McGill

ECSE 420
Lab 1: Pthreads

TA: Loren Lugosch

September 2016

Outline

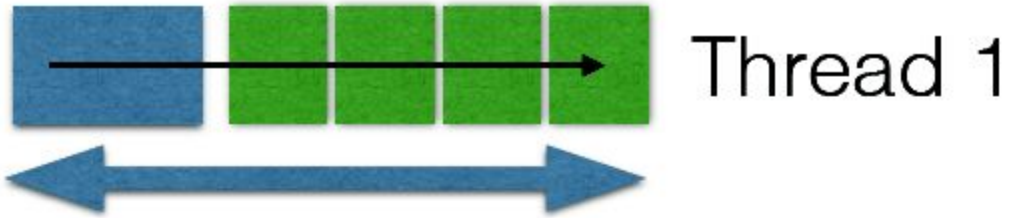
- Assignment 1
- Pthreads
- Lab 1

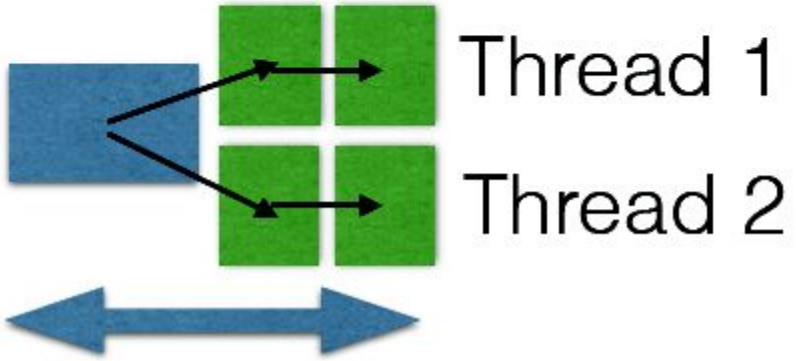
Assignment 1

- How is it going?

Pthreads

- Review: since CPUs have multiple cores, we can use multiple threads to get speedup





Pthreads

- Pthreads = POSIX threads (POSIX is a cross-platform OS API)
- Framework for writing multithreaded programs
- Today we will cover the following in Pthreads:
  - Creating threads
  - Joining threads
  - Mutexes

Pthreads

- To create a thread, use

```
pthread_create(pthread_t *thread,
pthread_attr_t *attr, void
*(*start_routine)(void *), void *arg);
```

- `pthread_t *thread`: pointer to the thread
- `pthread_attr_t *attr`: attributes to apply to this thread
- `void *(*start_routine)(void *)`: the function this thread executes
- `void *arg`: arguments to pass to thread function above

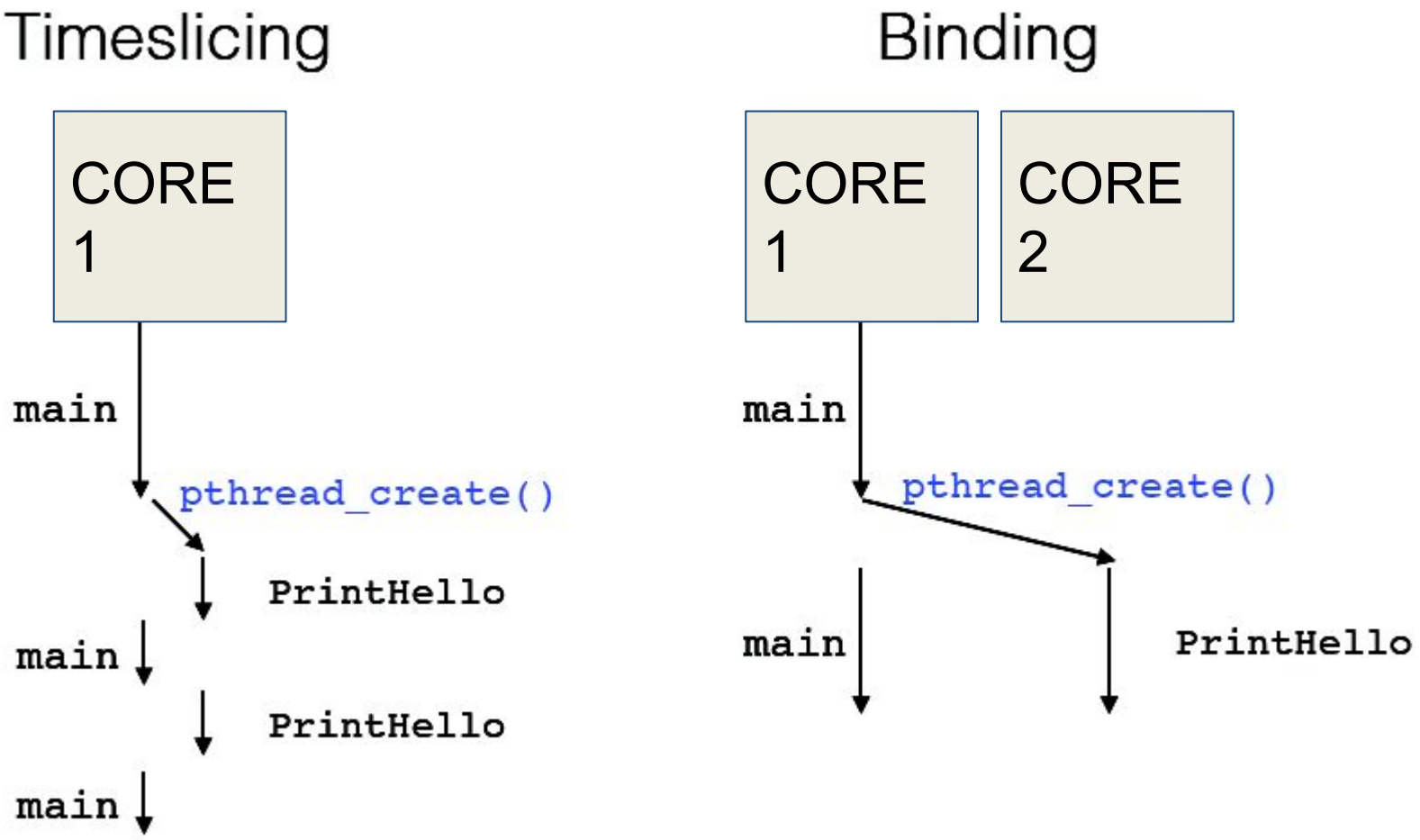
Pthreads

- Example:

```
void *PrintHello(void *arg)
{
    printf("Hello World!");
    pthread_exit(NULL);
}

...
// in main:
pthread_create(&thread, NULL, PrintHello, NULL);
```

Pthreads



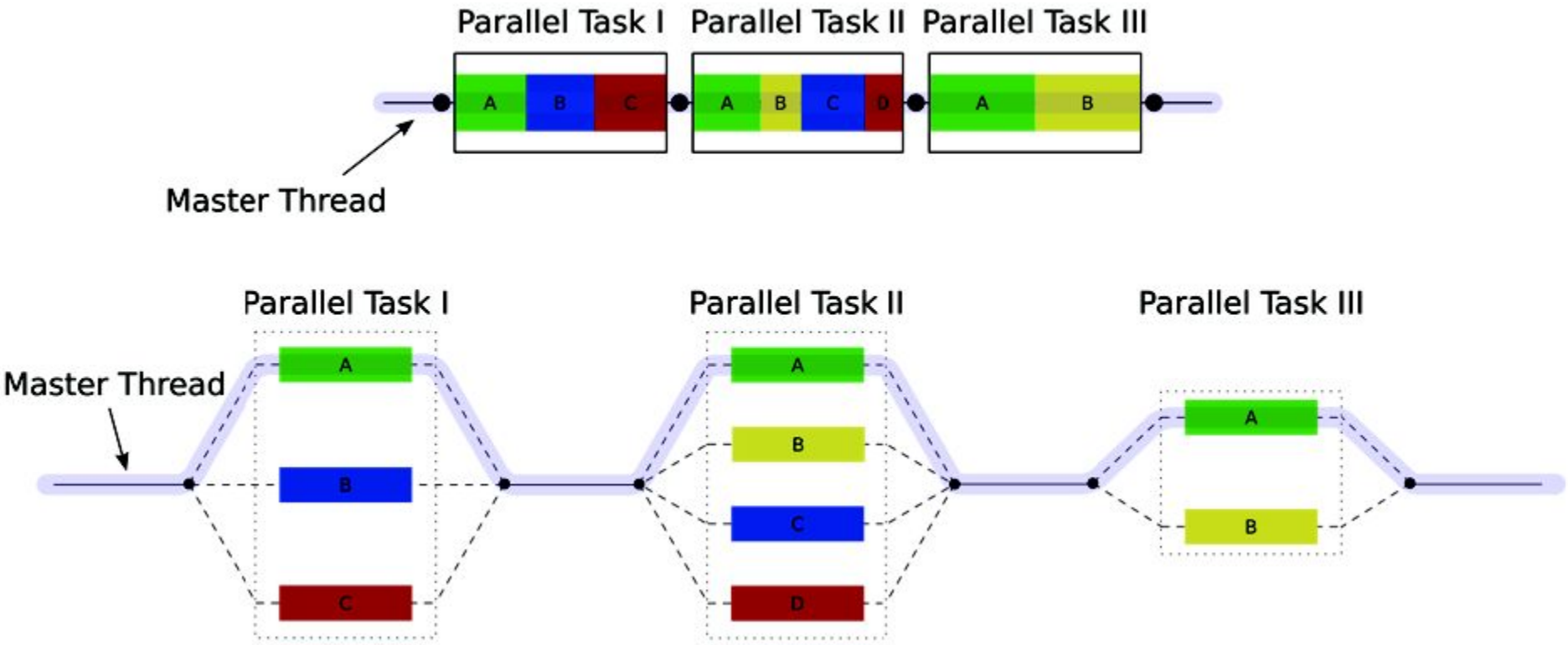
Pthreads

- **Joining** a thread = waiting for a thread to complete
- When thread X joins thread Y, X sleeps (= changes scheduler state) until it gets a signal from thread Y
- OS won't schedule a sleeping thread → therefore the thread waits
- A thread can only join one thread at a time
- It's a good idea for the main() thread to join all threads-- otherwise, main() can finish before the threads it creates, which can cause problems

Pthreads

- Pass arguments using a struct
- Example:

Pthreads



Pthreads

- Consider the following code:

```
count++;
```

Compiles to something like this:

```
LOAD R1,R2 //R2 address of "count"
ADD R1,R1,1
STORE R1,R2
```

Pthreads

- Suppose that initially `count == 0` and two threads run this code at the same time:

Thread1

```
LOAD R1,R2
ADD R1,R1,1
STORE R1,R2
```

Thread2

```
LOAD R1,R2
ADD R1,R1,1
STORE R1,R2
```

- What is the value of `count` after the code runs?

Pthreads

- This is called the reader-writer problem
- Solution: use a **mutex**
  - Lock the mutex when you need to use a shared resource, then unlock the mutex when done
  - If the mutex is already locked, sleep until unlocked

```
pthread_mutex_t lock_count;
pthread_mutex_init(&lock_count, NULL);
```

```
pthread_mutex_lock(&lock_count);
... write to shared variable ...
pthread_mutex_unlock(&lock_count);
```

Pthreads

- Some good references on Pthreads:
- https://courses.engr.illinois.edu/cs241/fa2010/ppt/10-pthread-examples.pdf
- http://randu.org/tutorials/threads/

Lab 1

- In the first lab, you will
  - write some image processing algorithms
  - multithread using Pthreads or OpenMP (covered next week)
  - measure speedup
- Remainder of tutorial: I will stay and help
- SSH into the Trottier servers-- your program must compile and run on these servers!
- <firstname>.<lastname>@mail.mcgill.ca @TR5130GU-<1 to 15>.ECE.McGill.CA