



CIS 3990 Recitation 07

Threads

2025-10-28

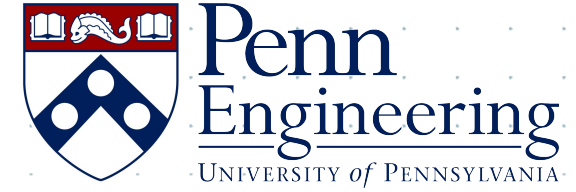
Agenda

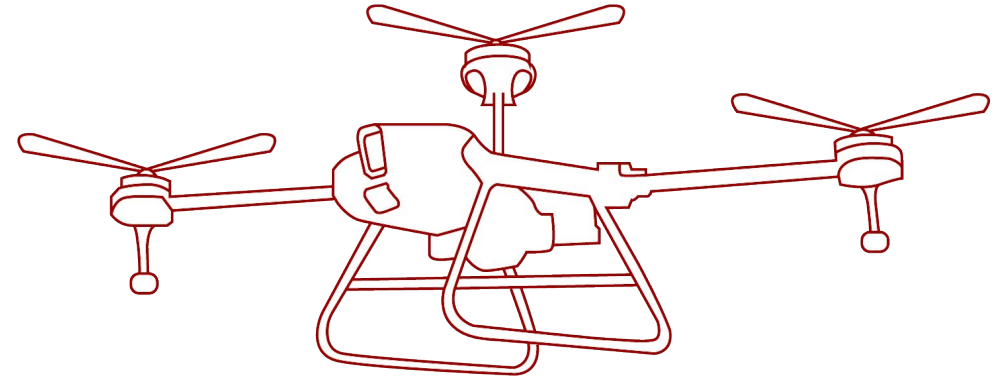
01 Course logistics

02 Threads

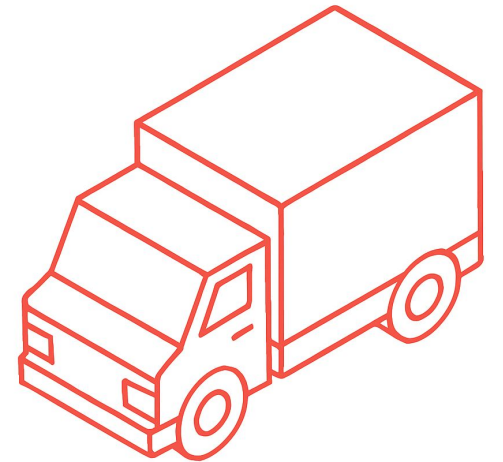
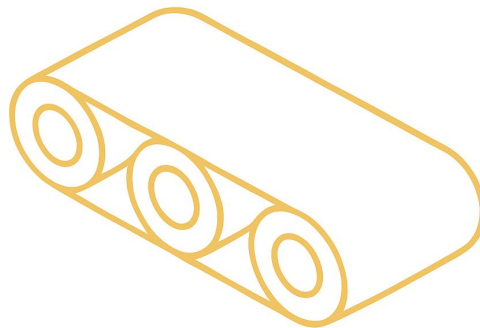
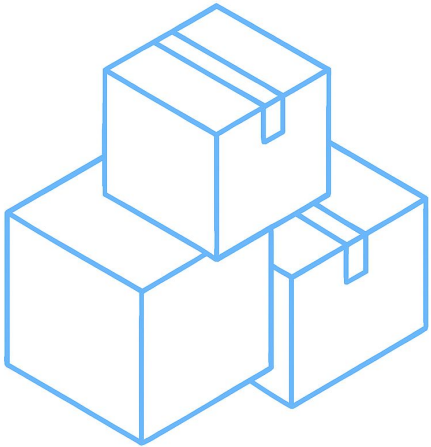
03 Locks

04 Open Q&A



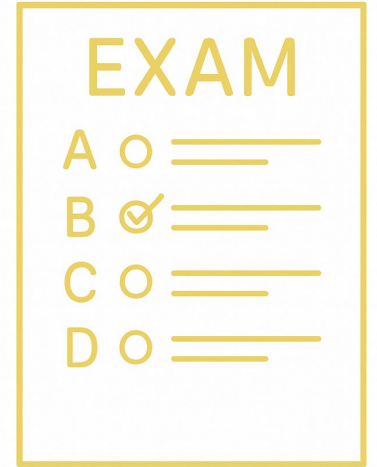


Logistics

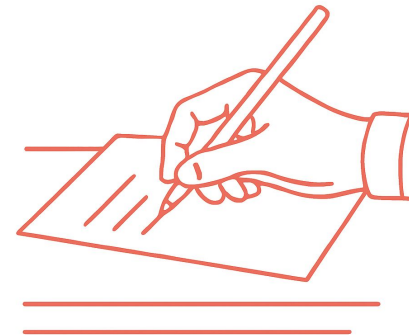
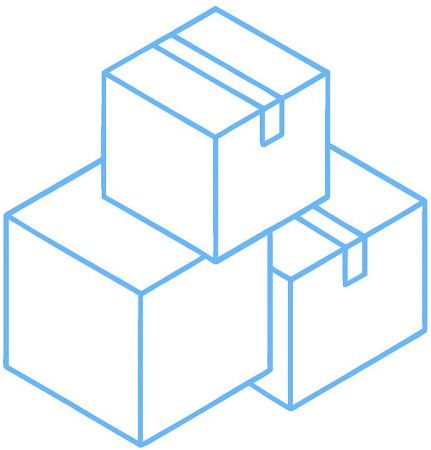


Logistics

- HW08 out - due 11:59pm on **Tuesday, November 4** - via Gradescope
- Re-opens for the last check-in are processed, let us know if forgot anything / selected the wrong date/assignment
- We're still working on midterm and hw style (re)grading
- Check-in08 due Mon, November 3



Threads



Threads

- By now, I hope the statement “threads share memory” is thoroughly imprinted in your brain
- We’ve worked with 3 “types”
 - pthreads
 - `std::thread`
 - `std::jthread`

Threads

- C (POSIX): pthreads
 - create a thread with `pthread_create()` function
 - output parameter, address of thread
 - attributes (usually NULL)
 - function to be executed
 - pointer to function arguments
 - thread function is `void*` return type
- C++: threads and jthreads
 - create a thread using a constructor
 - function to be executed
 - function arguments (if they exist)
 - thread function is `void` return type
 - need to pass in refs using `std::ref()`, not `&`

Threads

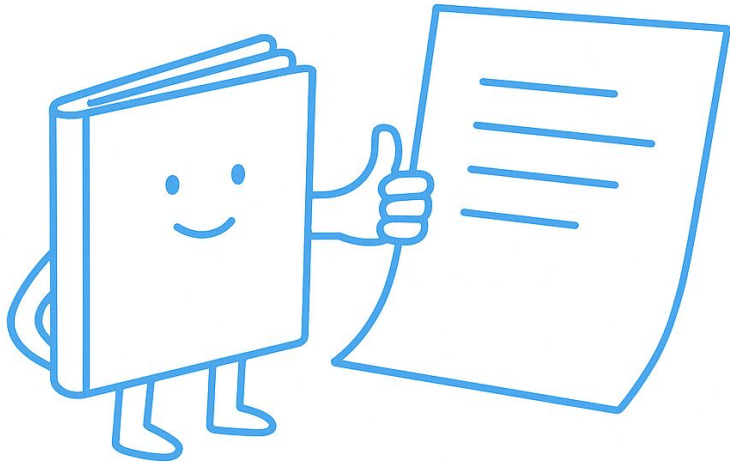
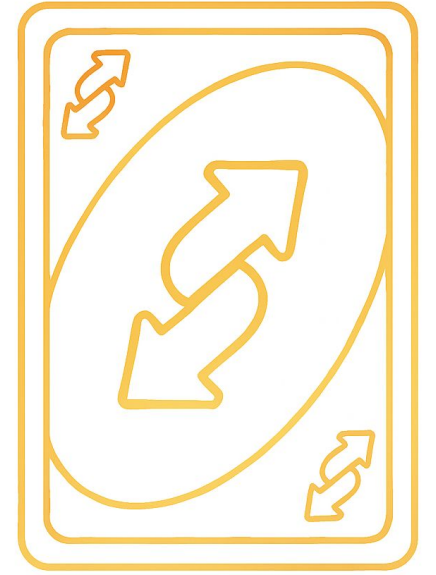
- C (POSIX): pthreads
 - create a thread with `pthread_create()` function
 - output parameter, address of thread
 - attributes (usually NULL)
 - function to be executed
 - pointer to function arguments
 - thread function is `void*` return type
- C++: threads and jthreads
 - create a thread using a constructor
 - function to be executed
 - function arguments (if they exist)
 - thread function is `void` return type
 - need to pass in refs using `std::ref()`, not `&`

← much less going on!

C++ Threads

- `std::thread`
 - C++11
 - parent thread needs to manually call `join()`
- `std::jthread`
 - C++20 - implemented off `std::thread`
 - automatically joins using `jthread`'s destructor

Locks



Locks

- The locks we've seen so far:
 - `pthread_mutex`
 - `std::mutex`
 - `std::scoped_lock`
 - `std::unique_lock`
- other stuff:
 - `std::condition_variable`
 - `wait(unique_lock& lock)`
 - `notify_one()`
 - `notify_all()`
 - `std::atomic`

Locks

- C (pthread) mutex versus `std::mutex`
 - very similar: `lock()`, `trylock()`, `unlock()`
 - `std::mutex` is a wrapper around `pthread_mutex*`
 - generally `std::mutex` is more preferred to use, unless you're working with pthreads
- `std::scoped_lock` and `std::unique_lock`
 - used to surround critical section in a "local scope"
 - lock is acquired at construction, released on destruction (when program leaves the local scope)
 - `unique_lock` can call `lock()` and `unlock()`
 - need `unique_lock` for condition variables

When do we use what?

Most generally, resources introduced in this unit are introduced to prevent **data races**.

Circle all that apply:

1. we would use a mutex to:
 - a. prevent race conditions
 - b. prevent data races
 - c. ensure exclusive access to a shared resource
 - d. synchronize two or more threads
2. we would use a condition variable to:
 - a. prevent race conditions
 - b. prevent data races
 - c. ensure exclusive access to a shared resource
 - d. synchronize two or more threads

Using Locks

1. identify your critical sections
 - a. lock at the start of critical section
 - b. unlock at end of critical section
2. identify branches in control flow
 - a. error-checking conditionals
 - b. exception handling
 - c. if a program can exit a critical section in different locations, you need to call `unlock()` at all of them!
3. consider dependencies on another thread's work
 - a. can one thread acquire a lock and then get stuck waiting for another thread? -> introduce a condition variable
 - b. eg. producer consumer problem

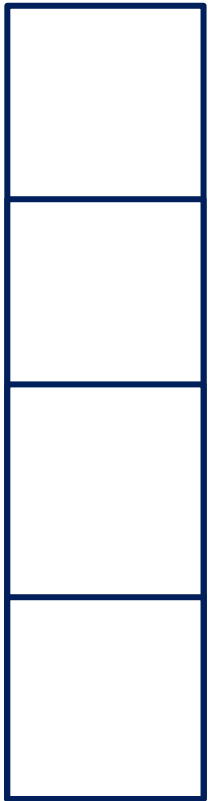
Example: Image Processing

I went on a trip to New York recently, and took many photos. I want to process them by blurring and then making them greyscale before I post them to my photography account on Twitter.

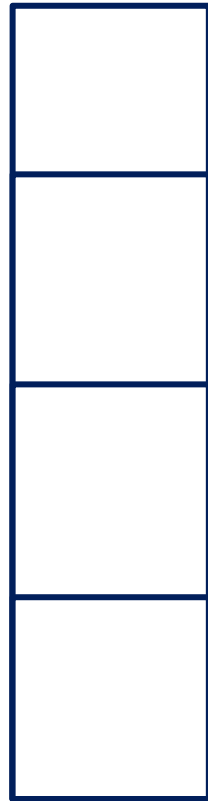
I've written two functions so far (implementation not included) using an Image object:

```
void blur(Image img);  
void grey(Image img);
```

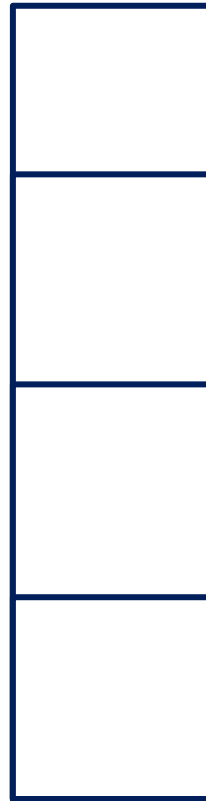
Image Processing: Data Structures



unprocessed



blurred



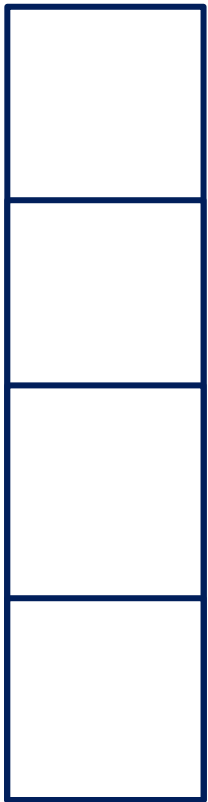
blurred and
greyscale

images start in unprocessed queue.

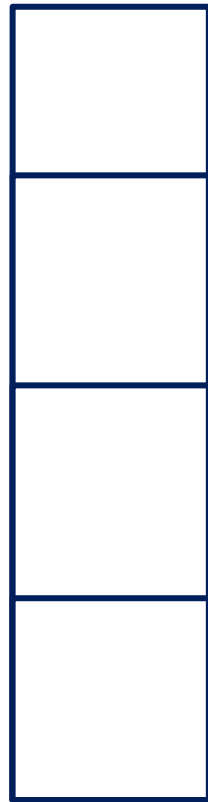
`blur()` : blur image in the unprocessed queue, then move into blurred queue

`grey()` : convert image in blurred queue to greyscale, then move to the last queue.

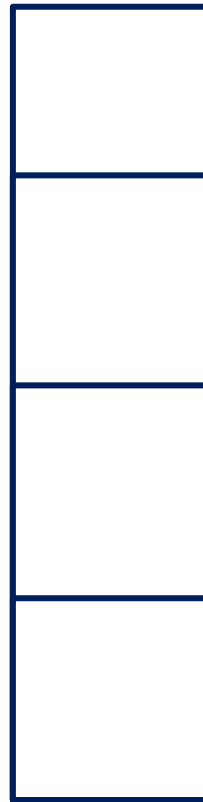
Does this work? Is this optimal?



unprocessed



blurred



blurred and
greyscale

```
void grey_img() {  
    while(true) {  
        counterLock.lock();  
        if (remainingImages == 0) {  
            break;  
        }  
        grey(unprocessed[remainingImages]);  
        remainingImages -= 1;  
        counterLock.unlock();  
    }  
}
```

Notes:

- grey() converts image in blurred queue to greyscale, then move to the last queue.
- 2 threads process blurring, and 2 threads process greyscale

The image features a vibrant red background with a series of concentric circles in varying shades of red and dark red, creating a tunnel-like effect. In the center, a dark blue circle serves as a backdrop for the text. The text "That's all Folks!" is written in a white, elegant script font, with a slight shadow effect that makes it stand out against the blue circle and the red background.

That's all Folks!