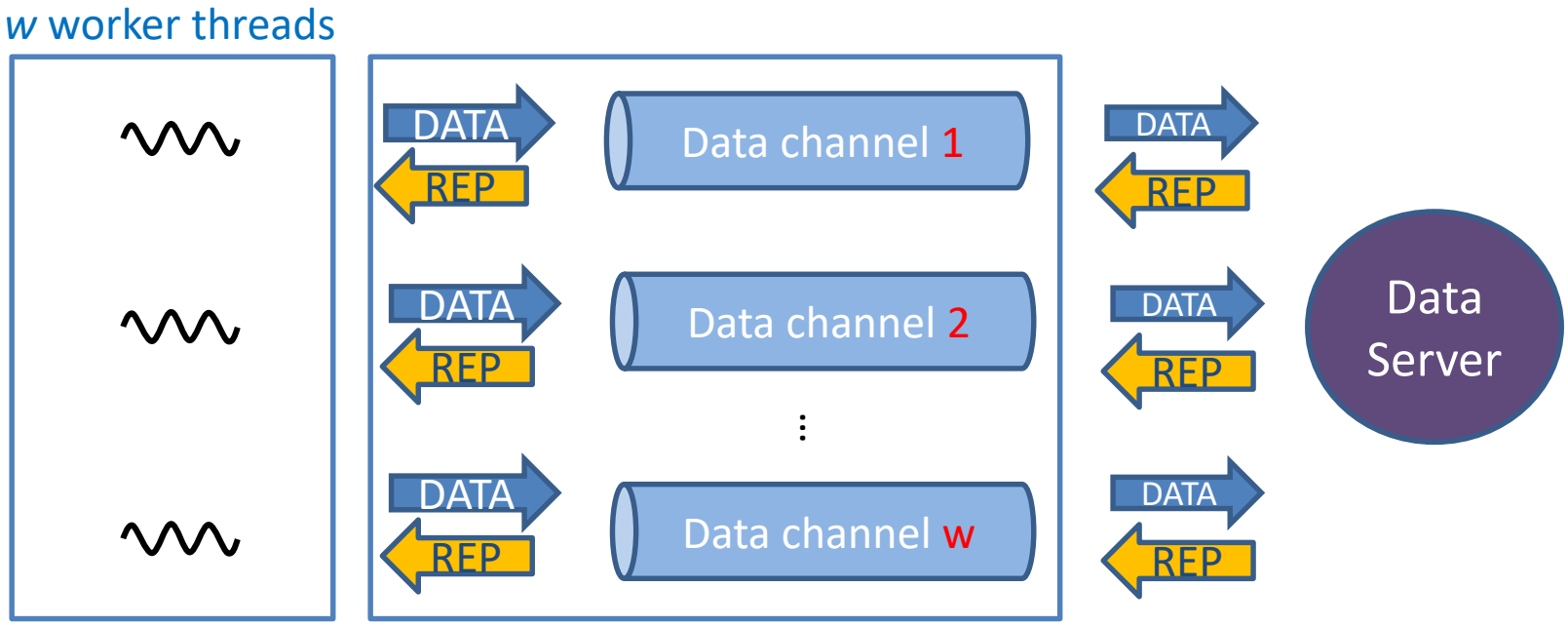
CSCE 313
PA5: I/O Multiplexing

PA4 Review



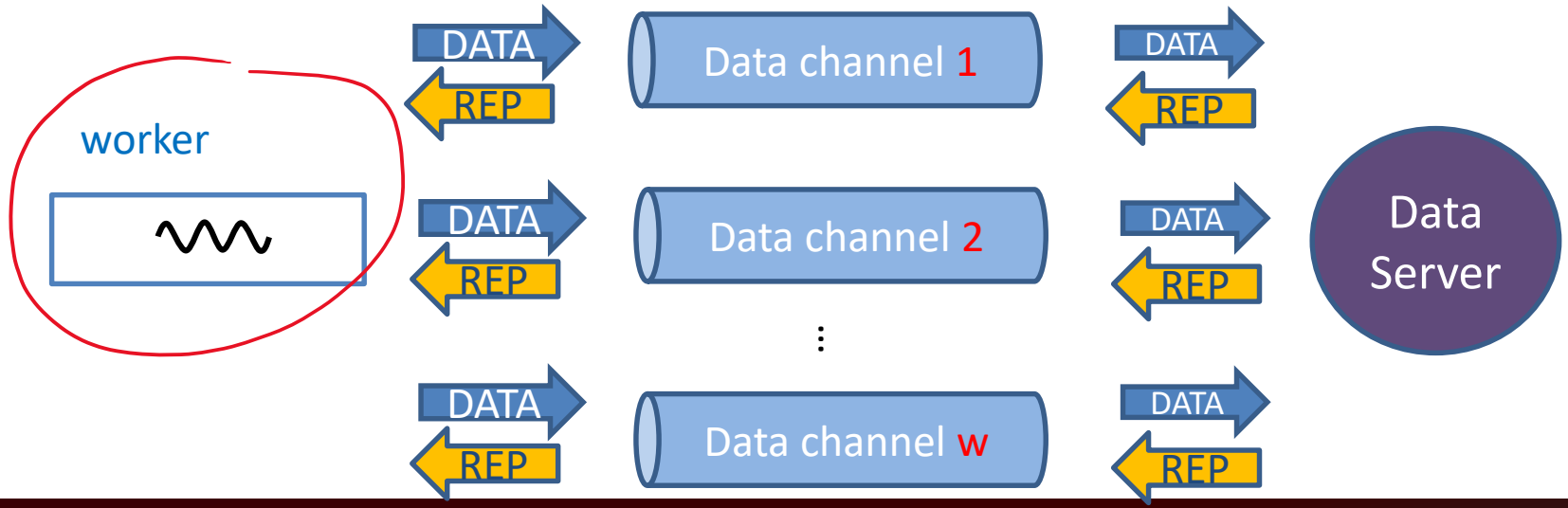
- Each worker spends most of the time waiting for server responses (inter-process delay)
  - Assume 10 sec communication delay
  - 1 worker can do 0.1 req/sec, 10 workers can do 1 req/sec

PA4 Review

- Each worker watches one data channel for response
  - Blocking I/O: Wait till data is ready
  - Low CPU utilization as most workers are waiting
- Workers are constantly switched on and off CPU
  - High context switching cost
  - Given x CPU cores, is there any reason to have more than x workers?

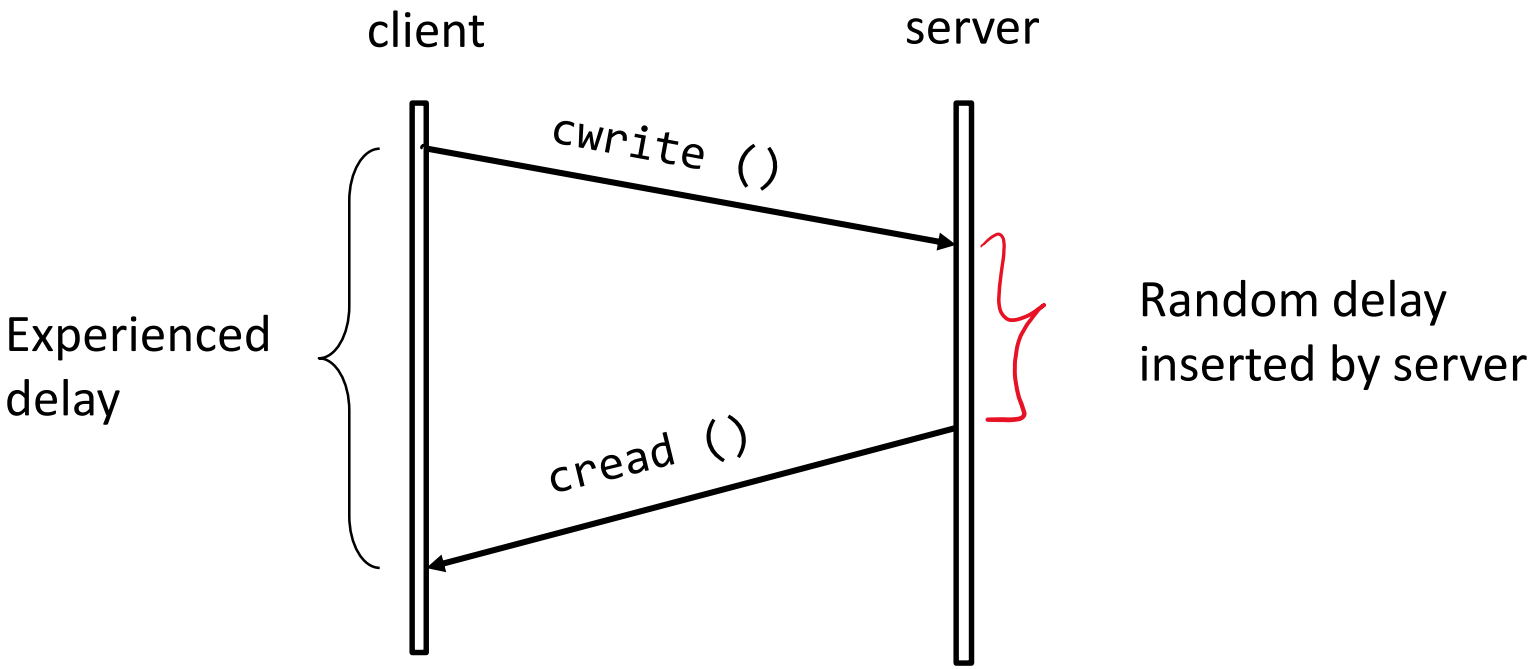
PA5

- PA5 Method: Use one single worker thread to deal with the I/O for all data channels
  - –Avoid context switching
  - –Non-blocking I/O: After issuing a request, switches to other tasks without waiting for response
  - –When data from data channel i is ready, the client detects that by waiting on epoll()

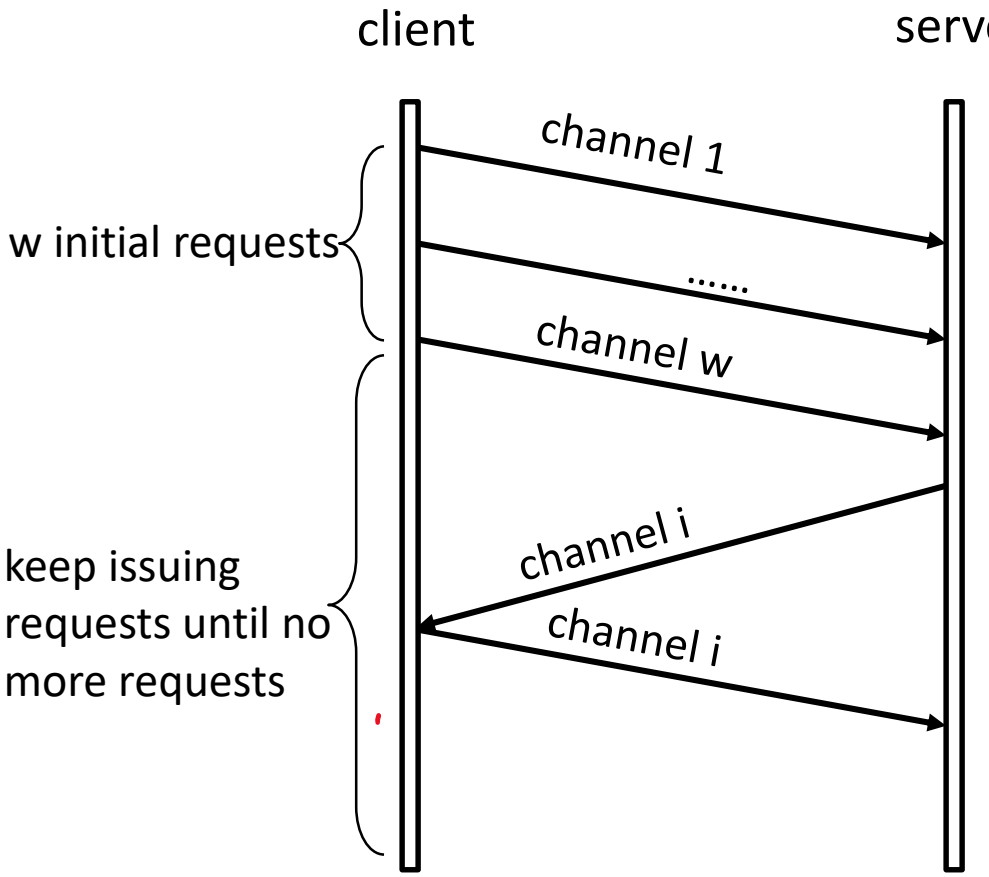


What We Did in PA4

- In PA4, we called `cwrite()` immediately followed by a `cread()` – this made a thread always until data comes back from server
- In PA5, we will change that by issuing a bunch of `cwrite()`'s and then a bunch of `cread()`'s



PA5



Priming: Issue w requests to make all worker channels busy

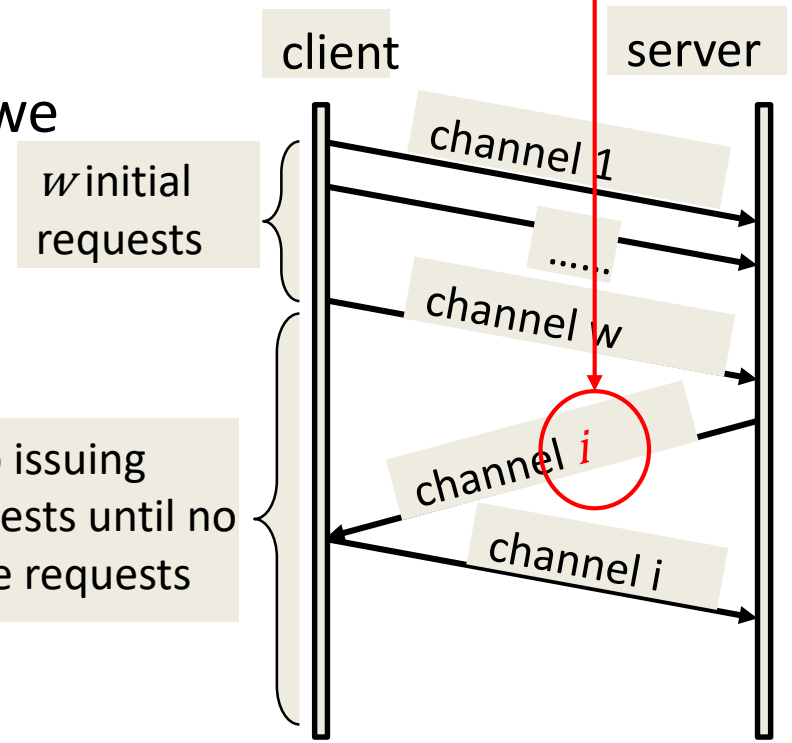
After receiving response from channel i, issue another request to the same channel and make it busy again

When receiving response from channel i, how do we know the corresponding request?

- Maintain an array of w requests, one for each channel that remembers the last request sent through

The Big Question

- How do we know the perfect order in which to receive responses?
  - You cannot expect the responses will arrive in the same order as the corresponding requests were sent
  - What is the value of *i* in the picture?
- If we wait for responses in an order different from the one they arrive in, we would be waiting unnecessarily
  - Leads to inefficiency
- We will still "block" for the channels as a group, not individually
- The function to use is `epoll()`



Epoll - Usage

- Use epoll() to listen to w worker channels
  - It is a collection of file descriptors from our request channels
  - `epoll_create1()` creates the empty list
  - `epoll_ctl()` lets us add file descriptors (i.e., FIFORequestChannel::rfd's, no need for the wfd's) from each channel
  - `epoll_wait()` waits until one or more in the list has some activity, i.e., data available to read
  - The best place to look is `epoll(7)` linux man pages

```
int epoll_create1(int flags);

int epoll_ctl(int epfd, int op, int fd,
              struct epoll_event *event);

int epoll_wait(int epfd, struct epoll_event *events,
                    int maxevents, int timeout);
```