

Parallel Execution | Threads

Thread Basics

- ◆ A thread uses serial execution
 - Each line of code is executed one at a time
- ◆ Multicore CPUs can have multiple threads
 - Threads still executes serially
 - Each thread can execute different tasks
 - ▶ Better CPU utilization
- ◆ Threads are isolated from one another
 - Require additional work to communicate
 - ▶ Should communicate infrequently for performance reasons

Working With Threads

- ◆ Threads are “spawned” (created)
 - Threads can spawn threads
 - Use the “main” thread for spawning in most cases
 - ▶ *fn main()* is the main thread
- ◆ Code is no longer executed line-by-line with threads
 - Requires careful planning
- ◆ When a thread completes work, it should be “joined” back into the main thread
 - Ensures that the thread has completed

Thread Execution

```
println!("1");  
println!("2");  
println!("3");
```

1 2 3

```
println!("A");  
println!("B");  
println!("C");
```

A B C

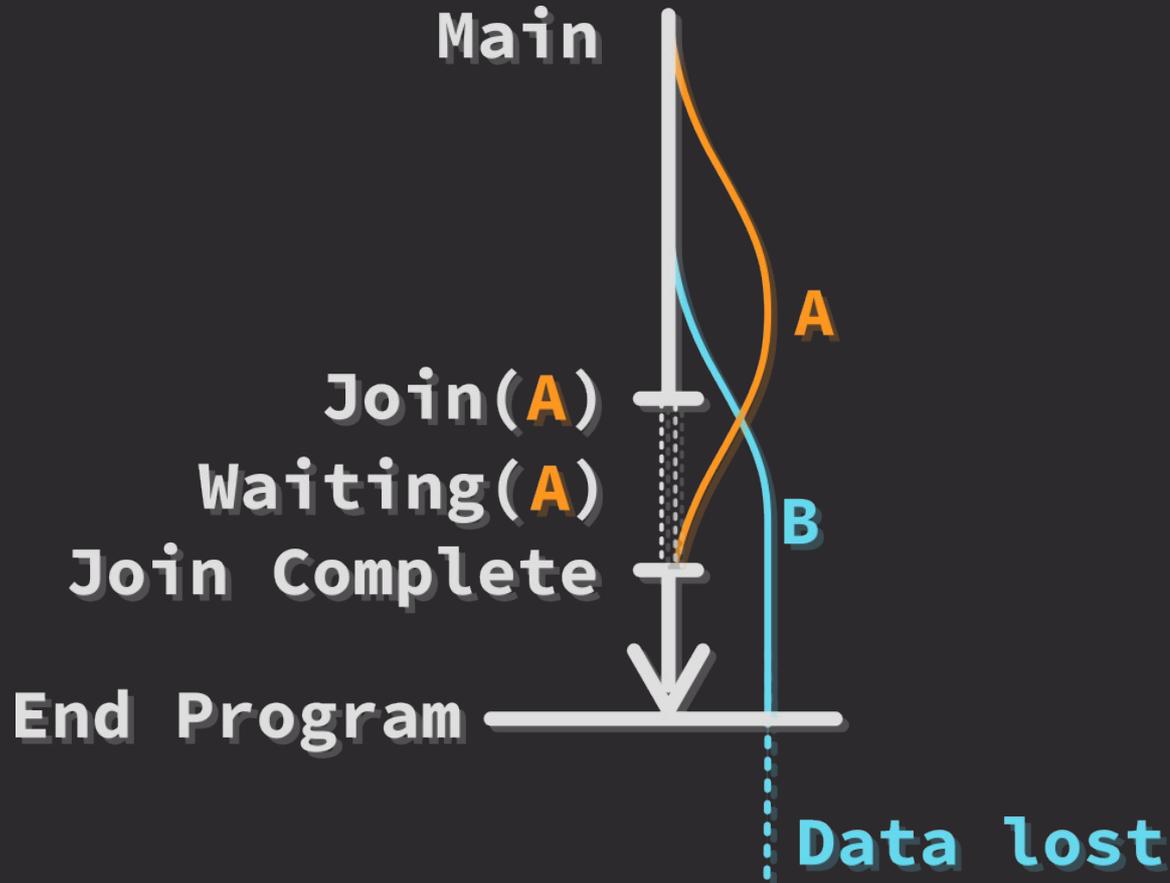
1 2 3 A B C

A B C 1 2 3

1 A 2 B 3 C

1 2 A B C 3

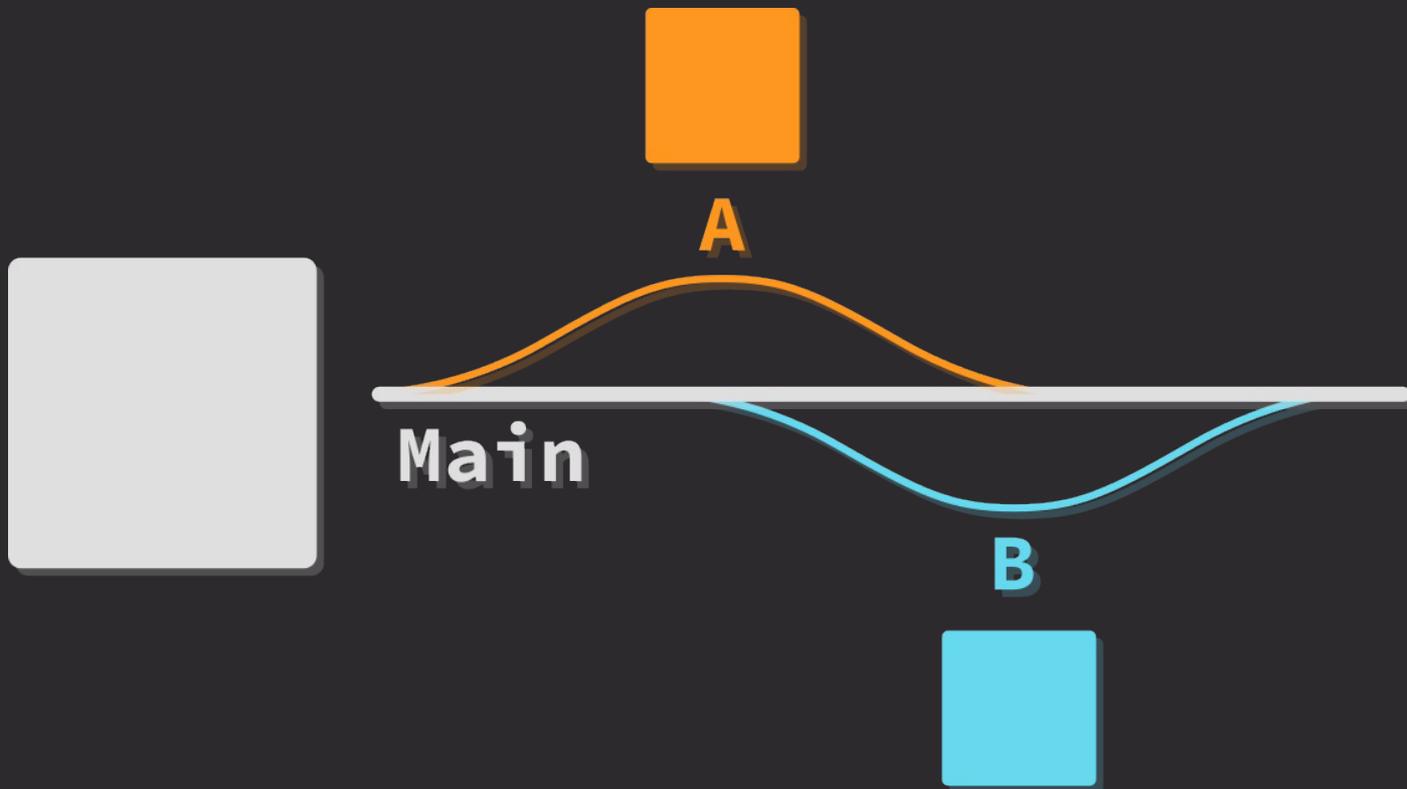
Thread Lifetime



■ Thread Memory

- ◆ Threads have “thread-local” memory
 - Owned by the thread
 - Only accessible in the thread
- ◆ Data can be copied or moved into threads
 - Can be done when thread created
 - Becomes thread-local

Thread Memory



■ Spawning a Thread

```
use std::thread;
let handle = thread::spawn(move || {
    // .. code ..
});
handle.join();
```

`JoinHandle<type>`

Recap

- ◆ Threads are non-deterministic
 - Execution order will vary each time the program runs
- ◆ Ending the main thread will terminate all spawned threads
 - **Join** on the main thread to wait for threads to complete
- ◆ Each thread has it's own chunk of memory