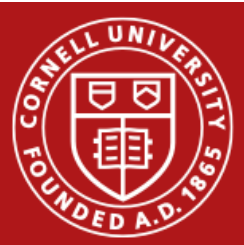
Project 1
Non-Preemptive Multitasking

Ege Mihmanli

Department of Computer Science
Cornell University

September 2, 2016



First things first

- Welcome to PortOS
- Project 1 is already released!
- Due on September 19th at 11:59pm

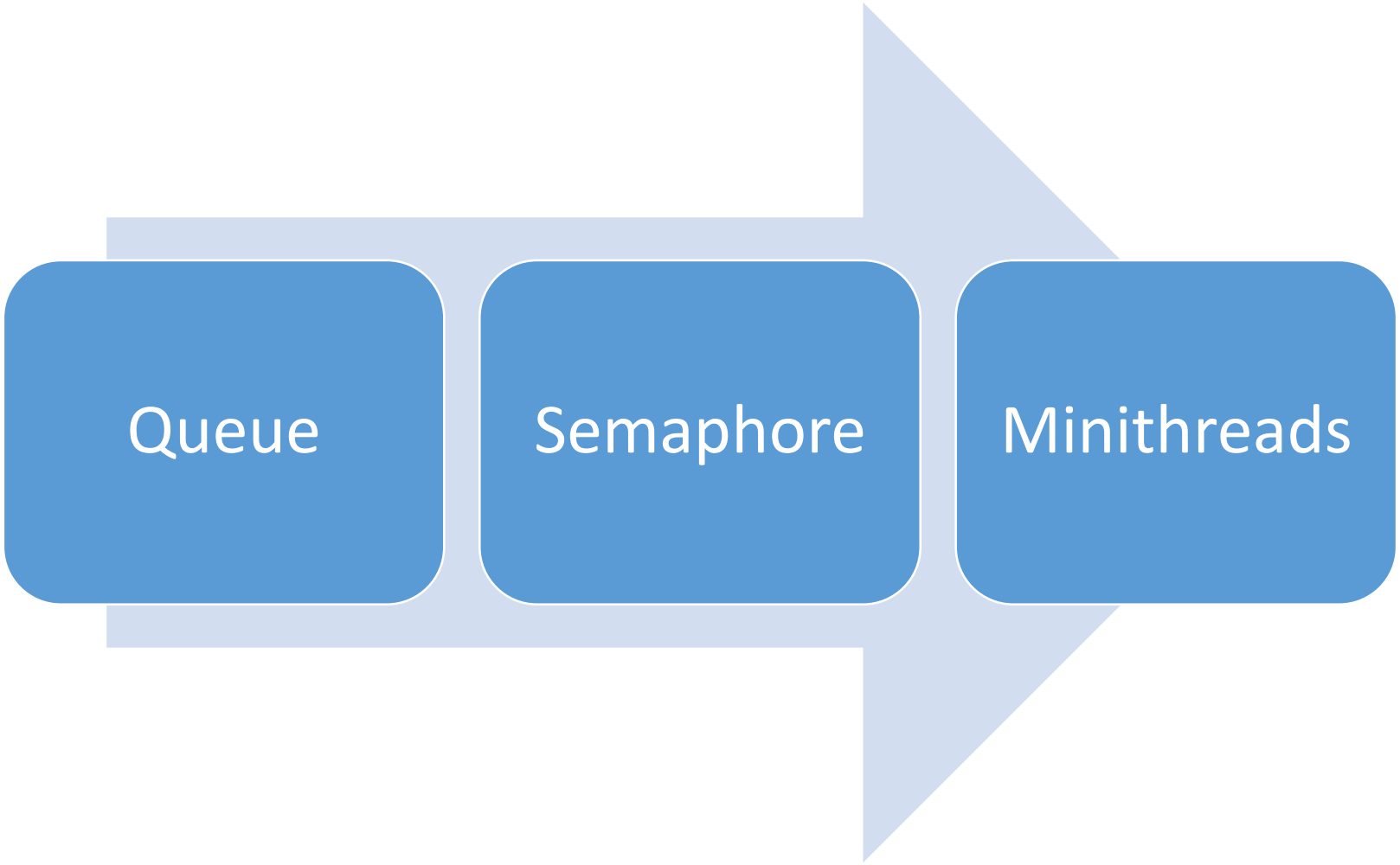
GitHub

- We are using github.coecis.cornell.edu
- Sign in with Cornell credentials, i.e netID and password
- Projects released and submitted on GitHub

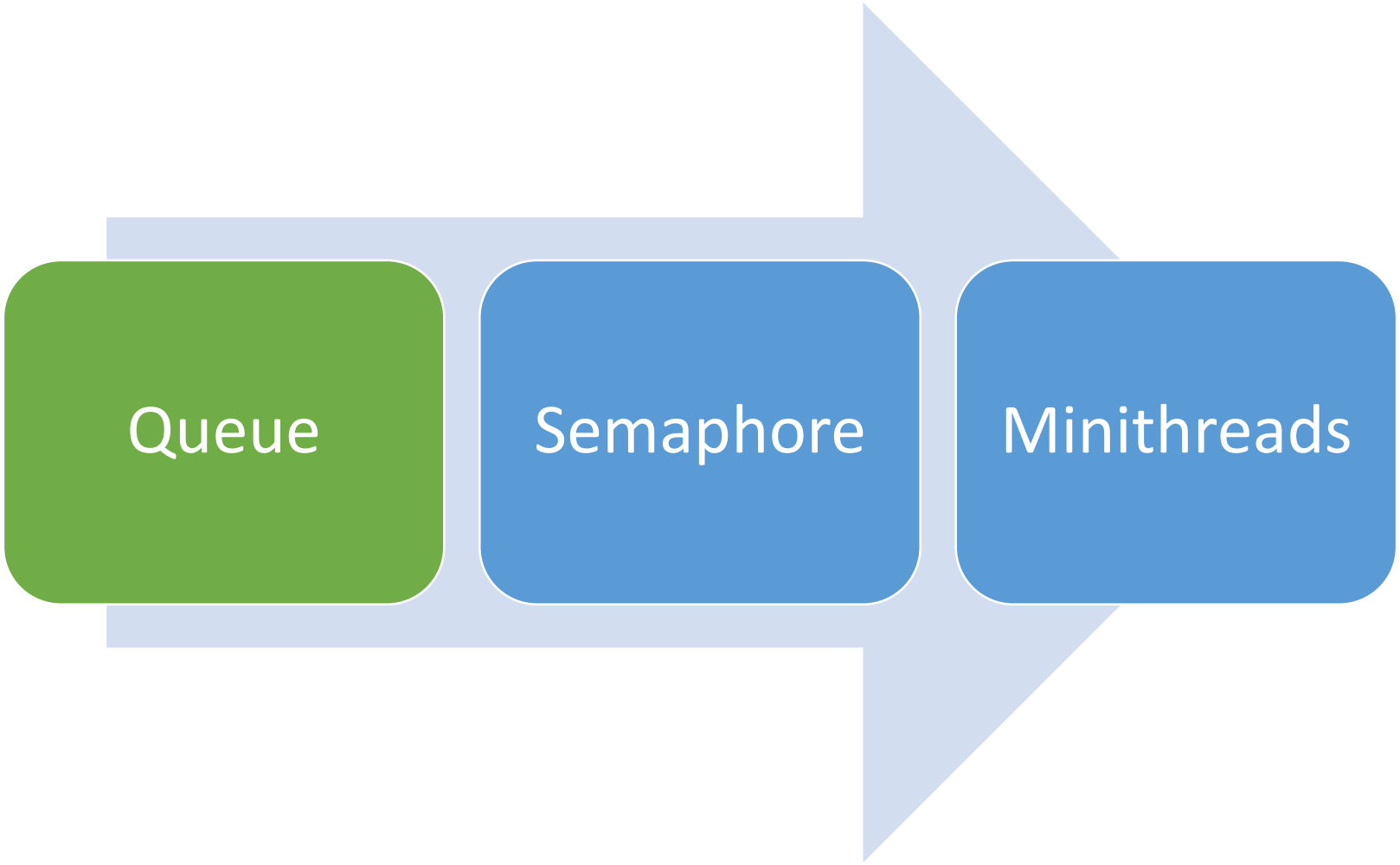
Goals

- Ramp up for C and PortOS
- Learn how threading works
- Implement synchronization primitives
- Large project ➔ bad coding style WILL bite later

Project Overview



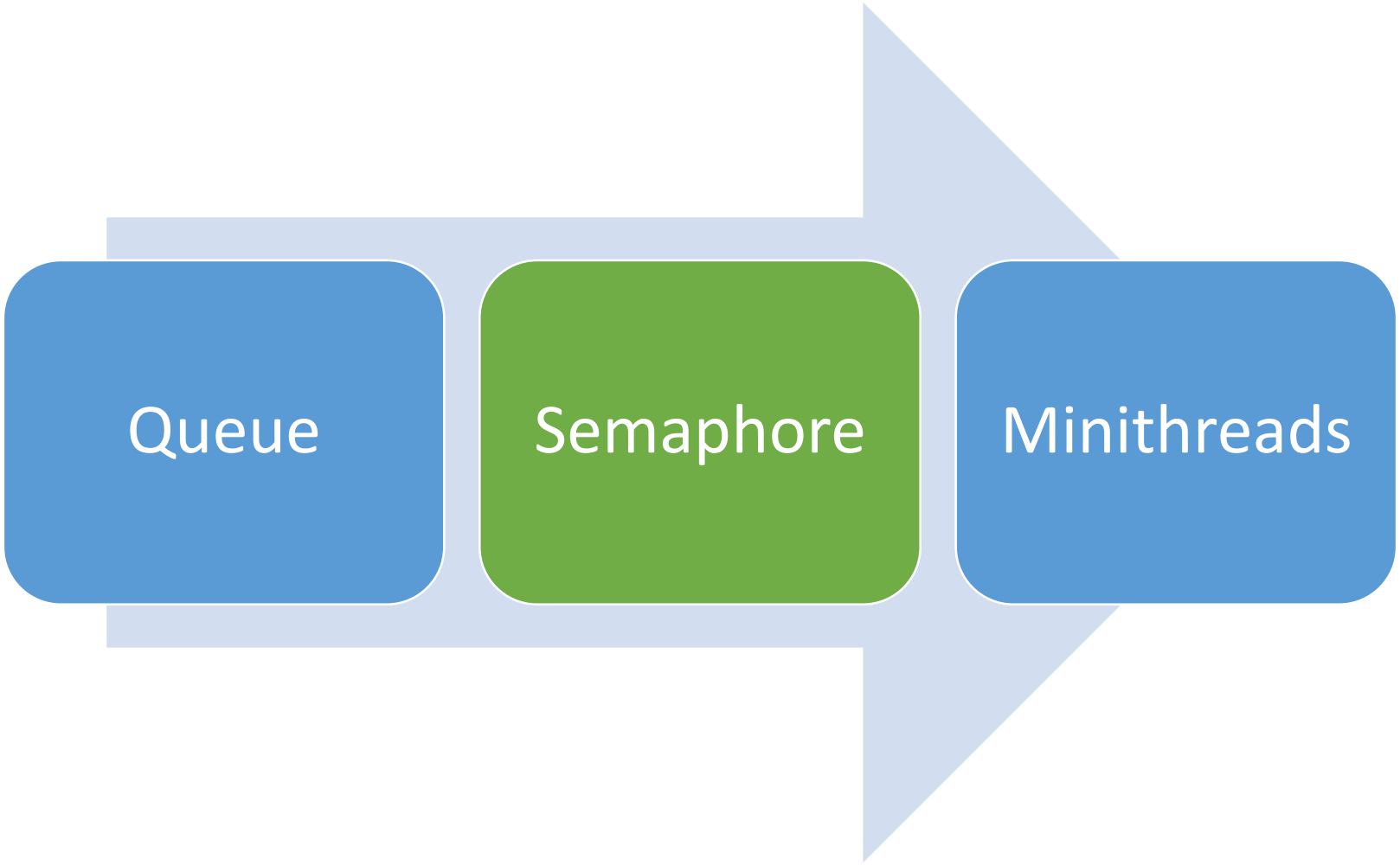
Project Overview



Queues

- Simple FIFO Queue
- Interface described in queue.h
- Use a linked list under the hood
- Prepend, append and dequeue must be **O(1)**

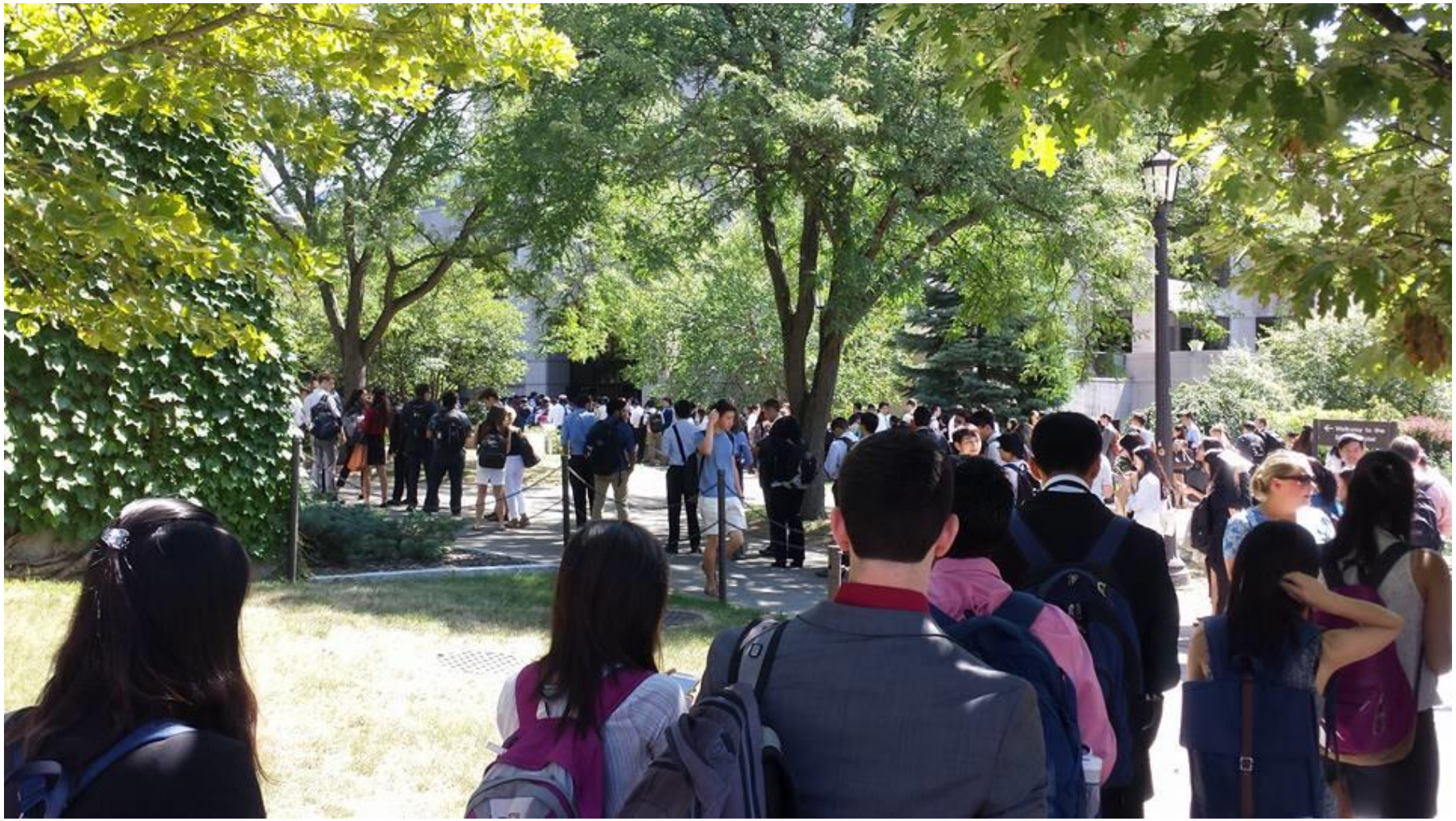
Project Overview



What is a semaphore?

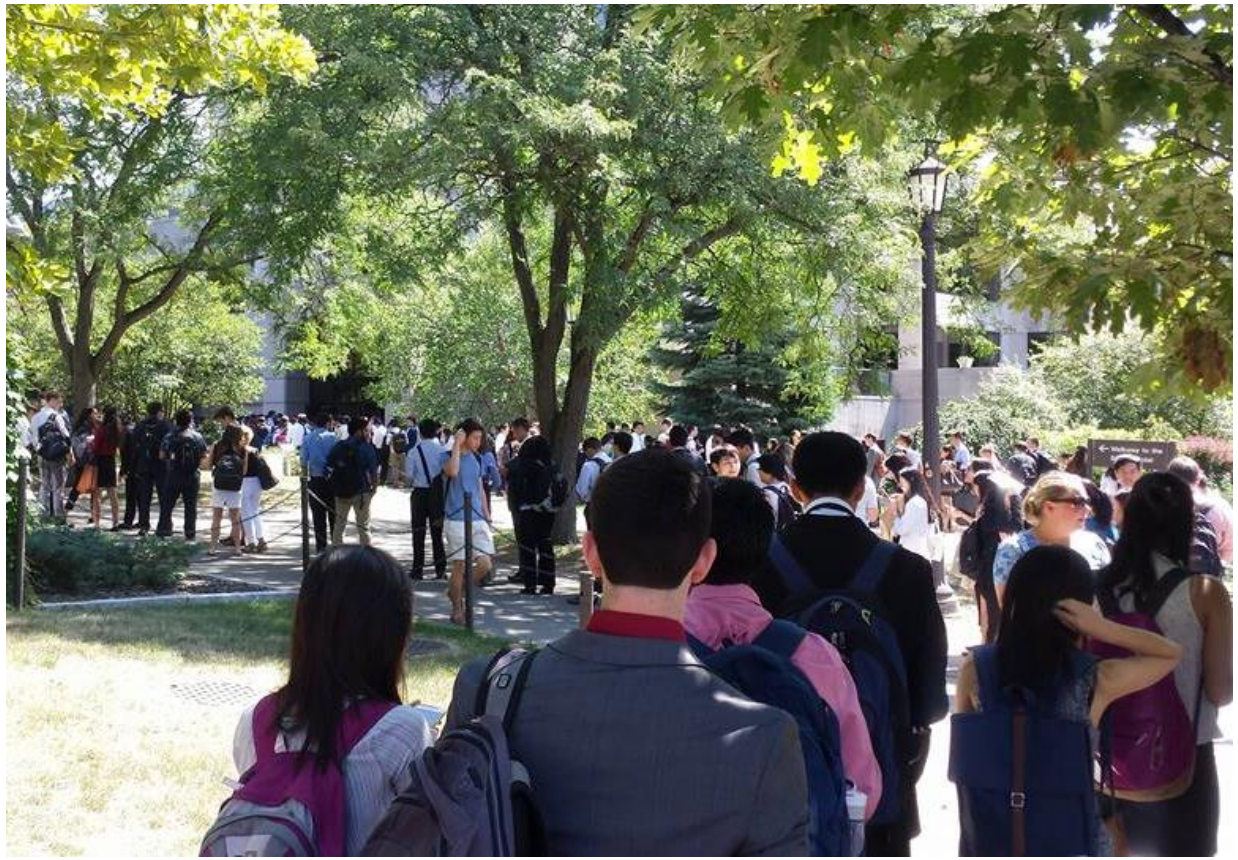
- Pillar of concurrent programming
- Actually, just another data structure
- Keeps a count
- Blocks/wakes up threads depending on situation

This is a semaphore



Let's make the analogy work

- Students → Threads
- Bouncers → Semaphore
- Legal max capacity→ Count
- Room space → Shared resource
- Line → Blocked threads



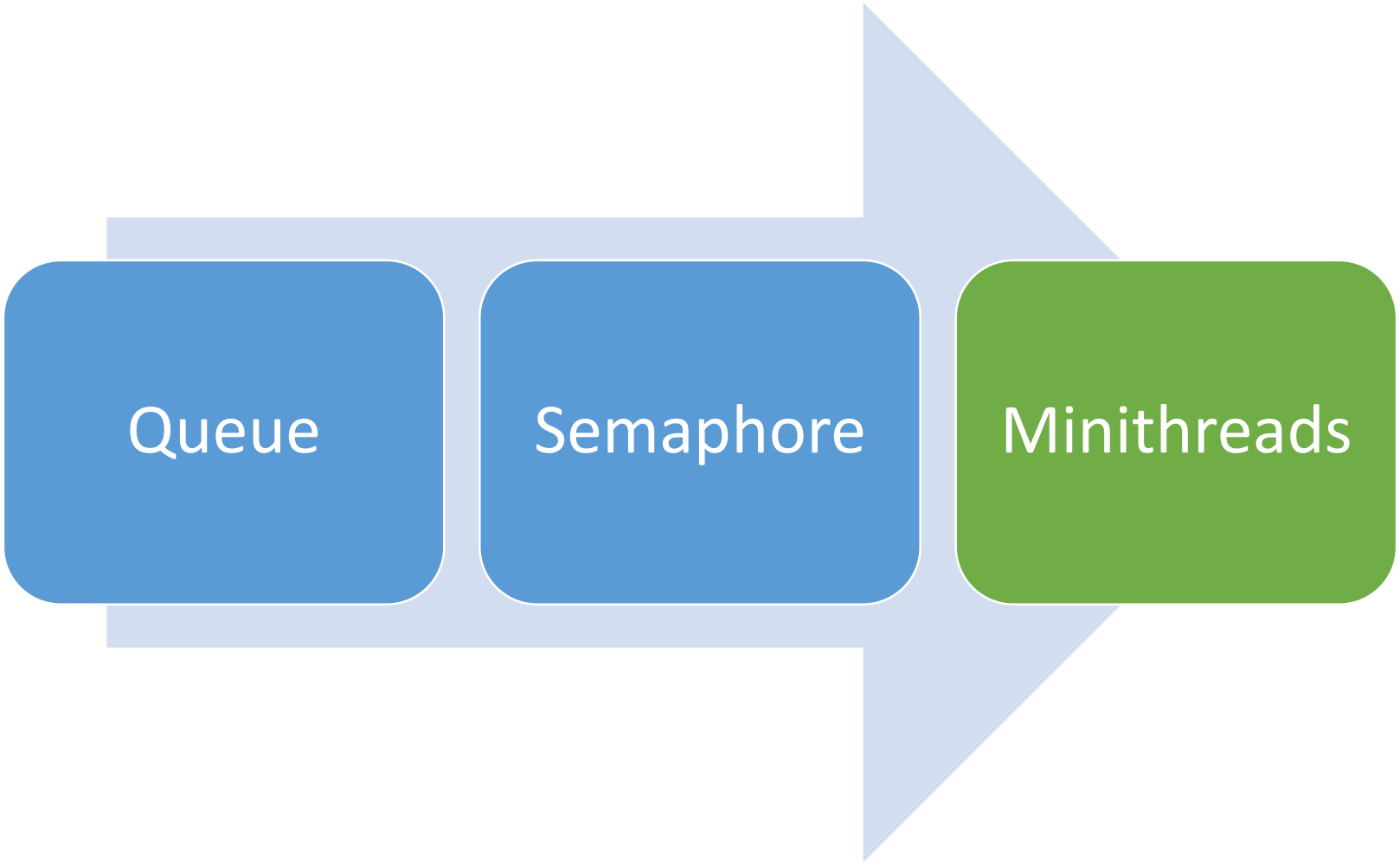
Concurrency 101

- Client decides how many threads can hold a semaphore (count)
- Counter is incremented/decremented atomically
    - P ↓ & V ↑
- P blocks if count == 0
- V wakes up blocked thread if count == 0

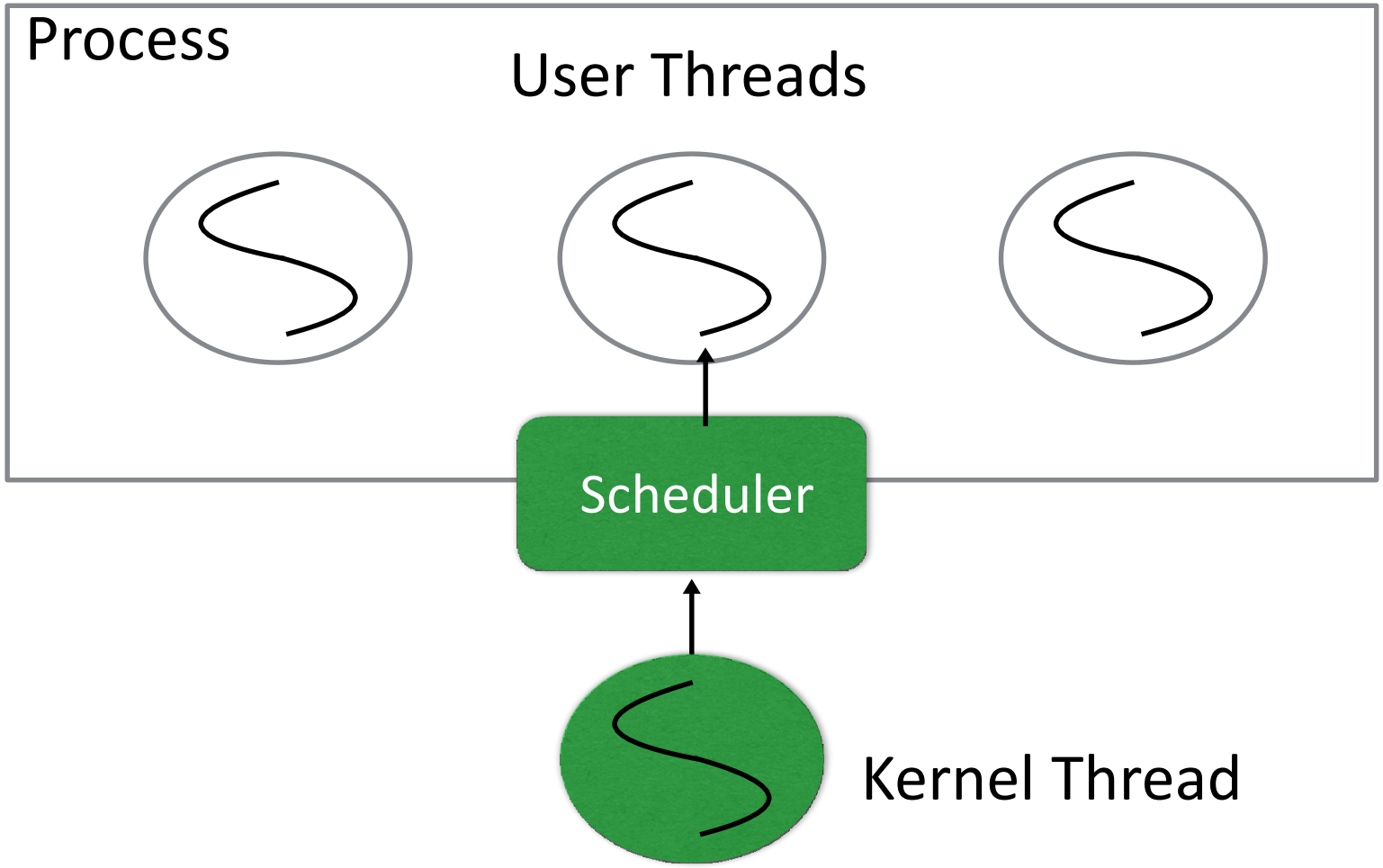
career_fair.c

```
take_shower();
get_dressed();
sweat_a_lot_on_your_way_over();
semaphore_p(); //attempt to walk in
talk_to_employers();
exaggerate_resume();
get_swag();
semaphore_v(); //walk out
complain_about_career_fair();
```

Project Overview



Minithreads



Scheduler

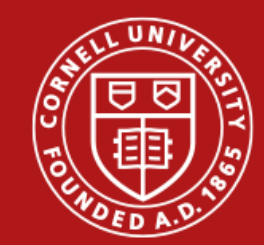
- First come first serve
- Just yield CPU to thread at the head of queue
- Expect this to get more complicated in Project 2
- Code style matters

Minithreads

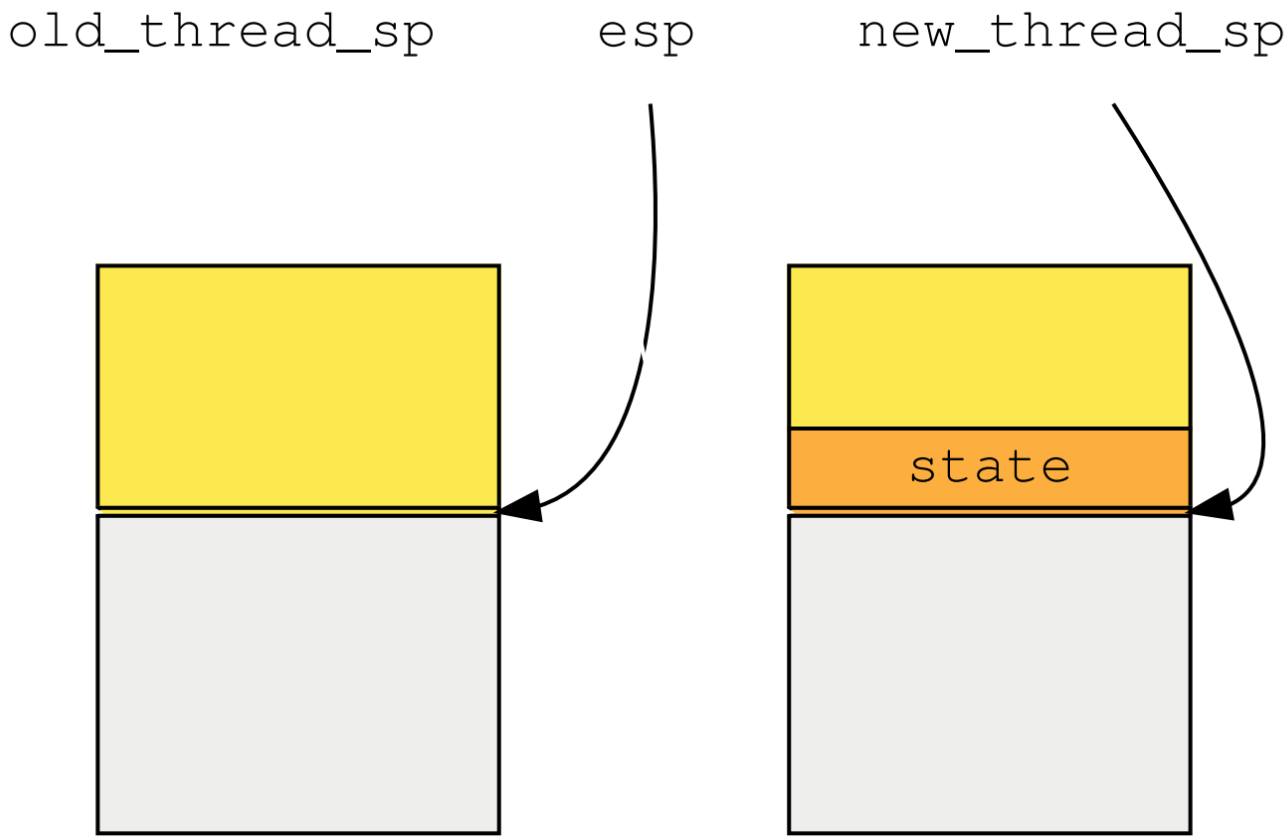
- What we call threads in PortOS
- Majority of the project
- Will need a Thread Control Block
  - Stack top pointer
  - stack base pointer
  - thread ID
  - Anything else you want

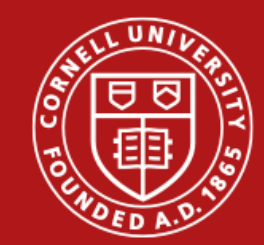
Useful functions

- We provide some functions we found useful
- Allocate stack → minithread_allocate_stack
- Initialize stack → minithread_initialize_stack
- Switching between threads → minithread_switch
- Make sure to read machineprimitives.h

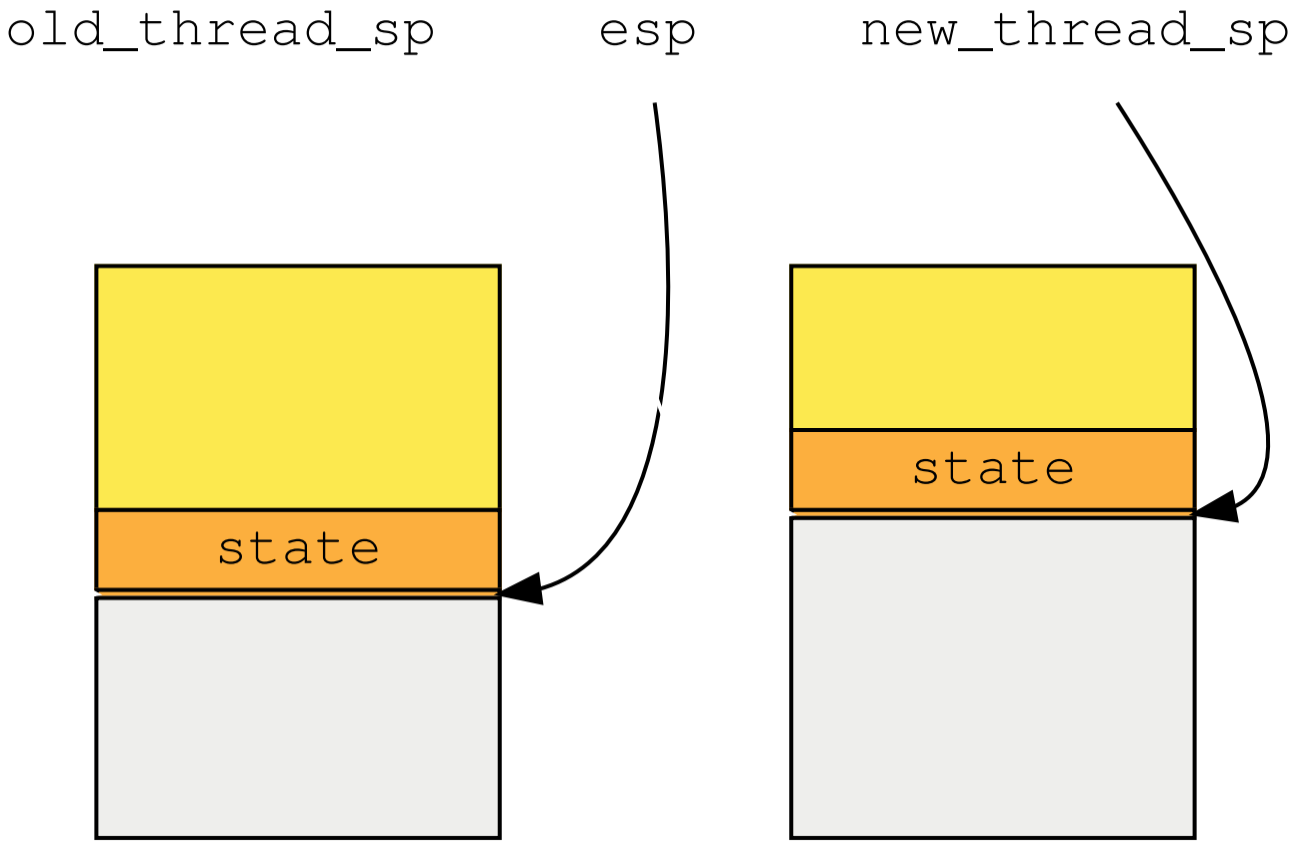


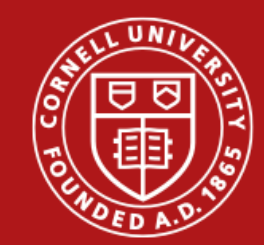
minithread switch



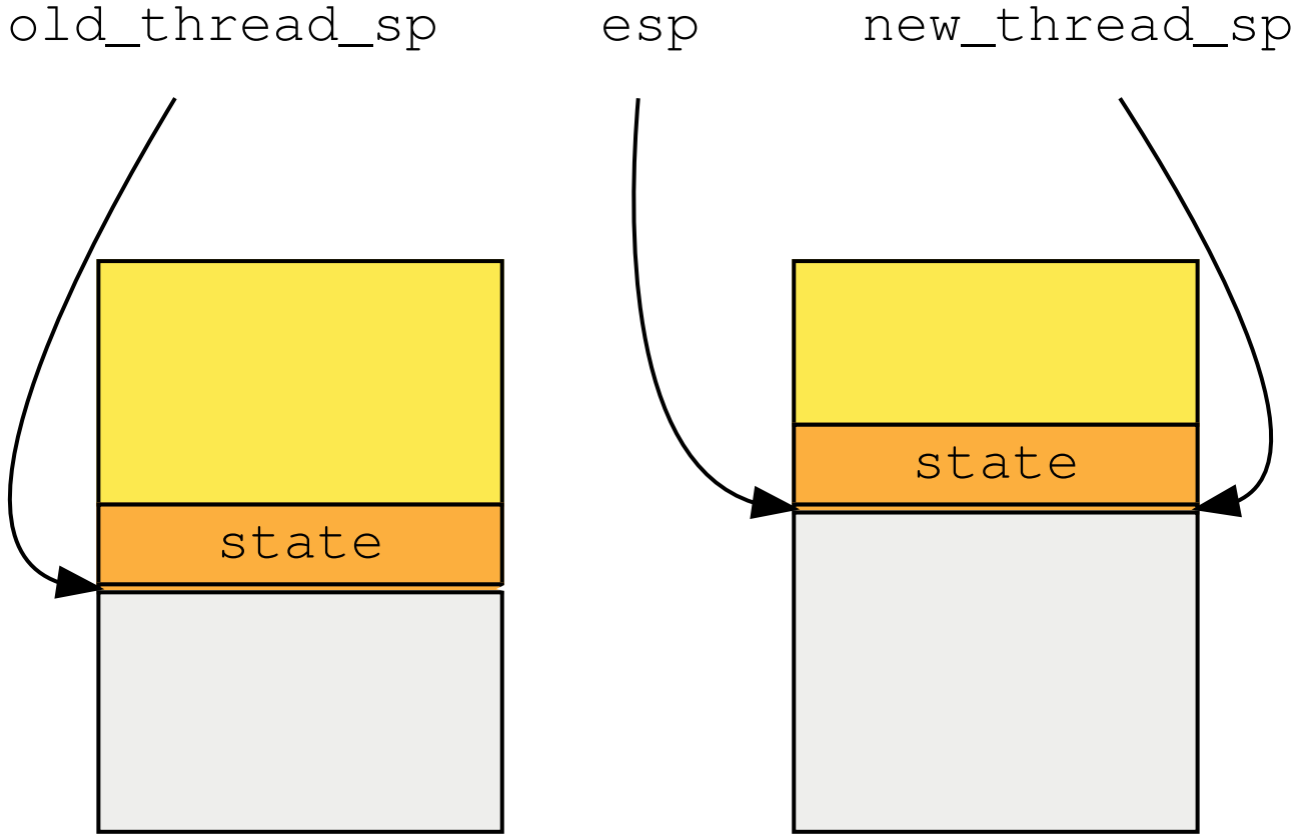


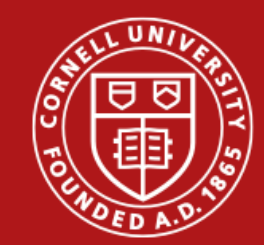
minithread_switch



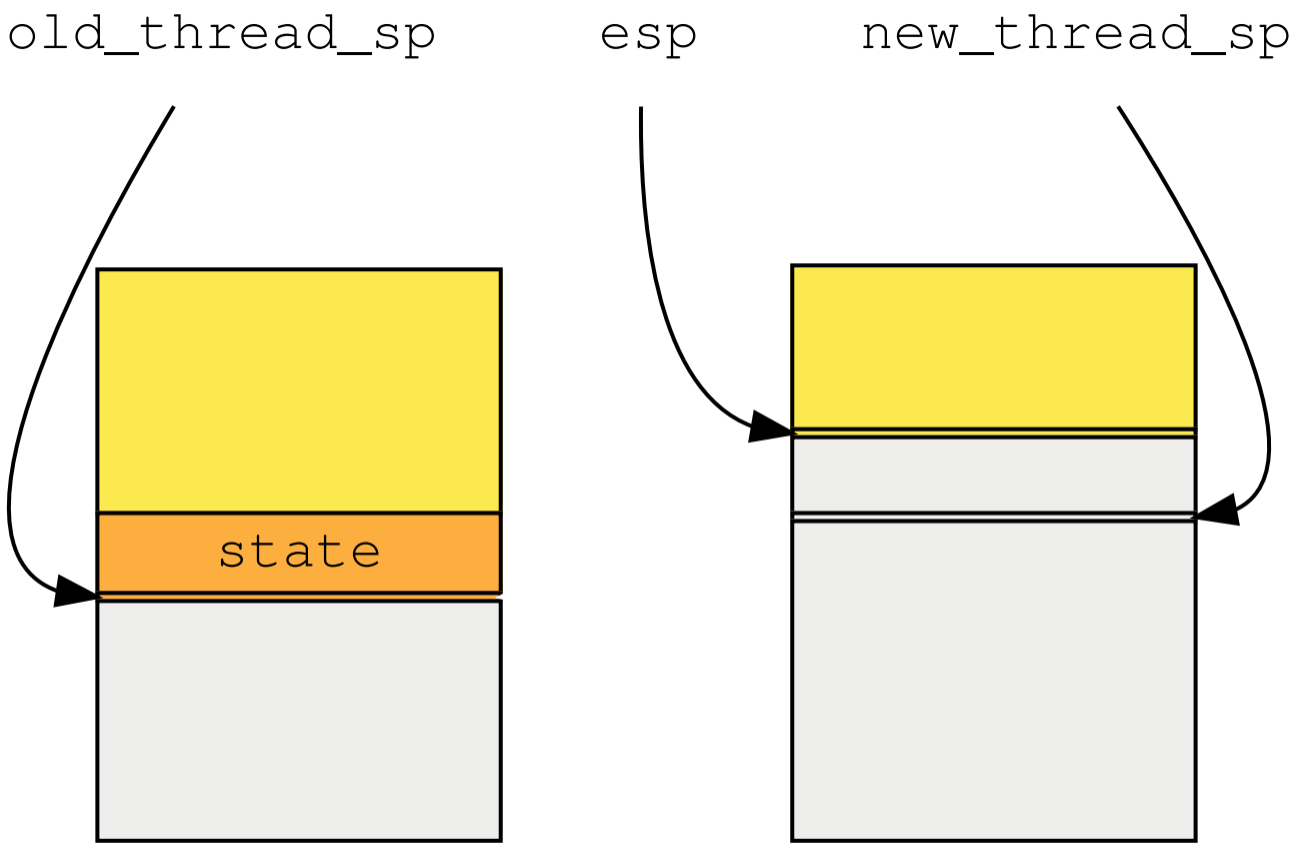


minithread_switch





minithread_switch



Bootstrapping

```
void minithread_system_initialize
```

- This bootstraps the system
- Use it to initialize queues, semaphores, global variables or data structures
- You will add more in projects to come

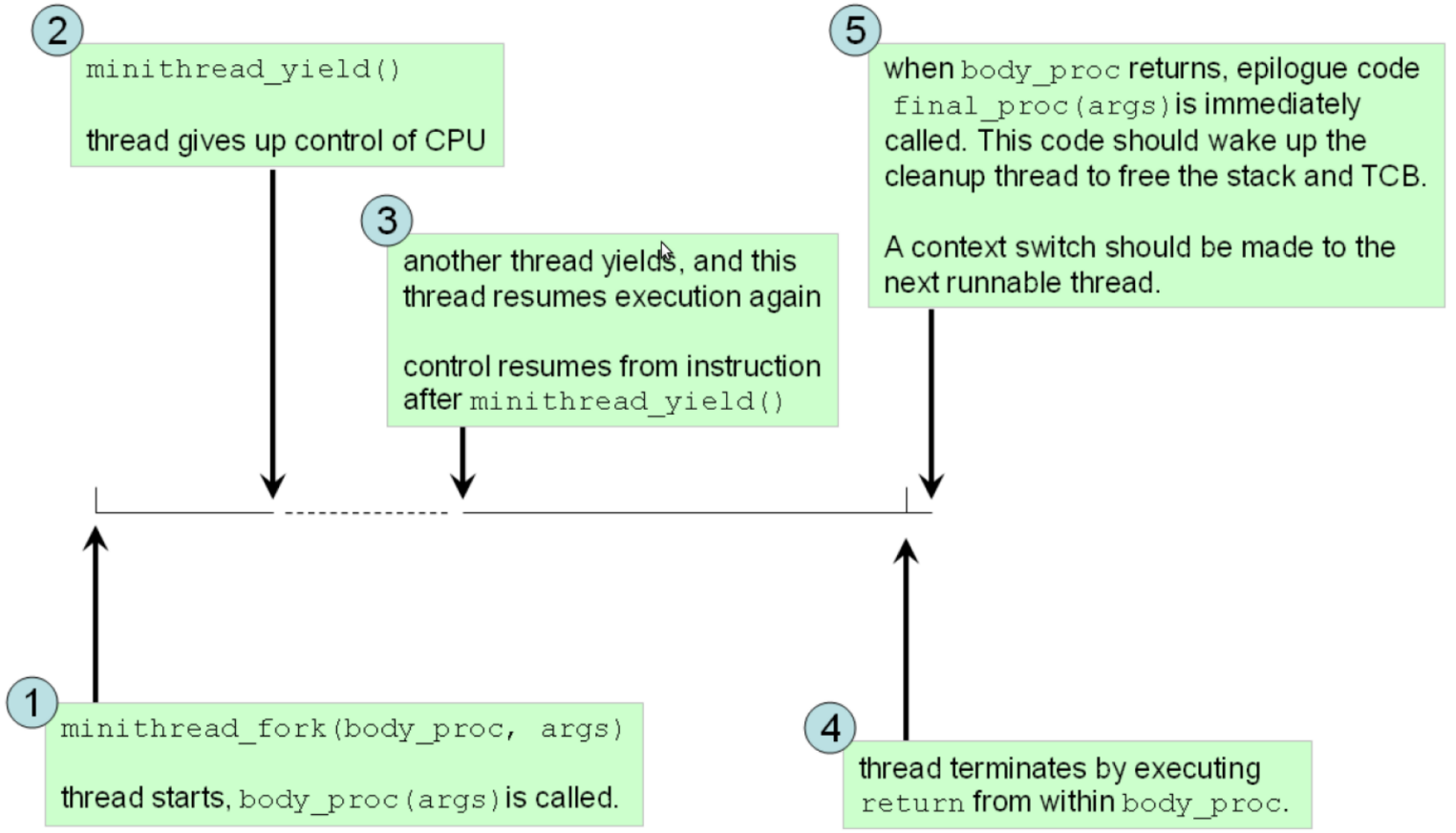
Bootstrapping

- What happens when there is no user thread left?
  - System shouldn't crash! It's an operating system
  - Run the **idle thread**
  - Only place where polling is OK!
- In our case, the kernel thread is the idle thread
- No need to allocate stack for it

Being Non-Preemptive

- What happens when a user thread runs forever?
  - In P1, we let it be!
- Assume that all threads are **good** and voluntarily yield
  - Threads yield by calling minithread_yield

Life of a minithread



Testing

- We supply a few primitive tests
  - Use it to see how minithreads work
- Sieve and buffer are good stress tests
- Remove **ALL** of your print statements and dead code before submission!

Coding Style

- Avoid unnecessary polling

```
while     (condition == False)
     minithread_yield();
```

- Unnecessary context switches are bad for you

- Check for NULL arguments! (malloc can return NULL)

Commenting

- Helps us understand your code
- Helps you understand your code
- Helps you notice bugs
- Helps us give partial credit for buggy cod
- Notice all the “helps”? Commenting is good!

Coding Style

- Naming convention is important
  - Underscores to delimit words:
    - minithread_switch
    - number_of_eges
  - Constants in ALL_CAPS

Coding in C

- Can't really say "I know C" without mastering pointers

```
int *int_ptr = (int*) malloc(sizeof(int));

int_ptr = 5;
```

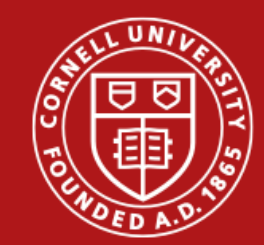
- What does this do?

Files you need to change

- queue.c/h
- synch.c/h
- minithread.c/h
- Important: you **don't have to** change header files!
- **DO NOT CHANGE ANY OTHER FILE**

```
12  int thread3(int* arg) {
13    printf("Thread 3.\n");
14
15    return 0;
16  }
17
18  int thread2(int* arg) {
19    minithread_fork(thread3, NULL);
20    printf("Thread 2.\n");
21    minithread_yield();
22
23    return 0;
24  }
25
```

```
26  int thread1(int* arg) {
27    minithread_fork(thread2, NULL);
28    printf("Thread 1.\n");
29    minithread_yield();
30    minithread_yield();
31
32    return 0;
33  }
34
35  int
36  main(int argc, char * argv[]) {
37    minithread_system_initialize(thread1, NULL);
38    return 0;
39  }
40
```


CORNELL UNIVERSITY
FOUNDED A.D. 1865

?